Lecture 16: Post's Correspondence Problem and Rice's Theorem

Lecturer: Abrahim Ladha *Scribe(s): Samina Shiraj Mulani*

1 Introduction

Last time we proved $HALT, A_{TM}, E_{TM}, EQ_{TM}$ are undecidable. You may notice these are all problems which are just variations of language acceptance problems. You should be asking the following two questions:

- Are all language acceptance problems undecidable for Turing machines?
- Are the only useful unsolvable problems variations of language acceptance problems?

The answer to the first question is yes, and the second question is no. That is the goal of today's lecture. properties

2 Rice's Theorem

Definition 2.1 (Non-trivial Property)**.** A property is non-trivial if there exists at least one machine with and one without the property. Not every machine has or hasn't the property.

An example of a trivial property could be like $P = \{\langle M \rangle \mid M \text{ is a Turing machine}\}$, or $P = \{\langle M \rangle \mid |L(M)| \geq 0\}$. Every machine has these properties, so they are also trivially decidable.

Definition 2.2 (Semantic Property)**.** A property is semantic if it is about the languages and not the machines. Such a definition may be hard to formalize, given the diversity of possible behaviors a machine can or cannot do. We may attempt to formalize this as if $L(M_1) = L(M_2)$, then either both $\langle M_1 \rangle, \langle M_2 \rangle \in P$ or both $\langle M_1 \rangle, \langle M_2 \rangle \notin P$.

A syntactic property is about the encoding of the machine. For example, "M has 17 states". Easily decidable, count the states. A semantic property might be "M recognizes a language which has some (perhaps different) Turing machine to recognize the same language with 17 states". Syntactic properties are about the encodings. Semantic properties are about the languages. Intuitively, a semantic property requires somehow knowing something about the execution of the machine without simulating it. It requires you to turn it on. This can only be an informal definition.

Theorem 1 (Rice's Theorem)**.** Let P be a non-trivial semantic property of Turing machines. The language

$$P = \{\langle M \rangle \mid L(M) \text{ has "the property"}\}$$

is undecidable.

Rice's theorem states that all non-trivial semantic properties of Turing machines are undecidable. This is not really a theorem about Turing machines, rather it is about the recognizable languages. But we can really only study these languages through the lens of Turing machines.

Proof. Let P be a non-trivial semantic property of Turing machines. Since P is non-trivial, there exists machine M_1, M_0 such that $M_1 \in P$ and $M_0 \notin P$. Without loss of generality, suppose that $\emptyset$ is a language that hasn't the property (otherwise, repeat the proof showing $\overline{P}$ instead). Assume to the contrary that P is decidable. We give a decider for A_{TM}.

Algorithm 1 decider for A_{TM}

```
on input ⟨M, w⟩
build M′ hardcoded from M, w, M₁
if M′ ∈ P then
    accept
else
    reject
```

Algorithm 2 M' hardcoded from M, w, M_1

```
on input x
M = ...
w = ...
M₁ = ...
Simulate M on w
if M accepts w then
    Simulate M₁ on x
    if M₁ accepts x then
        accept
```

Note that:

$$\begin{aligned} M' \in P &\iff L(M') = L(M_1) &&\iff M \text{ accepts } w \iff \langle M, w\rangle \in A_{TM}. \\ M' \notin P &\iff L(M') = \emptyset &&\iff M \text{ rejects or loops on } w \iff \langle M, w\rangle \notin A_{TM} \end{aligned}$$

We see that $f(\langle M, w\rangle) = \langle M'\rangle$ is such a reduction so $A_{TM} \leq_m P$ and so P is undecidable. □

We may attempt to inject a philosophical meaning into the theorem. The semantic properties appear to be dependent upon behavior some point in the future. Many of these languages are undecidable, yet recognizable. The recognizers simply simulate it until the event occurs. We may determine the future only when it becomes the past.

You should also take great care in applying the theorem. Student's often misapply it, in an attempt to show a language is undecidable. It may be simpler to do a reduction in some cases. It is a powerful theorem, and should come with the required warning labels.

3 An Unsolvable Puzzle

The answer to the second question is no. There do exist undecidable problems that have nothing to do with Turing machines, and there do exist unprovable statements in logic which have nothing to do with provability or self reference.

We describe such a problem in logic, the Continuum Hypothesis. Formally it states:

$$CH : \neg\exists C[|\mathbb{N}| < |C| < |\mathcal{P}(\mathbb{N})|]$$

There is no set whos cardinality lies strictly between a countable and uncountable set. In 1940, Gödel showed that you could not disprove it. In 1963, Paul Cohen showed you could not prove it. Cohen had to invent the technique of forcing, for which he won a Fields medal, the only one ever awarded for logic. By the fact CH can niether be proved or disproved, it is a statement which is independent. It gives no positive answer one way or the other if it is true or false. You may take it or its negation as an axiom without any consequence to the consistency of the system. Note that the statement appears to have nothing to do with any meta problem, not self-reference, or provability or anything. This is unlike Gödel's sentence which states "I am not provable".

We show an analogous problem. The point of today's lecture is only to show you that there exists an unsolvable puzzle. The problem statement has nothing to do with Turing machines. The existence of algorithmically unsolvable problems is not as conditional as it feels on the Church-Turing Thesis. There do exist unsolvable problems with nothing to do with language theory. Here, we give a puzzle with no algorithmic solution. It is provably unsolvable.

4 Post's Correspondence Problem

Let a "domino" or "tile" be a pair of strings, consisting of an upper and lower portion. For example, a set of tiles could be

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

We say a set has a "match", if given unlimited copies of each tile, there exists a sequence (possibly with repetition) where the concatenations of the top equal the concatentations of the bottom. For example, given the previous set of tiles, consider the sequence 2,1,3,2,4.

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

- The top elements concatenated are $a \cdot b \cdot ca \cdot a \cdot abc = abcaaabc$
- The bottom elements concatenated $= ab \cdot ca \cdot a \cdot ab \cdot c = abcaaabc$

So this set has a match.

Post's correspondence problem is algorithmically unsolvable. There is no algorithm given a set of tiles to determine if there is a match or not. Restated as decidability of a language:

Theorem 2.

$$PCP = \{ \ \langle P \rangle \ | \ P \text{ is a set of tiles with a match } \} \text{ is undecidable.}$$

The proof idea is simple but has lots of small details. First, lets explore its universality in some way.

5 Proof Idea

5.1 Forcing a Start

First note we can set up a set of tiles such that we can force any decision making procedure to temper its behavior a certain way. For example, for the following set of tiles, the first (and last) choices are fixed. Any procedure is tempered into picking the first tile first.

$$\left\{ \left[\frac{\#b}{\#} \right], \left[\frac{a}{b} \right], \left[\frac{\$}{a\$} \right] \right\}$$

It is the only tile where the top and bottom begin with the same symbol. Similarly the last tile for any match of this set (if it exists) is also forced.

5.2 Forcing a Next Tile With Deficiency

For any decision making procedure, we can force it so that the *next* tile has to begin the way we want it. Note that a decision making procedure need not make selections of tiles sequentially. There is a lot of creative things algorithms can do. But if a certain tile set has a match, then the nth tile must have the desired property we will force. Consider the set

$$\left\{ \left[\frac{\#a}{\#} \right], \left[\frac{a}{a} \right] \right\}$$

Suppose it was forced to choose the first tile.[1] Now the "working strings" of the top and the bottom are $\#a$ and a respectively. Since the top is longer than the bottom, the next tile is forced to have its bottom begin with an a. That means we can only choose tiles of the form $\frac{\dots}{a\dots}$.

Also notice this deficiency is never satisfied. A decision making procedure will be forced to choose tiles ad infinitum. The working strings will always be $\#a^{k+1}$ and $\#a^k$. This idea, intuitively can encode a Turing machine which loops.

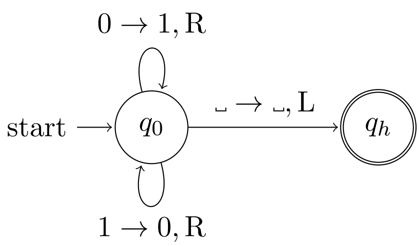
Using these ideas, we can encode the transition function of a Turing machine into a set of tiles. With the right setup, we can ensure that the tile instance only has a match if M accepts w. We will force the first tile, then force each of the next tiles to behave according to our Turing machine transition function. Then we will ensure there is a "cap piece" to match the deficiency if only if M accepts w.

[1]Forget for a moment that the second tile by itself is a match for this set. We will show a way around this later.

6 Proof of Unsolvability

6.1 Computation History

A computation history is a sequence of configurations in some string encoding made useful. Here, we will construct a set of tiles such that its only string match is this computation history. for example, the following is a computation history for the following machine.



$\#q_0 10\#0q_0 0\#01q_0 \sqcup\#0q_h 1\sqcup\#$

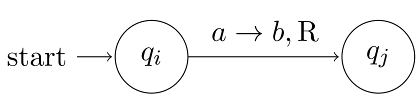
We may define an accepting computation history to be a computation history, where the last configuration is an accepting one. Notice that an accepting computation history is just a string which only exists if M accepts w. If M loops on w, such a computation history would be infinite in length, and then not a string. If M rejected w, such a computation history would end with a rejecting configuration instead of an accepting one. This is the heart of the method of accepting computation histories. We will use the fact that this string only exists if M accepts w, and we will create a set of tiles such that the only match is the accepting computation history. Then the set of tiles only has a match if there exists an accepting computation history, which only exists if M accepts w.

6.2 Construction

Proof. We construct a set of tiles $f(\langle M, w\rangle) = \langle P\rangle$ such that M accepts w if and only if P has a match. We begin our tile with this starting one.

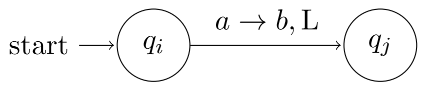
$$\left[\frac{\#}{\#q_0 w_1 w_2 ... w_n\#}\right]$$

Notice that the next tile is forced to begin with q_0 at the top. We will only add three[2] such tiles, where the tops of the tiles are $q_0 a, q_0 b, q_0 \sqcup$ so that only one gets picked to match to $q_0 w_1$.



Given a right transition in our machine, our configurations would change like $q_i a \to b q_i$. So we emulate this in our tiles. We add one tile per right move transition. If we have transition $\delta(q_i, a) = (q_j, b, R)$, we add tile $\left[\frac{q_i a}{b q_j}\right]$

[2]Technically $|\Gamma|$ for a well defined transition function



Of course, we must also simulate left moves, so if $\delta(q_i, a) = (q_j, b, L)$, our configurations would change looking like $cq_ia \to q_icb$. We add one domino per selection of c. Suppose $\Gamma = \{a, b, \textvisiblespace\}$. We need one for each possible left move of our machine. Note that we added one tile per right move, but three tiles per left move. This imbalance is just an artifact of the way we encode a snapshot of the state of the machine as a string.

$$\left[\frac{aq_ia}{q_jab}\right], \left[\frac{bq_ia}{q_jbb}\right], \left[\frac{\textvisiblespace q_ia}{q_j\textvisiblespace b}\right]$$

I hope you see the pattern here. We have created a set of tiles such that the decisions made to create a match are forced to simulate the Turing machine according to its transition function. The first tile creates a deficiency on the top. As the next sequence of tiles are forced fix this deficiency. As they do, they compute the next configuration and append it to the bottom! We need some more tiles to ensure the rest of the simulation is set up correctly.

$$\left[\frac{a}{a}\right], \left[\frac{b}{b}\right], \left[\frac{\textvisiblespace}{\textvisiblespace}\right]$$

We add one singleton tile (shown on the left) $\forall a \in \Gamma$ to make copies of the rest of the tape for us. Recall in a sequence of configurations, only a small local part of each sequential configuration changes. Most of the tape remains unchanged. These tiles are for performing this copying for us. Adding these tiles trivially adds a match but we show how to fix this later.

$$\left[\frac{\#}{\#}\right], \left[\frac{\#}{\textvisiblespace\#}\right]$$

We also need a cap between configurations and a way to use more space. Recall a configuration can have more blanks ($\textvisiblespace$) like leading zeroes because the tape is infinite. We only choose to write as many as necessary, one. If we want more, it will have to be done for the next configuration.

$$\left[\frac{aq_a}{q_a}\right], \left[\frac{bq_a}{q_a}\right], \left[\frac{\textvisiblespace q_a}{q_a}\right]$$

The accept state being q_a, we add the following tiles. This basically has the q_a "eat" the rest of the tape. This is so our cap fits nicely. Recall that on halting, the tape head may end where ever. This creates a slight amount of complexity for us, so we use this to simplify. We do not need to have the q_a eat right if we modify the machine to loop all the way to the right just before accepting.

$\left[\frac{q_a\#\#}{\#}\right]$ Once you reach the accept state in a Turing machine, you halt. However, our match will keep going to clean up the tape only so we can insert nice end cap. This completes the match. Note we have no end cap for rejection. This cap can only be placed if M accepts w.

You may have noticed we add tiles to not enforce the rule of a single start, like $\left[\frac{a}{a}\right]$ or $\left[\frac{\#}{\#}\right]$. We now modify all our tiles to enforce the start we want is the actual start. Given a set of dominoes we modify them in the following way. For $u = u_1 ... u_n$, let

$$\begin{aligned}\bullet u &= \bullet u_1 \bullet u_2 \bullet ... \bullet u_n \\ u\bullet &= u_1 \bullet u_2 \bullet ... \bullet u_n \bullet \\ \bullet u \bullet &= \bullet u_1 \bullet u_2 \bullet ... \bullet u_n \bullet\end{aligned}$$

Let $\left[\frac{t_s}{b_s}\right]$ be the start tile, $\left[\frac{t_e}{b_e}\right]$ be the end tile.

Given our set of tiles -

$$\left\{\left[\frac{t_s}{b_s}\right]\left[\frac{t_1}{b_1}\right]...\left[\frac{t_k}{b_k}\right]\left[\frac{t_e}{b_e}\right]\right\}$$

We modify them like -

$$\left\{\left[\frac{\bullet t_s}{\bullet b_s \bullet}\right]\left[\frac{\bullet t_1}{b_1 \bullet}\right]...\left[\frac{\bullet t_k}{b_k \bullet}\right]\left[\frac{\bullet t_e \bullet}{b_e \bullet}\right]\right\}$$

This can be generalized to make our reduction more like

$$A_{TM} \leq_m MPCP \leq_m PCP$$

but this correctly makes the start and end tiles for out match exactly the ones we want. It does come at the cost of using more symbols, and our match being twice as long. It is as if we skipped over every other cell of the tape. Our final set of tiles is then

$\left[\frac{\bullet\#}{\bullet\# \bullet q_0 \bullet w_1 \bullet w_2 \bullet ... \bullet w_n \bullet \#\bullet}\right]$ One start tile

$\left[\dfrac{\bullet q_i \bullet a}{b \bullet q_j \bullet}\right]$ For each right move transition like $\delta(q_i, a) = (q_j, b, L)$ we add one tile

$\left[\dfrac{\bullet a \bullet q_i \bullet a}{q_j \bullet a \bullet b \bullet}\right], \left[\dfrac{\bullet b \bullet q_i \bullet a}{q_j \bullet b \bullet b \bullet}\right], \left[\dfrac{\bullet \sqcup \bullet q_i \bullet a}{q_j \bullet \sqcup \bullet b \bullet}\right]$ For each left move transition like $\delta(q_i, a) = (q_j, b, L)$, we add $|\Gamma|$ tiles

$\left[\dfrac{\bullet a}{a \bullet}\right], \left[\dfrac{\bullet b}{b \bullet}\right], \left[\dfrac{\bullet \sqcup}{\sqcup \bullet}\right]$ $|\Gamma|$ tiles for copying

$\left[\dfrac{\bullet \#}{\# \bullet}\right], \left[\dfrac{\bullet \#}{\sqcup \bullet \# \bullet}\right]$ two extra tiles to help between configurations

$\left[\dfrac{\bullet a \bullet q_a}{q_a \bullet}\right], \left[\dfrac{\bullet b \bullet q_a}{q_a \bullet}\right], \left[\dfrac{\bullet \sqcup \bullet q_a}{q_a \bullet}\right]$ either $|\Gamma|$ or $2|\Gamma|$ tiles for eating

$\left[\dfrac{\bullet q_a \bullet \# \bullet \# \bullet}{\# \bullet}\right]$ One end tile

Lets stress why the computation is correct. We begin with:

$\left[\dfrac{\#}{\# C_0}\right]$ Then we are forced to add tiles in which the tops match C_0. By doing so, we have chosen the bottom to compute and place C_1

$\left[\dfrac{\# C_0 \#}{\# C_0 \# C_1}\right]$ Now, we must repeat, matching C_1 to force us to compute and place C_2.

$\left[\dfrac{\# C_0 \# C_1 \#}{\# C_0 \# C_1 \# C_2}\right]$ And so on.

The only way we can match is if we fix the deficiency, and the only way to do that is to place the end tile. We can only place the end tile if M accepts w. The match for our set of tiles exists if and only if there is an accepting computation of M on w. We had no reject end tile. If the machine loops, this computation history would be infinite and so there would be no match. We see that our construction $f(\langle M, w\rangle) = \langle P\rangle$ is correct. Namely

$$\langle M, w\rangle \in A_{TM} \iff \langle P\rangle \in PCP$$

So we conclude PCP is undecidable. □

For any kind of structure, we can note if there are enough degrees of freedom for us to simulate the transition function of a Turing machine, but perhaps not too many to make its problems too easy, any such structure will have unsolvable questions. This goes far beyond computational questions. There are unsolvable problems in combinatorics, geometry, topology, and more. Now that we have shown a simple combinatorial problem which is unsolvable, we can use this in further reductions.

7 Baba is You

Now we show Baba is You is undecidable. If we suppose that $BABA$ was solvable, that is, given any Baba is You level, there exists an algorithm to determine if it is winnable or not, we claim then you could solve PCP, a problem we just proved unsolvable. Our reduction would be added on like

$$A_{TM} \leq_m MPCP \leq_m PCP \leq_m BABA$$

The proof idea is given a set of tiles, to construct a Baba is You level which is winnable if and only if the tile set has a match. A reappearing theme is that the intuition is clear, even if the necessary gadgets are very complex.

- The paper: https://arxiv.org/abs/2205.00127
- The videos: https://www.youtube.com/playlist?list=PLE75TLHOnaOKrQsrhCUgOmuAX7l7dI66N
- The Baba is You level editor has online play where you can play other custom levels. https://hempuli.itch.io/baba-is-you-level-editor-beta