1. *do not* start with C_1,
2. *do not* end with an accepting configuration, or
3. where some C_i *does not* properly yield C_{i+1} under the rules of M.

If M does not accept w, no accepting computation history exists, so *all* strings fail in one way or another. Therefore G would generate all strings, as desired.

Now we get down to the actual construction of G. Instead of constructing G, we construct a PDA D. We know that we can use the construction given in Theorem 2.20 (page 115) to convert D to a CFG. We do so because, for our purposes, designing a PDA is easier than designing a CFG. In this instance, D will start by nondeterministically branching to guess which of the preceding three conditions to check. One branch checks on whether the beginning of the input string is C_1 and accepts if it isn't. Another branch checks on whether the input string ends with a configuration containing the accept state, q_{accept}, and accepts if it isn't.

The third branch is supposed to accept if some C_i does not properly yield C_{i+1}. It works by scanning the input until it nondeterministically decides that it has come to C_i. Next, it pushes C_i onto the stack until it comes to the end as marked by the # symbol. Then D pops the stack to compare with C_{i+1}. They are supposed to match except around the head position where the difference is dictated by the transition function of M. Finally, D accepts if it is a mismatch or an improper update.

The problem with this idea is that, when D pops C_i off the stack, it is in reverse order and not suitable for comparison with C_{i+1}. At this point the twist in the proof appears: We write the accepting computation history differently. Every other configuration appears in reverse order. The odd positions remain written in the forward order, but the even positions are written backward. Thus an accepting computation history would appear as shown in the following figure.

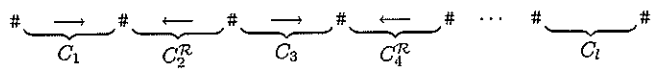


FIGURE **5.14**
Every other configuration written in reverse order

In this modified form, the PDA is able to push a configuration so that when it is popped, the order is suitable for comparison with the next one. We design D to accept any string that is not an accepting computation history in the modified form.

In Exercise 5.1 you can use Theorem 5.13 to show that EQ_{CFG} is undecidable.

5.2 A SIMPLE UNDECIDABLE PROBLEM

In this section we show that the phenomenon of undecidability is not confined to problems concerning automata. We give an example of an undecidable problem concerning simple manipulations of strings. It is called the ***Post correspondence problem***, or ***PCP***.

We can describe this problem easily as a type of puzzle. We begin with a collection of dominos, each containing two strings, one on each side. An individual domino looks like

$$\left[\frac{\texttt{a}}{\texttt{ab}}\right]$$

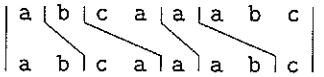
and a collection of dominos looks like

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

The task is to make a list of these 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***. For example, the following list is a match for this puzzle.

$$\left[\frac{\texttt{a}}{\texttt{ab}}\right]\left[\frac{\texttt{b}}{\texttt{ca}}\right]\left[\frac{\texttt{ca}}{\texttt{a}}\right]\left[\frac{\texttt{a}}{\texttt{ab}}\right]\left[\frac{\texttt{abc}}{\texttt{c}}\right].$$

Reading off the top string we get abcaaabc, which is the same as reading off the bottom. We can also depict this match by deforming the dominos so that the corresponding symbols from top and bottom line up.



For some collections of dominos finding a match may not be possible. For example, the collection

$$\left\{\left[\frac{\texttt{abc}}{\texttt{ab}}\right], \left[\frac{\texttt{ca}}{\texttt{a}}\right], \left[\frac{\texttt{acc}}{\texttt{ba}}\right]\right\}$$

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.

Before getting to the formal statement of this theorem and its proof, let's state the problem precisely and then express it as a language. An instance of the PCP

is a collection P of dominos:

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \cdots, \left[\frac{t_k}{b_k}\right] \right\},$$

and a match is a sequence $i_1, i_2, \ldots, i_l$, where $t_{i_1} t_{i_2} \cdots t_{i_l} = b_{i_1} b_{i_2} \cdots b_{i_l}$. The problem is to determine whether P has a match. Let

$$PCP = \{\langle P \rangle \mid P \text{ is an instance of the Post correspondence problem with a match}\}.$$

THEOREM 5.15

PCP is undecidable.

PROOF IDEA Conceptually this proof is simple, though it involves many technical details. The main technique is reduction from A_{TM} via accepting computation histories. We show that from any TM M and input w we can construct an instance P where a match is an accepting computation history for M on w. If we could determine whether the instance has a match, we would be able to determine whether M accepts w.

How can we construct P so that a match is an accepting computation history for M on w? We choose the dominos in P so that making a match forces a simulation of M to occur. In the match, each domino links a position or positions in one configuration with the corresponding one(s) in the next configuration.

Before getting to the construction we handle three small technical points. (Don't worry about them too much on your initial reading through this construction.) First, for convenience in constructing P, we assume that M on w never attempts to move its head off the left-hand end of the tape. That requires first altering M to prevent this behavior. Second, if $w = \varepsilon$, we use the string $\sqcup$ in place of w in the construction. Third, we modify the PCP to require that a match starts with the first domino,

$$\left[\frac{t_1}{b_1}\right].$$

Later we show how to eliminate this requirement. We call this problem the modified Post correspondence problem (MPCP). Let

$$MPCP = \{\langle P \rangle \mid P \text{ is an instance of the Post correspondence problem with a match that starts with the first domino}\}.$$

Now let's move into the details of the proof and design P to simulate M on w.

PROOF We let TM R decide the PCP and construct S deciding A_{TM}. Let

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}),$$

where Q, Σ, Γ, and δ, are the state set, input alphabet, tape alphabet, and transition function of M, respectively.

In this case S constructs an instance of the PCP P that has a match iff M accepts w. To do that S first constructs an instance P' of the MPCP. We describe the construction in seven parts, each of which accomplishes a particular aspect of simulating M on w. To explain what we are doing we interleave the construction with an example of the construction in action.

Part 1. The construction begins in the following manner.

$$\text{Put } \left[\frac{\#}{\#q_0 w_1 w_2 \cdots w_n \#}\right] \text{ into } P' \text{ as the first domino } \left[\frac{t_1}{b_1}\right].$$

Because P' is an instance of the MPCP, the match must begin with this domino. Thus the bottom string begins correctly with $C_1 = q_0 w_1 w_2 \cdots w_n$, the first configuration in the accepting computation history for M on w, as shown in the following figure.

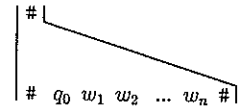


FIGURE 5.16
Beginning of the MPCP match

In this depiction of the partial match achieved so far, the bottom string consists of $\#q_0 w_1 w_2 \cdots w_n \#$ and the top string consists only of #. To get a match we need to extend the top string to match the bottom string. We provide additional dominos to allow this extension. The additional dominos cause M's next configuration to appear at the extension of the bottom string by forcing a single-step simulation of M.

In parts 2, 3, and 4, we add to P' dominos that perform the main part of the simulation. Part 2 handles head motions to the right, part 3 handles head motions to the left, and part 4 handles the tape cells not adjacent to the head.

Part 2. For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{\text{reject}}$,

$$\text{if } \delta(q, a) = (r, b, \text{R}), \text{ put } \left[\frac{qa}{br}\right] \text{ into } P'.$$

Part 3. For every $a, b, c \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{\text{reject}}$,

$$\text{if } \delta(q, a) = (r, b, \text{L}), \text{ put } \left[\frac{cqa}{rcb}\right] \text{ into } P'.$$

Part 4. For every $a \in \Gamma$,

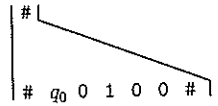
$$\text{put } \left[\frac{a}{a}\right] \text{ into } P'.$$

Now we make up a hypothetical example to illustrate what we have built so far. Let $\Gamma = \{0, 1, 2, \sqcup\}$. Say that w is the string 0100 and that the start state of M is q_0. In state q_0, upon reading a 0, let's say that the transition function dictates that M enters state q_7, writes a 2 on the tape, and moves its head to the right. In other words, $\delta(q_0, 0) = (q_7, 2, \mathrm{R})$.

Part 1 places the domino

$$\left[\frac{\#}{\#q_0 0100\#}\right] = \left[\frac{t_1}{b_1}\right]$$

in P', and the match begins:



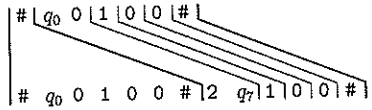
In addition, part 2 places the domino

$$\left[\frac{q_0 0}{2 q_7}\right]$$

as $\delta(q_0, 0) = (q_7, 2, \mathrm{R})$ and part 4 places the dominos

$$\left[\frac{0}{0}\right], \left[\frac{1}{1}\right], \left[\frac{2}{2}\right], \text{ and } \left[\frac{\sqcup}{\sqcup}\right]$$

in P', as 0, 1, 2, and $\sqcup$ are the members of Γ. That, together with part 5, allows us to extend the match to



Thus the dominos of parts 2, 3, and 4 let us extend the match by adding the second configuration after the first one. We want this process to continue, adding the third configuration, then the fourth, and so on. For it to happen we need to add one more domino for copying the # symbol.

Part 5.

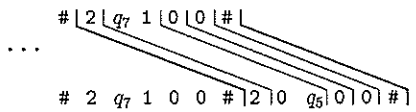
$$\text{Put } \left[\frac{\#}{\#}\right] \text{ and } \left[\frac{\#}{\sqcup\#}\right] \text{ into } P'.$$

The first of these dominos allows us to copy the # symbol that marks the separation of the configurations. In addition to that, the second domino allows us to add a blank symbol $\sqcup$ at the end of the configuration to simulate the infinitely many blanks to the right that are suppressed when we write the configuration.

Continuing with the example, let's say that in state q_7, upon reading a 1, M goes to state q_5, writes a 0, and moves the head to the right. That is, $\delta(q_7, 1) = (q_5, 0, \mathrm{R})$. Then we have the domino

$$\left[\frac{q_7 1}{0 q_5}\right] \text{ in } P'.$$

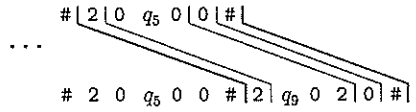
So the latest partial match extends to



Then, suppose that in state q_5, upon reading a 0, M goes to state q_9, writes a 2, and moves its head to the left. So $\delta(q_5, 0) = (q_9, 2, \mathrm{L})$. Then we have the dominos

$$\left[\frac{0 q_5 0}{q_9 02}\right], \left[\frac{1 q_5 0}{q_9 12}\right], \left[\frac{2 q_5 0}{q_9 22}\right], \text{ and } \left[\frac{\sqcup q_5 0}{q_9 \sqcup 2}\right].$$

The first one is relevant because the symbol to the left of the head is a 0. The preceding partial match extends to

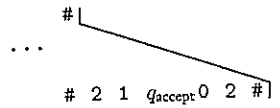


Note that, as we construct a match, we are forced to simulate M on input w. This process continues until M reaches a halting state. If an accept state occurs, we want to let the top of the partial match "catch up" with the bottom so that the match is complete. We can arrange for that to happen by adding additional dominos.

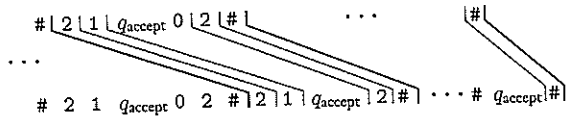
Part 6. For every $a \in \Gamma$,

$$\text{put } \left[\frac{a\, q_{\text{accept}}}{q_{\text{accept}}}\right] \text{ and } \left[\frac{q_{\text{accept}}\, a}{q_{\text{accept}}}\right] \text{ into } P'.$$

This step has the effect of adding "pseudo-steps" of the Turing machine after it has halted, where the head "eats" adjacent symbols until none are left. Continuing with the example, if the partial match up to the point when the machine halts in an accept state is



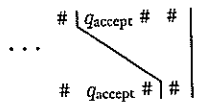
The dominos we have just added allow the match to continue:



Part 7. Finally we add the domino

$$\left[\frac{q_{\text{accept}}\#\#}{\#}\right]$$

and complete the match:



That concludes the construction of P'. Recall that P' is an instance of the MPCP whereby the match simulates the computation of M on w. To finish the proof, we recall that the MPCP differs from the PCP in that the match is required to start with the first domino in the list. If we view P' as an instance of the PCP instead of the MPCP, it obviously has a match, regardless of whether M halts on w. Can you find it? (Hint: It is very short.)

We now show how to convert P' to P, an instance of the PCP that still simulates M on w. We do so with a somewhat technical trick. The idea is to build the requirement of starting with the first domino directly into the problem so that stating the explicit requirement becomes unnecessary. We need to introduce some notation for this purpose.

Let $u = u_1 u_2 \cdots u_n$ be any string of length n. Define $\star u$, $u\star$, and $\star u\star$ to be the three strings

$$\begin{aligned} \star u &= *u_1 * u_2 * u_3 * \cdots * u_n \\ u\star &= u_1 * u_2 * u_3 * \cdots * u_n * \\ \star u\star &= *u_1 * u_2 * u_3 * \cdots * u_n *. \end{aligned}$$

Here, $\star u$ adds the symbol $*$ before every character in u, $u\star$ adds one after each character in u, and $\star u\star$ adds one both before and after each character in u.

To convert P' to P, an instance of the PCP, we do the following. If P' were the collection

$$\left\{\left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \left[\frac{t_3}{b_3}\right], \cdots, \left[\frac{t_k}{b_k}\right]\right\},$$

we let P be the collection

$$\left\{\left[\frac{\star t_1}{\star b_1\star}\right], \left[\frac{\star t_1}{b_1\star}\right], \left[\frac{\star t_2}{b_2\star}\right], \left[\frac{\star t_3}{b_3\star}\right], \cdots, \left[\frac{\star t_k}{b_k\star}\right], \left[\frac{*\diamond}{\diamond}\right]\right\}.$$

Considering P as an instance of the PCP, we see that the only domino that could possibly start a match is the first one,

$$\left[\frac{\star t_1}{\star b_1\star}\right],$$

because it is the only one where both the top and the bottom start with the same symbol—namely, $*$. Besides forcing the match to start with the first domino, the presence of the $*$s doesn't affect possible matches because they simply interleave with the original symbols. The original symbols now occur in the even positions of the match. The domino

$$\left[\frac{*\diamond}{\diamond}\right]$$

is there to allow the top to add the extra $*$ at the end of the match.