Chapter 4

The Post Correspondence Problem; Applications to Undecidability Results

4.1 The Post Correspondence Problem

The Post correspondence problem (due to Emil Post) is another undecidable problem that turns out to be a very helpful tool for proving problems in logic or in formal language theory to be undecidable.

Definition 4.1. Let Σ be an alphabet with at least two letters. An instance of the *Post Correspondence problem* (for short, PCP) is given by two nonempty sequences $U = (u_1, \ldots, u_m)$ and $V = (v_1, \ldots, v_m)$ of strings $u_i, v_i \in \Sigma^*$.

Equivalently, an instance of the PCP is a sequence of pairs $(u_1, v_1), \ldots, (u_m, v_m)$.

The problem is to find whether there is a (finite) sequence $(i_1, \ldots, i_p)$, with $i_j \in \{1, \ldots, m\}$ for $j = 1, \ldots, p$, so that

$$u_{i_1} u_{i_2} \cdots u_{i_p} = v_{i_1} v_{i_2} \cdots v_{i_p}.$$

Example 4.1. Consider the following problem:

$$(abab, ababaaa), (aaabbb, bb), (aab, baab),$$
$$(ba, baa), (ab, ba), (aa, a).$$

There is a solution for the string 1234556:

$$abab\, aaabbb\, aab\, ba\, ab\, ab\, aa = ababaaa\, bb\, baab\, baa\, ba\, ba\, a.$$

If you are not convinced that this is a hard problem, try solving the following instance of the PCP:

$$\{(aab, a), (ab, abb), (ab, bab), (ba, aab).\}$$

The shortest solution is a sequence of length 66.

We are beginning to suspect that this is a hard problem. Indeed, it is undecidable!

Theorem 4.1. *(Emil Post, 1946) The Post correspondence problem is undecidable, provided that the alphabet Σ has at least two symbols.*

There are several ways of proving Theorem 4.1, but the strategy is more or less the same: reduce the halting problem to the PCP, by encoding sequences of ID's as partial solutions of the PCP.

In Machtey and Young [**?**] (Section 2.6), the undecidability of the PCP is shown by demonstrating how to simulate the computation of a Turing machine as a sequence of ID's.

IN the notes, we give a proof involving special kinds of RAM programs (called Post machines in Manna [**?**]), which is an adaptation of a proof due to Dana Scott presented in Manna [**?**] (Section 1.5.4, Theorem 1.8).

4.2 Some Undecidability Results for CFG's

Theorem 4.2. *It is undecidable whether a context-free grammar is ambiguous.*

Proof. We reduce the PCP to the ambiguity problem for CFG's. Given any instance $U = (u_1, \ldots, u_m)$ and $V = (v_1, \ldots, v_m)$ of the PCP, let $c_1, \ldots, c_m$ be m new symbols, and consider the following languages:

$$\begin{aligned} L_U = \{u_{i_1} \cdots u_{i_p} c_{i_p} \cdots c_{i_1} \mid 1 \leq i_j \leq m, \\ 1 \leq j \leq p,\, p \geq 1\}, \\ L_V = \{v_{i_1} \cdots v_{i_p} c_{i_p} \cdots c_{i_1} \mid 1 \leq i_j \leq m, \\ 1 \leq j \leq p,\, p \geq 1\}, \end{aligned}$$

and $L_{U,V} = L_U \cup L_V$.

We can easily construct a CFG, $G_{U,V}$, generating $L_{U,V}$. The productions are:

$$
\begin{aligned}
S &\longrightarrow S_U \\
S &\longrightarrow S_V \\
S_U &\longrightarrow u_i S_U c_i \\
S_U &\longrightarrow u_i c_i \\
S_V &\longrightarrow v_i S_V c_i \\
S_V &\longrightarrow v_i c_i.
\end{aligned}
$$

It is easily seen that the PCP for (U, V) has a solution iff $L_U \cap L_V \neq \emptyset$ iff G is ambiguous. $\square$

Remark: As a corollary, we also obtain the following result: It is undecidable for arbitrary context-free grammars G_1 and G_2 whether $L(G_1) \cap L(G_2) = \emptyset$ (see also Theorem 4.4).

Recall that the computations of a Turing Machine, M, can be described in terms of instantaneous descriptions, $upav$.

We can encode computations

$$ID_0 \vdash ID_1 \vdash \cdots \vdash ID_n$$

halting in a proper ID, as the language, L_M, consisting all of strings

$$w_0\#w_1^R\#w_2\#w_3^R\#\cdots\#w_{2k}\#w_{2k+1}^R,$$

or

$$w_0\#w_1^R\#w_2\#w_3^R\#\cdots\#w_{2k-2}\#w_{2k-1}^R\#w_{2k},$$

where $k \geq 0$, w_0 is a starting ID, $w_i \vdash w_{i+1}$ for all i with $0 \leq i < 2k+1$ and w_{2k+1} is proper halting ID in the first case, $0 \leq i < 2k$ and w_{2k} is proper halting ID in the second case.

The language L_M turns out to be the intersection of two context-free languages L_M^0 and L_M^1 defined as follows:

(1) The strings in L_M^0 are of the form

$$w_0\#w_1^R\#w_2\#w_3^R\#\cdots\#w_{2k}\#w_{2k+1}^R$$

or

$$w_0\#w_1^R\#w_2\#w_3^R\#\cdots\#w_{2k-2}\#w_{2k-1}^R\#w_{2k},$$

where $w_{2i} \vdash w_{2i+1}$ for all $i \geq 0$, and w_{2k} is a proper halting ID in the second case.

(2) The strings in L_M^1 are of the form

$$w_0\#w_1^R\#w_2\#w_3^R\#\cdots\#w_{2k}\#w_{2k+1}^R$$

or

$$w_0\#w_1^R\#w_2\#w_3^R\#\cdots\#w_{2k-2}\#w_{2k-1}^R\#w_{2k},$$

where $w_{2i+1} \vdash w_{2i+2}$ for all $i \geq 0$, w_0 is a starting ID, and w_{2k+1} is a proper halting ID in the first case.

Theorem 4.3. *Given any Turing machine M, the languages L_M^0 and L_M^1 are context-free, and $L_M = L_M^0 \cap L_M^1$.*

Proof. We can construct PDA's accepting L_M^0 and L_M^1. It is easily checked that $L_M = L_M^0 \cap L_M^1$. $\square$

As a corollary, we obtain the following undecidability result:

Theorem 4.4. *It is undecidable for arbitrary context-free grammars G_1 and G_2 whether $L(G_1) \cap L(G_2) = \emptyset$.*

Proof. We can reduce the problem of deciding whether a partial recursive function is undefined everywhere to the above problem. By Rice's theorem, the first problem is undecidable.

However, this problem is equivalent to deciding whether a Turing machine never halts in a proper ID. By Theorem 4.3, the languages L_M^0 and L_M^1 are context-free. Thus, we can construct context-free grammars G_1 and G_2 so that $L_M^0 = L(G_1)$ and $L_M^1 = L(G_2)$. Then, M never halts in a proper ID iff $L_M = \emptyset$ iff (by Theorem 4.3), $L_M = L(G_1) \cap L(G_2) = \emptyset$. $\square$

Given a Turing machine M, the language L_M is defined over the alphabet $\Delta = \Gamma \cup Q \cup \{\#\}$. The following fact is also useful to prove undecidability:

Theorem 4.5. *Given any Turing machine M, the language $\Delta^* - L_M$ is context-free.*

Proof. One can easily check that the conditions for not belonging to L_M can be checked by a PDA. $\square$

As a corollary, we obtain:

Theorem 4.6. *Given any context-free grammar, $G = (V, \Sigma, P, S)$, it is undecidable whether $L(G) = \Sigma^*$.*

Proof. We can reduce the problem of deciding whether a Turing machine never halts in a proper ID to the above problem.

Indeed, given M, by Theorem 4.5, the language $\Delta^* - L_M$ is context-free. Thus, there is a CFG, G, so that $L(G) = \Delta^* - L_M$. However, M never halts in a proper ID iff $L_M = \emptyset$ iff $L(G) = \Delta^*$. $\square$

As a consequence, we also obtain the following:

Theorem 4.7. *Given any two context-free grammar, G_1 and G_2, and any regular language, R, the following facts hold:*

(1) $L(G_1) = L(G_2)$ is undecidable.

(2) $L(G_1) \subseteq L(G_2)$ is undecidable.

(3) $L(G_1) = R$ is undecidable.

(4) $R \subseteq L(G_2)$ is undecidable.

In contrast to (4), the property $L(G_1) \subseteq R$ is decidable!

4.3 More Undecidable Properties of Languages; Greibach's Theorem

We conclude with a nice theorem of S. Greibach, which is a sort of version of Rice's theorem for families of languages.

Let $\mathcal{L}$ be a countable family of languages. We assume that there is a coding function $c\colon \mathcal{L} \to \mathbb{N}$ and that this function can be extended to code the regular languages (all alphabets are subsets of some given countably infinite set).

We also assume that $\mathcal{L}$ is effectively closed under union, and concatenation with the regular languages.

This means that given any two languages L_1 and L_2 in $\mathcal{L}$, we have $L_1 \cup L_2 \in \mathcal{L}$, and $c(L_1 \cup L_2)$ is given by a recursive function of $c(L_1)$ and $c(L_2)$, and that for every regular language R, we have $L_1R \in \mathcal{L}$, $RL_1 \in \mathcal{L}$, and $c(RL_1)$ and $c(L_1R)$ are recursive functions of $c(R)$ and $c(L_1)$.

Given any language, $L \subseteq \Sigma^*$, and any string, $w \in \Sigma^*$, we define L/w by

$$L/w = \{u \in \Sigma^* \mid uw \in L\}.$$

Theorem 4.8. *(Greibach) Let $\mathcal{L}$ be a countable family of languages that is effectively closed under union, and concatenation with the regular languages, and assume that the problem $L = \Sigma^*$ is undecidable for $L \in \mathcal{L}$ and any given sufficiently large alphabet Σ. Let P be any nontrivial property of languages that is true for the regular languages, and so that if $P(L)$ holds for any $L \in \mathcal{L}$, then $P(L/a)$ also holds for any letter a. Then, P is undecidable for $\mathcal{L}$.*

Proof. Since P is nontrivial for $\mathcal{L}$, there is some $L_0 \in \mathcal{L}$ so that $P(L_0)$ is false.

Let Σ be large enough, so that $L_0 \subseteq \Sigma^*$, and the problem $L = \Sigma^*$ is undecidable for $L \in \mathcal{L}$.

We show that given any $L \in \mathcal{L}$, with $L \subseteq \Sigma^*$, we can construct a language $L_1 \in \mathcal{L}$, so that $L = \Sigma^*$ iff $P(L_1)$ holds. Thus, the problem $L = \Sigma^*$ for $L \in \mathcal{L}$ reduces to property P for $\mathcal{L}$, and since for Σ big enough, the first problem is undecidable, so is the second.

For any $L \in \mathcal{L}$, with $L \subseteq \Sigma^*$, let

$$L_1 = L_0 \# \Sigma^* \cup \Sigma^* \# L.$$

Since $\mathcal{L}$ is effectively closed under union and concatenation with the regular languages, we have $L_1 \in \mathcal{L}$.

If $L = \Sigma^*$, then $L_1 = \Sigma^* \# \Sigma^*$, a regular language, and thus, $P(L_1)$ holds, since P holds for the regular languages.

Conversely, we would like to prove that if $L \neq \Sigma^*$, then $P(L_1)$ is false.

Since $L \neq \Sigma^*$, there is some $w \notin L$. But then,

$$L_1/\#w = L_0.$$

Since P is preserved under quotient by a single letter, by a trivial induction, if $P(L_1)$ holds, then $P(L_0)$ also holds. However, $P(L_0)$ is false, so $P(L_1)$ must be false.

Thus, we proved that $L = \Sigma^*$ iff $P(L_1)$ holds, as claimed.

$\square$

Greibach's theorem can be used to show that it is undecidable whether a context-free grammar generates a regular language.

It can also be used to show that it is undecidable whether a context-free language is inherently ambiguous.