Lecture 21:
An Undecidable/ Uncomputable Problem: The Post Correspondence Problem (PCP)

Lecturer: Debanjan Bhowmik, Assistant Professor,
Department of Electrical Engineering,
Indian Institute of Technology Delhi
Scribed by: Het Patel and Shambhav Tewari

1 Introduction

In the previous lectures, we have looked at several undecidable languages: A_{TM}, $HALT_{TM}$, E_{TM}, etc. However, these languages are defined strictly within the realm of automata and Turning-machine theory. In this lecture, we will explore the phenomenon of undecidability outside the realm of problems concerning automata and Turing machines, i.e., we will look at a problem which can be considered as puzzle: the Post Correspondence Problem (PCP). We will ultimately show its connection with Turing machines and show that solving this puzzle is equivalent to deciding A_{TM}. Since we already know that A_{TM} is undecidable, by equivalence/ reduction, we show here that this puzzle is uncomputable for some cases.

2 Reducibility

As we have seen in the previous lecture, a reduction is a way of converting one problem to another problem in such a way that a solution to the second problem can be used to solve the first problem. Reducibility always involves two problems, which we call A and B. If A reduces to B, we can use a solution to B to solve A.

Reducibility plays an important role in classifying problems by decidability, and later in complexity theory as well. When A is reducible to B, solving A cannot be harder than solving B because a solution to B gives a solution to A. In terms of computability theory, if A is reducible to B and B is decidable, A also is decidable. Equivalently, if A is undecidable and reducible to B, B is undecidable. This last version is key to proving that various problems are undecidable, including the PCP problem we deal with here.

3 Post Correspondence Problem

3.1 Definition of the problem

Let A be an alphabet with at least two symbols. The input of the problem consists of two finite lists (of length N) $a_1, ..., a_N$ and $b_1, ..., b_N$ of words over A. The task is to pick a sequence of indices (repetition allowed) $(i_k)_{1 \leq k \leq K}$ with $K \geq 1$ and $1 \leq i_k \leq N \ \forall k$ such that

$$a_{i_1}...a_{i_K} = b_{i_1}...b_{i_K} \tag{1}$$

The decision problem then is to decide if such a solution exists or not, for the given two lists above. We will prove here that this problem is unsolvable by any algorithm.

Example 1: Let the alphabet $A = \{0, 1\}$ and the word lists (N = 3) be:

a_1	a_2	a_3
0	01	110

b_1	b_2	b_3
100	00	11

A solution to this problem would be the index sequence $(3, 2, 3, 1)$, because

$$a_3a_2a_3a_1 = 110 \cdot 01 \cdot 110 \cdot 0 = 110011100 = 11 \cdot 00 \cdot 11 \cdot 100 = b_3b_2b_3b_1 \tag{2}$$

Furthermore, since (3, 2, 3, 1) is a solution, so are all of its "repetitions", such as (3, 2, 3, 1, 3, 2, 3, 1), etc.; that is, when a solution exists, there are infinitely many solutions of this repetitive kind.

However, if the two lists had consisted of only a_2, a_3 and b_2, b_3 from those sets, then there would have been no such index sequence (the last digit of any such a string is not the same as the digit before it, whereas b only constructs pairs of the same digit).

An instance of the PCP is a set P of two finite lists of length N. The problem is to determine whether P has a solution index sequence as described above. In other words, an instance of the PCP is a set of N dominos, each with two strings on it - a numerator strings and a denominator string: $[\frac{a_1}{b_1}]$,...,$[\frac{a_N}{b_N}]$ The task is to arrange the dominos (repetition allowed) in such a way that the concatenation of the numerator strings (according to the solution index sequence) is the same as that of the denominator strings.

We can express the problem as a language as:

$$PCP = \{< P > | P \textit{ is an instance of} \\ \textit{the Post Correspondence Problem with a solution/match}\} \tag{3}$$

For a given instance, the instance having a solution (also called a match) is the same as saying that the string/ instance belongs to the language PCP above. Similarly, the instance not having a solution/ match means it doesn't belong to PCP. Thus, given an instance (a set of N dominoes each with a numerator string and a denominator string), the ability of finding out whether that instance has a match is same as the ability of deciding the language PCP. We show in this lecture PCP is undecidable, i.e., given an instance, we will not be able to tell 100 percent of the times if that instance has a match or not.

3.2 Computation Histories

Computation History (CH) is a formal way of representing the computation of a Turing machine M on a particular input w. It is simply the sequence of configurations that the machine goes through as it processes the input. An accepting computation history for M on w is a sequence of configurations, $C_1, C_2, ..., C_l$, where C_1 is the start configuration of M on w, and C_l is an accepting configuration of M, and each C_i legally follows from C_{i1} according to the rules of M. A rejecting computation history for M on w is defined similarly, except that C_l is a rejecting configuration.

If M does not halt on w, no accepting or rejecting CH exists for M on w. Also, M accepts w iff there is an accepting CH of M on w.

Computation History will prove to be a powerful tool when dealing with problems like PCP.

3.3 Undecidability of PCP

We have already seen that A_{TM} is an undecidable language. In order to prove that PCP is undecidable, it suffices to show that we can reduce A_{TM} to PCP. Thus, we will show that if we can decide PCP, then we can decide A_{TM}. Given any Turing machine M and input w, we construct an instance $P_{M,w}$ such that $P_{M,w}$ has a solution index sequence (or match) iff M accepts w. That is, $P_{M,w}$ has a match iff there is an accepting CH of M on w. So, if we can determine whether the instance has a match, we would be able to determine whether M accepts w. For this purpose, we choose the dominos in $P_{M,w}$ so that making a match forces a simulation of M to occur. To force simulation of M, we make 2 modifications to Turing Machine M and one change to our PCP problem:

1. M on input w never attempts to move the tape head beyond the left end of input tape.
2. If input w is the empty string ϵ, then we use the string $\_$ in place of w.
3. We modify the PCP problem to require that the first index in the solution index sequence is 1, i.e. the match starts with the first domino.

We call this problem the Modified Post Correspondence Problem (MPCP).

$$MPCP = \{< D > | D \textit{ is an instance of the PCP with a match that starts with first domino}\} \quad (4)$$

We will now prove the undecidability of PCP by first proving that MPCP is undecidable and then proving that MPCP can be reduced to PCP.

Proof: Let TM R decide the PCP and construct S deciding A_{TM}. S constructs an instance of the PCP $P_{M,w}$(=P) that has a match iff $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ accepts $w = w_1 w_2 ... w_n$. We will firstly show the construction of an instance P' of the MPCP by S. We do it through the following parts:

1. **First domino of the match**: Put $[\frac{\#}{\#q_0 w_1 w_2 ... w_n \#}]$ into P' as the first domino $[\frac{a_1}{b_1}]$ (this will be, by definition, the first domino of the match). The bottom string corresponds to the first configuration in the configuration history of M while M works on $w = w_1 w_2 ... w_n$

2. **A transition of the TM M for the tape head moving right:** For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$, if $\delta(q, a) = (r, b, R)$, put $[\frac{qa}{br}]$ into P'. This handles head motions to the right.

3. **A transition of the TM M for the tape head moving left:** For every $a, b, c \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$, if $\delta(q, a) = (r, b, L)$, put $[\frac{cqa}{rcb}]$ into P'. This handles head motions to the left.

4. **Taking care of the contents of the tape where the tape head isn't there:** For every $a \in \Gamma$, put $[\frac{a}{a}]$ into P'. This handles the tape cells not adjacent to the head.

5. Note that only the first domino's numerator has a # while there are two #'s from the first domino itself. So, we need to add one more domino for copying the symbol. In addition to that, we need to add a blank symbol at the end of the configuration to simulate the infinitely many blanks to the right. So, put $[\frac{\#}{\#}]$ and $[\frac{\#}{\_\#}]$ into P'.

6. For every $a \in \Gamma$, put $[\frac{a q_{accept}}{q_{accept}}]$ and $[\frac{q_{accept} a}{q_{accept}}]$ into P'. These dominos have the effect of adding "pseudo-steps" of the TM after it has halted, where the head "eats" adjacent symbols until none are left.

7. Finally, we add the domino $[\frac{q_{accept}\#\#}{\#}]$ into P' and complete the match.

Thus, we conclude the construction of P'. We will now convert to P' to P, an instance of PCP that still stimulates M on w.

For $u = u_1 u_2 ... u_n$, define $\$u = *u_1 * u_2 * u_3 * ... * u_n$, $u\$ = u_1 * u_2 * u_3 * ... * u_n *$, and $\$u\$ = *u_1 * u_2 * u_3 * ... * u_n *$. Now, for the conversion: If P' were the collection $\{[\frac{a_1}{b_1}], [\frac{a_2}{b_2}], [\frac{a_3}{b_3}], ..., [\frac{a_k}{b_k}]\}$, we define P to be the collection $\{[\frac{\$a_1}{\$b_1\$}], [\frac{\$a_2}{b_2\$}], [\frac{\$a_3}{b_3\$}], ..., [\frac{\$a_k}{b_k\$}], [\frac{\$\Omega}{\Omega}]\}$. Note that the only domino that could start a match is the first one because it is the only one whose denominator starts with a $*$ while all numerators start with a $*$. Adding $*$'s does not have any other effect on the match. We add the last domino to allow the numerator string to add the extra $*$ at the end of the match.

Example: Let $\Gamma = \{0, 1, 2, \_\}$, and w be the string 0100. Corresponding to the TM M, let a transition be: $\delta(q_0, 0) = (q_7, 2, R)$. So, part 1 places the domino $[\frac{\#}{\#q_0 0100\#}]$ in P', and the match begins:

$$numStr = \#$$

$$denomStr = \#q_0 0100\#$$

In addition, part 2 places the domino $[\frac{q_0 0}{2 q_7}]$ (corresponding to the above transition), and part 4 places the dominos $[\frac{0}{0}]$, $[\frac{1}{1}]$, $[\frac{2}{2}]$, and $[\frac{=}{\_}]$. Using parts 2,4, and 5, we extend our match to:

$$numStr = \# \cdot q_0 0 \cdot 1 \cdot 0 \cdot 0 \cdot \#$$

$$denomStr = \#q_0 0100\# \cdot 2q_7 \cdot 1 \cdot 0 \cdot 0 \cdot \#$$

If $\delta(q_7, 1) = (q_5, 0, R)$ then we have the domino $[\frac{q_7 1}{0 q_5}]$ by part2. So, the latest partial match extends to:

$$numStr = ...2 \cdot q_7 1 \cdot 0 \cdot 0 \cdot \#$$

$$denomStr = ...(2q_7 \cdot 1 \cdot 0 \cdot 0 \cdot \#) \cdot 2 \cdot 0 q_5 \cdot 0 \cdot 0 \cdot \#$$

Continuing this further till we reach the accept state by using dominos from only parts 2, 3, 4, and 5. Then, we will use dominos from part 6 and then finally we will add domino from part 7. Thus, making a match can force the simulation of M to occur. We can now convert this P' to an instance of PCP by using the simple technique described above.

4 References

1. Michael Sipser's "Introduction to the Theory of Computation" (Chapter 5.3)