

CS 273

Lecture 26: Post's Correspondence Problem and Tilings

24 April 2008

This lecture covers Post's Correspondence Problem (section 5.2 in Sipser). Undecidability of this problem implies the undecidability of CFG ambiguity. We will also see how to simulate a TM with 2D tiling patterns and, as a consequence, show how undecidability implies the existence of aperiodic tilings.

1 Post's Correspondence Problem

Suppose that we have a set of *domino tiles*. Each domino piece has a string at the top and a string at the bottom., for example

abb
bc

. So a set S of dominos might look like:

$$S = \left\{ \begin{array}{|c|} \hline b \\ \hline ca \\ \hline \end{array}, \begin{array}{|c|} \hline a \\ \hline ab \\ \hline \end{array}, \begin{array}{|c|} \hline ca \\ \hline a \\ \hline \end{array}, \begin{array}{|c|} \hline abc \\ \hline c \\ \hline \end{array} \right\}.$$

A *match* for S is an ordered list of, one or more, dominos from S , such that if you read the symbols on the tops, this makes the same string as reading the symbols on the bottom. You can use the same domino more than once in a match, and you do not have to use all the elements of S . For example, here is a match for our example set

$$\begin{array}{|c|} \hline a \\ \hline ab \\ \hline \end{array} \begin{array}{|c|} \hline b \\ \hline ca \\ \hline \end{array} \begin{array}{|c|} \hline ca \\ \hline a \\ \hline \end{array} \begin{array}{|c|} \hline a \\ \hline ab \\ \hline \end{array} \begin{array}{|c|} \hline abc \\ \hline c \\ \hline \end{array}.$$

The tops and bottoms of the dominos both form the string $abcaaabc$.

Not all sets have a match. For example, T does not have a match because the tops are all longer than the bottoms.

$$T = \left\{ \begin{array}{|c|} \hline abc \\ \hline c \\ \hline \end{array}, \begin{array}{|c|} \hline ba \\ \hline a \\ \hline \end{array}, \begin{array}{|c|} \hline bb \\ \hline b \\ \hline \end{array} \right\}.$$

The set R does not have a match because there are no d's or f's in the top strings.

$$R = \left\{ \begin{array}{|c|} \hline ab \\ \hline df \\ \hline \end{array}, \begin{array}{|c|} \hline cb \\ \hline bd \\ \hline \end{array}, \begin{array}{|c|} \hline aa \\ \hline fa \\ \hline \end{array} \right\}.$$

It seems like it should be fairly easy to figure out whether a set of dominos has a match, but this problem is actually undecidable.

Post's Correspondence Problem (PCP) is the problem of deciding whether a set of dominos has a match or not.

The *modified Post's Correspondence Problem* (MPCP) is just like PCP except that we specify both the set of tiles and also a special tile. Matches for MPCP have to start with the special tile.

We will show that PCP undecidable in two steps. First, we will reduce A_{TM} to MPCP. Then we will reduce MPCP to PCP.

1.1 Reduction of A_{TM} to MPCP

1.1.1 An informal description of the set of tiles generated and why it works

Given a string $\langle M, w \rangle$, we will generate a set of tiles T such that T has a match, if and only if, M accepts w . In the set T there would a special initial tile that must be used first. The string the match would generate would be an accepting trace of M on w . That is, the top and bottom strings of the tiles of T forming the match will look like

$$\#C_1\#C_2\#\dots\#C_n\#\#,$$

where the C_i are configurations of the TM M . Here C_1 should be the start configuration, and C_n should be an accept configuration. (We are slightly oversimplifying what we are going to do, but do not worry about this quite yet.)

Here, C_i implies the configuration C_{i+1} . As such, C_i and C_{i+1} are almost identical, except for maybe a chunk of 3 or 4 letters. We will set the initial tile to be

$$\begin{array}{|c|} \hline \# \\ \hline \#q_0w\# \\ \hline \end{array},$$

where q_0w is the initial configuration of M when executed on w . At this point, the bottom string is way long than the shorter string. As such, to get a match, the tiling would have to copy the content of the bottom row to the top row. We will set up the tiles so that the copying would result in copying also the configuration C_0 to the bottom row, while performing the computation of M on C_0 . As such, in the end of this process, the tiling would look like

$$\frac{\#C_0\#}{\#C_0\#C_1\#}.$$

The trick is that again the bottom part is longer, and again to get a match, the only possibility is to copy C_1 to the top part, and in the process writing out C_2 on the bottom part, resulting in

$$\frac{\#C_0\#C_1\#}{\#C_0\#C_1\#C_2\#}.$$

Now, how are we going to do this copying/computation? The idea is to introduce for every character $x \in \Sigma_M \cup \{\#\}$, a copying tile

$$\begin{array}{|c|} \hline x \\ \hline x \\ \hline \end{array}.$$

Here Σ_M denotes the alphabet set used by M . Similarly, δ_M denotes the transition function of M .

Next, assume we have the transition $\delta_M(q, \mathbf{a}) = (q', \mathbf{b}, \mathbf{R})$, then we will introduce the computation tile

$$\begin{array}{|c|} \hline qa \\ \hline bq' \\ \hline \end{array}.$$

Similarly, for the transition $\delta_M(q_2, \mathbf{c}) = (\hat{q}, \mathbf{d}, \mathbf{L})$, we will introduce the following computation tiles

$$\forall y \in \Sigma_M \quad \begin{array}{|c|} \hline yq_2c \\ \hline \hat{q}yd \\ \hline \end{array}.$$

Here is what is going on in the i th stage: The bottom row as an additional configuration C_i written in it. To get a match, the tiling has to copy the configuration to the top row. But the copying tiles, only copy regular characters, and it can not copy states. Thus, when the copying process reaches the state character in C_i , it must use the right computation tile to copy this substring to the top row. Then it continues copying the rest of the configuration to the top. Naturally, as the copying goes on from the bottom row to the top row, new characters are added to the bottom row. The critical observation is that the computation tiles guarantee, that the added string to the bottom row is exactly C_{i+1} , since we copied the characters verbatim in the areas of C_i unrelated to the computation of M on C_i , and the computation tile copied exactly the right string for the small region where the computation changes.

Thus, if the starting configuration before the i th stage was

$$\frac{\#C_0\#C_1\#\dots\#C_{i-1}\#}{\#C_0\#C_1\#\dots\#C_i\#}$$

then after the i th stage, the string generated by the tiling (which is trying so hard to match the bottom and top rows) is

$$\frac{\#C_0\#C_1\#\dots\#C_{i-1}\# \ C_i\#}{\#C_0\#C_1\#\dots\#C_i\# \ C_{i+1}\#}.$$

Let this process continue till we reach the accepting configuration $C_n = \alpha q_{\text{acc}} x \beta$, where α, β are some string and x is some character in Σ_M . Here, the tiling we have so far looks like

$$\frac{\#C_0\#C_1\#\dots\#C_{n-1}\#}{\#C_0\#C_1\#C_2\#\dots\#C_n\#}.$$

The question is how do we make this into a match? The idea is that now, since C_n is an accepting configuration, we should now treat the rest of the tiling as a cleanup stage, and slowly reduce C_n to the empty string, as we copy it up and down. How do we do that? Well, let us introduce a delete tile

$$\begin{array}{|c|} \hline q_{\text{acc}}x \\ \hline q_{\text{acc}} \\ \hline \end{array}$$

into our set of tiles T (also known as a pacman tile). Clearly, if we use the copying tiles to copy $C_n = \alpha q_{\text{acc}} x \beta$ to the top row, and erase in the process the character x , using the above “delete x ” tile. We get

$$\frac{\#C_0\#C_1\#\dots\#C_{n-1}\#\alpha q_{\text{acc}}x\beta\#}{\#C_0\#C_1\#C_2\#\dots\#C_{n-1}\#\alpha q_{\text{acc}}x\beta\#\alpha q_{\text{acc}}\beta\#}.$$

We can now repeat this process, by introducing such delete tiles for every character of Σ , and also introducing backward delete tiles like

$$\left[\begin{array}{c} \mathbf{x}q_{\text{acc}} \\ q_{\text{acc}} \end{array} \right].$$

Thus, by using these delete tiles, we will get the tiling

$$\frac{\#C_0\#C_1\#\dots\#C_{n-1}\#\alpha q_{\text{acc}}x\beta\#\alpha q_{\text{acc}}\beta\#\dots\#q_{\text{acc}}y\#}{\#C_0\#C_1\#C_2\#\dots\#C_{n-1}\#\alpha q_{\text{acc}}x\beta\#\alpha q_{\text{acc}}\beta\#\dots\#q_{\text{acc}}y\#q_{\text{acc}}\#}.$$

To finish of the tiling, we introduce a stopper tile

$$\left[\begin{array}{c} q_{\text{acc}}\#\#\# \\ \#\# \end{array} \right].$$

Adding it to the tiling results in the required match. This is the tiling

$$\frac{\overbrace{\#C_0\#C_1\#\dots\#C_{n-1}\#\alpha q_{\text{acc}}x\beta\#}^{\text{the accepting trace}} \quad \overbrace{\alpha q_{\text{acc}}\beta\#\dots\#q_{\text{acc}}y\#q_{\text{acc}}\#\#\#}^{\text{winding down the match}}}{\overbrace{\#C_0\#C_1\#\dots\#C_{n-1}\#\alpha q_{\text{acc}}x\beta\#}^{\text{the accepting trace}} \quad \overbrace{\alpha q_{\text{acc}}\beta\#\dots\#q_{\text{acc}}y\#q_{\text{acc}}\#\#\#}^{\text{winding down the match}}}.$$

Its now easy to argue that with this set of tiles, if there is a match it must have the above structure which encodes an accepting trace.

1.1.2 Computing the tiles

Let us recap the above description. We are given a string $\langle M, w \rangle$, and we are generating a set of tiles T , as follows. The set containing the initial tile is

$$T_1 = \left\{ \left[\begin{array}{c} \# \\ \#q_0w\# \end{array} \right] \right\}. \quad /* \text{ initial tile } */$$

The set of copying tiles is

$$T_2 = \left\{ \left[\begin{array}{c} \mathbf{x} \\ \mathbf{x} \end{array} \right] \mid x \in \Sigma_M \cup \{\#\} \right\}.$$

The set of computation tiles for right movement of the head is

$$T_3 = \left\{ \left[\begin{array}{c} q\mathbf{x} \\ yq' \end{array} \right] \mid \forall q, x, q', y \text{ such that } \delta_M(q, x) = (q', y, \mathbf{R}) \right\}.$$

The set of computation tiles for left movement of the head is

$$T_4 = \left\{ \begin{array}{|c|} \hline zqx \\ \hline q'zy \\ \hline \end{array} \middle| \forall q, x, q', y, z \text{ such that } \delta_M(q, x) = (q', y, L) \right\}.$$

The set of pacman tiles (i.e., delete tiles) is

$$T_5 = \left\{ \begin{array}{|c|} \hline q_{\text{acc}}x \\ \hline q_{\text{acc}} \\ \hline \end{array}, \begin{array}{|c|} \hline xq_{\text{acc}} \\ \hline q_{\text{acc}} \\ \hline \end{array} \middle| \forall x \in \Sigma_M \right\}.$$

Finally, the set of containing the stopper tiles is

$$T_6 = \left\{ \begin{array}{|c|} \hline q_{\text{acc}}\#\#\# \\ \hline \#\# \\ \hline \end{array} \right\}.$$

Let $T = T_1 \cup T_2 \cup T_3 \cup T_4 \cup T_5 \cup T_6$. Clearly, given $\langle M, w \rangle$, we can easily compute the set T . Let *AlgTM2MPCP* denote the algorithm performing this conversion.

We summarize our work so far.

Lemma 1.1 *Given a string $\langle M, w \rangle$, the algorithm *AlgTM2MPCP* computes a set of tiles T that is an instance of MPCP. Furthermore, T contains a match if and only if M accepts w .*

This implies the following result.

Theorem 1.2 *The MPCP problem is undecidable.*

Proof: The reduction is from A_{TM} . Indeed, assume for the sake of contradiction that the MPCP problem is decidable, and we are given a decider *decider_MPCP* for it. Next, we use it to build the following decider for A_{TM} .

```

Decider7-ATM ( $\langle M, w \rangle$ )
   $T \leftarrow \text{AlgTM2MPCP}(\langle M, w \rangle)$ 
   $res \leftarrow \text{decider\_MPCP}(T)$ .
  return  $res$ 

```

Clearly, this is a decider, and it accepts if and only if M halts on w . But this is a contradiction, since A_{TM} is undecidable.

Thus, our assumption (that MPCP is decidable) is false, implying the claim. ■

1.2 Reduction to MPCP to PCP

We are almost done, but not quite. Since our reduction only show that MPCP is undecidable, but we want to show that PCP is decidable. An instance of MPCP is a set of tiles (just like an instance of PCP), with the additional constraint that a specific tile is denoted as an initial tile that **must** be used as the first tile in the matching. As such, to convert this instance T of MPCP into PCP we need to somehow remove the need to specify the initial tile.

To this end, let us introduce a new special \star character into the alphabet used by the tiles. Next, given a string $w = x_1x_2 \dots x_m$, let us denote

$$\star w = \star x_1 \star x_2 \dots \star x_m \quad \star w \star = \star x_1 \star x_2 \dots \star x_m \star \quad w \star = x_1 \star x_2 \dots \star x_m \star .$$

In particular, if

$$T = \left\{ \begin{array}{|c|} \hline t_1 \\ \hline b_1 \\ \hline \end{array}, \dots, \begin{array}{|c|} \hline t_m \\ \hline b_m \\ \hline \end{array} \right\},$$

where $\begin{array}{|c|} \hline t_1 \\ \hline b_1 \\ \hline \end{array}$ is the special initial tile, then the new set of tiles would be

$$X = \left\{ \begin{array}{|c|} \hline \star t_1 \\ \hline b_1 \star \\ \hline \end{array}, \begin{array}{|c|} \hline \star t_2 \\ \hline b_2 \star \\ \hline \end{array}, \dots, \begin{array}{|c|} \hline \star t_m \\ \hline b_m \star \\ \hline \end{array} \right\} \cup \left\{ \begin{array}{|c|} \hline \star t_1 \\ \hline \star b_1 \star \\ \hline \end{array} \right\}.$$

Note, that in this new set of tiles, the only tile that can serve as the first tile in the match is

$$\begin{array}{|c|} \hline \star t_1 \\ \hline \star b_1 \star \\ \hline \end{array},$$

since its the only tile that has in the bottom string a \star as the first character. Now, to take care of the balance of stars in the end of the string, we also add the tile

$$Y = X \cup \left\{ \begin{array}{|c|} \hline \star \star \\ \hline \star \\ \hline \end{array} \right\}$$

Its now easy to verify that if the original instance of T of MPCP had a match, then the set Y also has a match.

The important thing about Y is that it does not need to specify what is the special initial tile (this is a minor difference to T , but a difference nevertheless). As such, U is an instance of PCP. We conclude:

Lemma 1.3 *Given a string $\langle M, w \rangle$, the algorithm [AlgTM2PCP](#) computes a set of tiles T that is an instance of PCP. Furthermore, T contains a match if and only if M accepts w .*

As before, this implies the described result.

Theorem 1.4 *The PCP problem is undecidable.*

2 Reduction of PCP to $\text{AMBIG}_{\text{CFG}}$

We can use the PCP result to prove a useful fact about context-free grammars. Let us define

$$\text{AMBIG}_{\text{CFG}} = \left\{ \langle G \rangle \mid G \text{ is a CFG and } G \text{ is ambiguous} \right\}.$$

We remind the reader that a grammar G is ambiguous if there are two different ways for G to generate some word w .

We will show this problem is undecidable by a reduction from PCP. That is, given a PCP problem S , we will construct a context-free grammar which is ambiguous exactly when S has a match. This means that any decider for $\text{AMBIG}_{\text{CFG}}$ could be used to solve PCP, so such a decider can not exist.

Specifically, suppose that S looks like

$$S = \left\{ \left[\begin{array}{c} t_1 \\ b_1 \end{array} \right], \dots, \left[\begin{array}{c} t_k \\ b_k \end{array} \right] \right\}.$$

We could define the corresponding grammar as

$$D \rightarrow T \mid B$$

$$T \rightarrow t_1 T \mid t_2 T \mid \dots t_k T \mid t_1 \mid \dots \mid t_k$$

$$B \rightarrow b_1 B \mid b_2 B \mid \dots b_k B \mid b_1 \mid \dots \mid b_k,$$

with D as the initial symbol. This grammar is ambiguous if the tops of a sequence of tiles form the same string as the bottoms of a sequence of tiles. However, there is nothing forcing the two sequences to use the same tiles in the same order.

So, we will add some labels to our rules which name the set of tiles we have used. Let us suppose the tiles are named d_1 through d_k . Then we will make our grammar generate strings like $t_i t_j \dots t_m d_m \dots d_j d_i$ where the second part of the string contains the labels of the tiles used to build the first part of the string (in reverse order).

So our final grammar H looks like

$$D \rightarrow T \mid B$$

$$T \rightarrow t_1 T d_1 \mid t_2 T d_2 \mid \dots t_k T d_k \mid t_1 d_1 \mid \dots \mid t_k d_k$$

$$B \rightarrow b_1 B d_1 \mid b_2 B d_2 \mid \dots b_k B d_k \mid b_1 d_1 \mid \dots \mid b_k d_k.$$

Here VD is the initial symbols. Clearly, there is an ambiguous word $\text{win}L(H)$ if and only if the given instance of PCP has a match. Namely, deciding if a grammar is ambiguous is equivalent to deciding an instance of PCP. But since PCP is undecidable, we get that deciding if a CFG grammar is ambiguous is undecidable.

Theorem 2.1 *The language $\text{AMBIG}_{\text{CFG}}$ is undecidable.*

3 2D tilings

Show some of the pretty tiling pictures linked on the 273 lectures web page, walking through the following basic ideas.

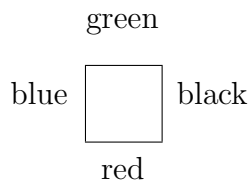
A tiling of the plane is *periodic* if it is generated by repeating the contents of some rectangular patch of the plane. Otherwise the tiling is *aperiodic*.

A set of tiles is *aperiodic* if these tiles can tile the whole plane but all tilings generated by these set are aperiodic.

Wang's conjectured, in 1961, that if a set of tiles can cover the plane at all, it can cover the plane periodically.

This is a tempting conjecture, but it is wrong.

In 1966, Robert Berger found a set of 20426 Wang tiles that is aperiodic. A Wang tile is a square tile with colors on its edges. The colors need to match when you put the tiles together.



Various people found smaller and smaller aperiodic sets of Wang tiles. The minimum so far is due to Karel Culik II, and it is made out of 13 tiles.

Other researchers have built aperiodic sets of tiles with pretty shapes, e.g. the Penrose tiles. (Show pretty pictures.)

3.1 Tilings and undecidability

Let us restrict our attention to square tiles that are unit squares, as they are easier to understand. Next, consider the problem of deciding whether a set T of such unit square tiles can cover the plane. If there were no aperiodic tile sets, then we could decide this problem as follows

Can T tile the plane?

Enumerate all pairs of integers (m, n) . For each pair, do

- Find (by exhaustive search) all ways to cover an m by n box using the tiles in T .
- If there are no ways to tile this box, the plane can not be tiled either. So halt and reject.
- Check whether any of these tilings has matching top and bottom, left and right sides. If so, we can repeat this pattern to tile the whole plane. So halt and accept

However, this procedure will loop forever if the set T has only aperiodic tilings.

In general, we have the following correspondence between 2D tilings and Turing machine behaviors:

tile set	Turing machine
can not cover the plane	halts
has a periodic tiling	loops repeating configurations
has an aperiodic tilings	runs forever without repeating configurations

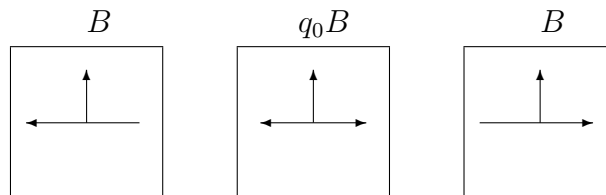
4 Simulating a TM with a tiling

In fact, we can reduce A_{TM} to a version of the tiling problem called the *tiling completion problem*. This problem asks whether, given a set of tiles and a partial tiling of the plane, can we complete this into a tiling of the whole plane.

To decide A_{TM} using a tiling decider, we first hardcode our input string into M , making a TM M_w . We then construct a tiling which covers the whole plane exactly when M_w does not halt on a blank input tape.

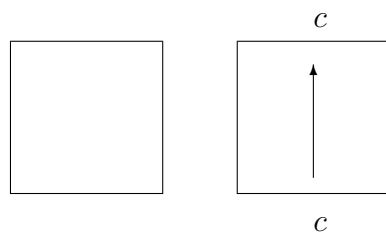
We have three types of tiles: start tiles, copy tiles, and action tiles. These tiles have labels on their edges and also arrows, both of which have to match when the tiles are put together.

Start tiles: Here is the partial tiling we use as input to the problem. B is the blank symbol for M_w and q_0 is its start state.

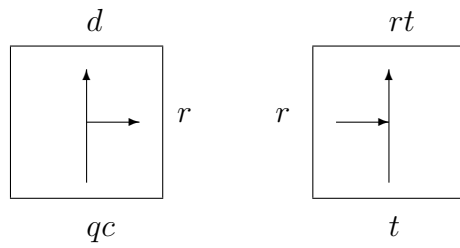


The tile set is engineered so that the rest of this row can only be filled by repeating the left and right tiles from this set.

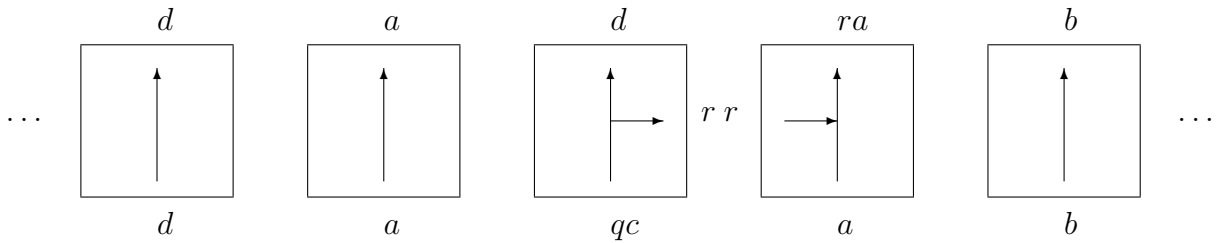
Copy tiles: For every character c in M_w 's alphabet, we have a tile that just copies c from one row to the next. We also have an entirely blank tile which must (given the design of this tile set) cover the lower half-plane.



Action tiles: A transition of M_w $\delta(q, c) = (r, d, R)$ is implemented using a “split tile” (left below) and a set of “merge tiles” (right below) for every character t in the tape alphabet.



So a single transition line might look like:



So, this reduction shows that the tiling completion problem is undecidable.

References

Branko Grünbaum and G. C. Shephard (1986) **Tilings and Patterns**, W H Freeman.

Raphael M. Robinson (1971) Undecidability and nonperiodicity for tilings of the plane *Inventiones Mathematicae* 12/3, pp. 177-209.