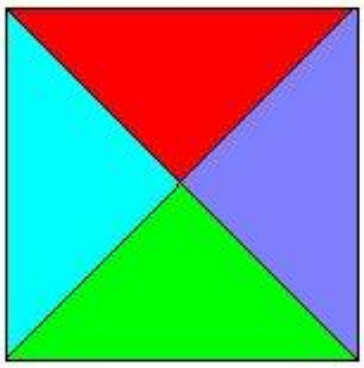
Tiling Problems

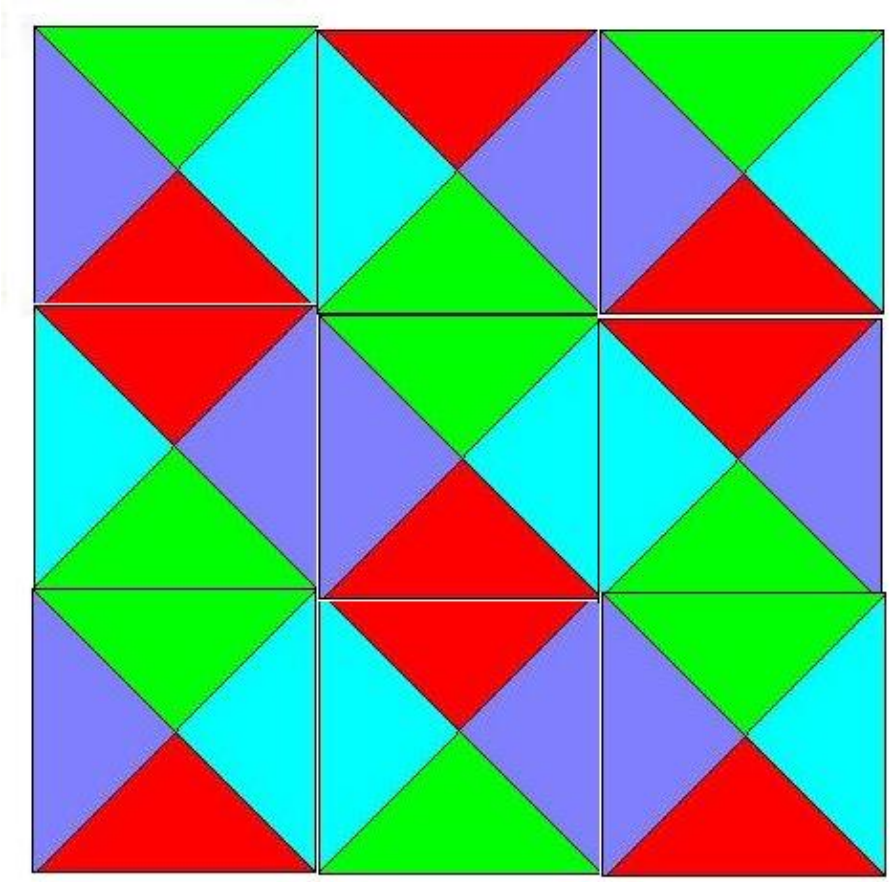
This document supersedes the earlier notes posted about the tiling problem.

1 An Undecidable Problem about Tilings of the Plane

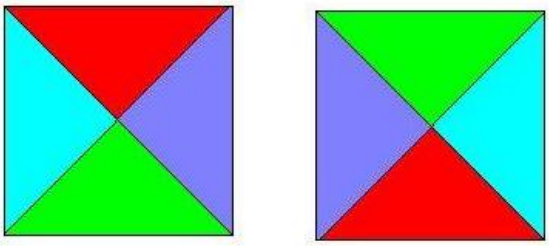
The undecidable problems we saw at the start of our unit on undecidability were all questions about the behavior of computer programs. But in fact there are many decision problems which, on the surface, have nothing to do with computer programs yet can still be proved to have no algorithmic solution. One of these is a tiling problem. Imagine square tiles divided into colored triangular quadrants.



The problem is to determine whether we can cover a quandrant of the infinite plane with these tiles in such a manner that the colors on adjacent tiles match up. Even if we have just one type of tile, but are allowed to rotate it it 180 degrees, then we can cover the entire plane.



Let us make a rule that we are not allowed to turn tiles in this fashion. In that case, our single tile does not suffice to cover the plane, but the set of two tiles



does. If we wanted to, we could enforce this restriction on turning by designing the tiles with little slits and notches so that the would only fit together in one orientation.

Here is our general statement of the tiling problem: An instance of the problem is a finite set T of tile types, together with a specification of a distinguished corner tile $c \in T$. Each tile is a sequence $t = (n, e, s, w)$ of four symbols, which serve to identify the colors at the top, right, bottom and left edges of the tile. A tiling is a function $f : \mathbf{N} \times \mathbf{N} \to T$ that tells which tile is associated to each square on the infinite quarter-plane. The requirement of consistency of colors can be written as a pair of conditions

$$f(i,j)_1 = f(i,j+1)_3, f(i,j)_= f(i,j+1)_4$$

for all $i, j \geq 0$. (In other words, the first (= top) color of the tile in position (i, j) is the same as the third(= bottom) color of the tile in position $(i, j + 1)$, etc.) The requirement that the corner tile be placed in the corner is expressed formally as

$$f(0,0) = c$$

A problem instance $< T, c >$ can thus be encoded as a finite string of symbols, representing the set of tiles along with the special corner tile.

We will prove:

Theorem 1 *The language*

$$\{< T, c >: \texttt{there is a tiling } f : \mathbf{N} \times \mathbf{N} \to T \texttt{ with } f(0,0) = c\}$$

is undecidable.

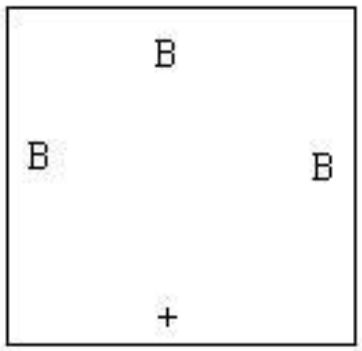
We have formulated this decision problem as a language. Informally what this says is that there is no computer program that takes as input a specification of the set T of tile types and the corner tile, and answers correctly whether the infinite quarter-plane can be tiled. As usual, the proof of undecidability will be done by a reduction from a problem we already know to be undecidable. Strange as it seems, we can reduce a version of the halting problem for Turing machines to this tiling question, and the proof is pretty easy.

Let M be a Turing machine with initial sate q_0. We'll denote the blank symbol by B. We'll describe a finite set T of tiles associated with M. The colors on these tiles will be drawn from the set

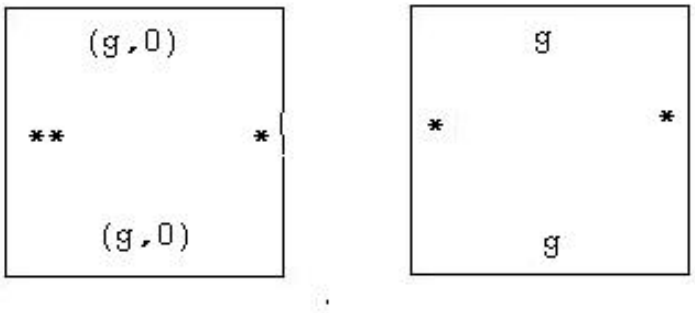
$$\{+, *, **\} \cup \Gamma \cup \Gamma \times 0 \cup Q \times \Gamma \cup Q \times \Gamma \times 0 \cup Q \times L, R,$$

where Q is the state set and G is the tape alphabet.

Our set includes the tile:

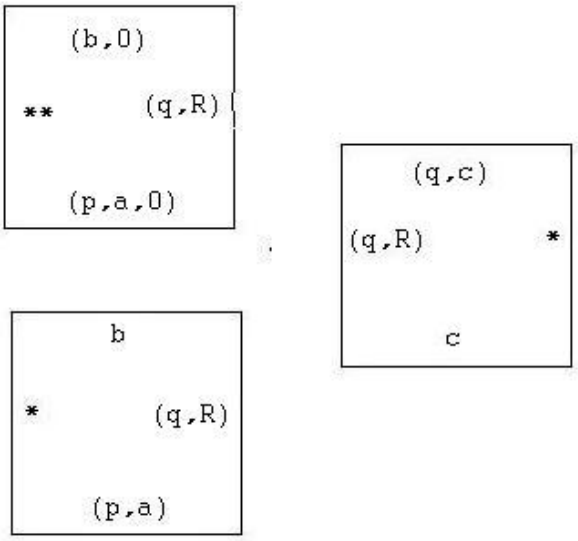


For every tape symbol $g \in \Gamma$, our set includes the two tiles

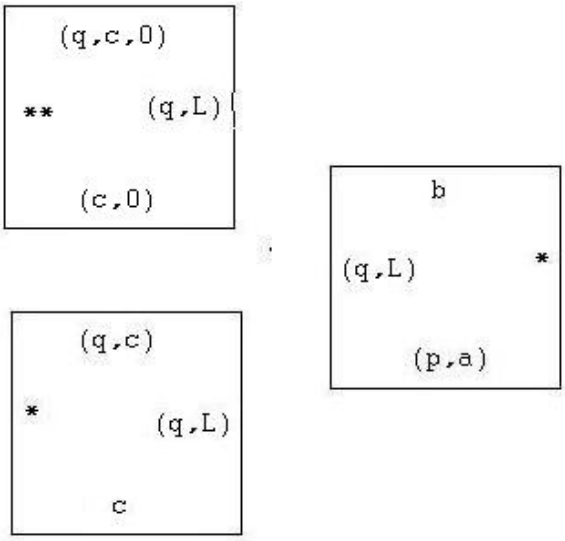


The left-hand tile should be considered a variant of the one on the right; tiles with a 0 component in the top and bottom colors and two stars as the left color are meant to tile the left-hand wall of the quadrant.

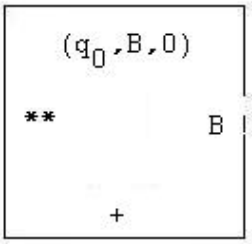
For each right-moving transition $(p, a) \to (q, b, R)$ (in state p, reading symbol a, change a to b, change to state q and move right), and for each tape symbol c, our set includes the three tiles



For each left-moving transition $(p, a) \rightarrow (q, b, L)$ and each tape symbol c, our set includes the three tiles

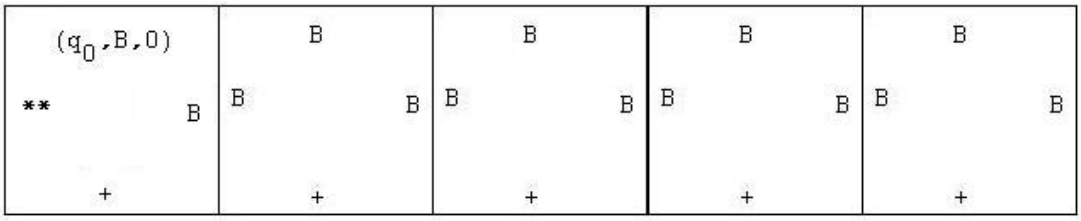


Finally, our corner tile is

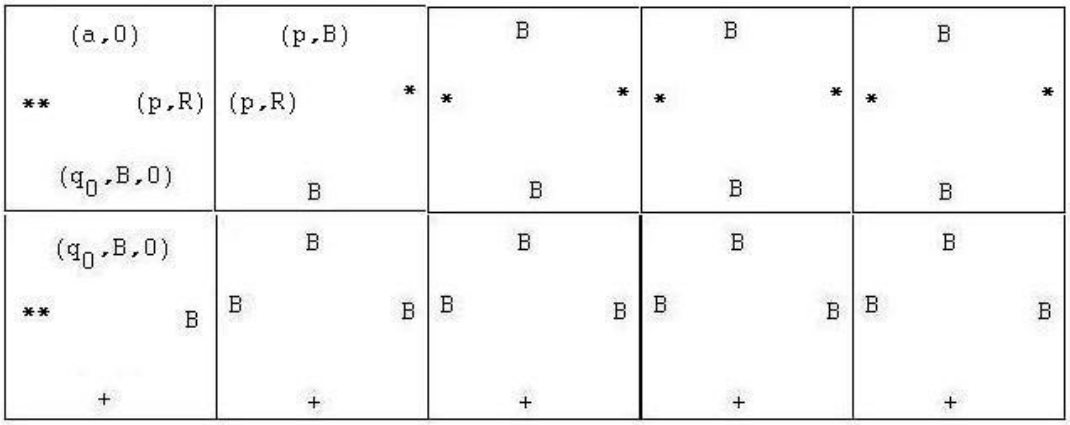


The first important thing to observe is that given the specification of a Turing machine, we can produce the finite set of tiles associated with it: That is, there is an algorithm that takes input $< M >$ and produces the assoicated $< T, c >$. We have just described this algorithm.

Now let us see what it takes to tile a quadrant of the plane with this set of tiles. The placement of the corner tile is forced, and once that is done we have no choice about how to tile the first row:



To continue the tiling, there is at most only one possible tile we can put at the start of the second row, and this necessitates a transition of the form $(q_0, B) \rightarrow (p, a, R)$. This in turn forces the tiling for the remainder of the second row: Notice that the labels of the top



edges of the first row spell out the initial configuration of the machine when started on a blank tape (you need to ignore the 0 in the leftmost column). The top edges of the second

row spell out the configuration that follows from this one. The only way to continue the tiling into a third row, is to match the color (p, B) in the second column, and the only way to do this is to have a transition $(p, B) \to (p', c, R)$ or $(p, B) \to (p', c, L)$. This will force the tiling of the entire third row, and the labels at the tops of these tiles will again spell out the subsequent configuration.

Thus we can tile the complete infinite quadrant if and only if M runs forever when started on a blank tape. So if we had an algorithm for deciding the tiling problem, we would have one for deciding if a Turing machine halts or runs forever when started on a blank tape.

To see that this latter problem (a simple variant of the halting problem) is undecidable, we will reduce our original version of the halting problem to it. Suppose we had an algorithm to decide whether a given machine halts when started on a blank tape. We could then determine for arbitrary M and w whether M ever halts when started on w. Given M, w, construct a Turing machine M' which when started on a blank tape, writes the string w and then simulates M. Then M' halts on a blank tape if and only if M halts on w.

The theorem we have just proved illustrates the utility of the Turing machine model of computation. Because it is so simple, and because the syntactic restrictions are so few, it is relatively easy to simulate the behavior of Turing machines in a variety of combinatorial problems.

2 An NP-Complete Problem about Tilings of the Plane

An instance of the *bounded tiling problem* is a finite set T of tile types, as above, and a sequence $(t_1, \ldots, t_s)$ where each $t_i \in T$, and where the left edge of t_{i+1} has the same color as the right edge of t_i (so that this sequence is the start of a consistent tiling of the first row). If there are k different tiles then there are no more than $4k$ different colors, so each color can be encoded in $O(\log k)$ symbols. Thus the set T can be encoded by $O(k \log k)$ symbols, and the sequence by $O(s \log k)$ symbols. So the length of the input is $N = O((s + k) \log k)$.

An input is accepted if there is a tiling of the $s \times s$ square $\{0, ..., s-1\} \times \{0, ..., s-1\}$ with the given tiles such that the first row of the tiling is the one given in the problem instance.

Note that this problem is decidable by the brute force algorithm of trying out every possible tiling of the remaining $s^2 - s$ cells with tiles from T, and checking whether the tiling is consistent.

Let us first see that this problem is in NP: A prospective tiling is given by guessing a sequence of s^2 tiles. The sequence is encoded by a string of length $O((s \log k)^2) = O(N^2)$, so the time required to produce the guess is bounded above by a polynomial in the size of the input. To verify that the guess is a correct tiling, we have to examine each edge of

the $s2$ tiles, so roughly $2s^2$ pairs of edges must be inspected, which is still polynomial in the original input size. Note that if we had not given the first row and just encoded s in binary, then the size of the guessed tiling could have been exponential in the size of the input. This is the reason for insisting that the whole first row be given–to ensure that the size of a tiling be polynomial in the size of the input.

Now let us suppose this tiling problem is actually in P. We want to show that every problem in NP must then be in P. Let L be in NP. Then L is recognized by a nondeterministic Turing machine M such that for some $c, r > 0$, every computation of M on an input of length n terminates (in acceptance or rejection) after no more than cn^r steps.

Let us tweak M a little bit to make a new machine N: N behaves exactly like M, except whereveve M has a transition into the accept state:

$$(q, a) \rightarrow q_{accept},$$

N instead runs forever. That is, we add a new state called $q_{forever}$ and replace each transition of the above form by the transitions

$$(q, a) \rightarrow (q_{forever}, a, R)$$

$$(q_{forever}, b) \rightarrow (q_{forever}, a, R),$$

for all tape symbols b.

Note the critical property of N: For every input string w, every computation of M on w ends after at most $c|w|^r$ steps, and M accepts w if and only if some computation of M on w ends in the accept state. Thus M *accepts w if and only if some computation of N on w lasts for at least $c|w|^r + 1$ steps.*

So, given M and w, we construct N as just described. We use the transition function of N to construct a set of tile types T as described in the preceding section (note that nothing in that construction requires the Turing machine to be deterministic). Let $w = a_1 \cdots a_n$. We add a different corner tile with $**$ at left, $+$ at bottom, $(q_0, a_1, 0)$ at top, and B at right, and $n - 1$ additional tiles with $+$ at bottom, B at left and right, and $a_2, \ldots, a_n$ at top. Set $s = c|w|^r + 1$ and specify the first row of the tiling: The first n tiles in the row are the corner tile just specified, followed by the $n - 1$ tiles with labels $a_2, \ldots, a_n$ at the top; the remaining $s - n$ tiles in the first row are the ones with B at top, right, and bottom.

Note that it takes time polynomial in the length of w to extract this description of the set of tile types and the first row. Once again, tiling of the subsequent rows is possible if and only if there is a computation of N whose successive configurations match the top colors in each row. In particular, tiling the first $s = c|w|^r + 1$ rows is possible if and only if M accepts w.

Thus if we had an algorithm for deciding the bounded tiling problem in polynomial time, then for every language L in NP, we would have a polynomial-time algorithm to decide whether w is in L, and thus P would be equal to NP. We say that the bounded

tiling problem is *NP-complete*. This means that, first, it is in NP, and second, that every probably in NP can be reduced to it in polynomial time. Proving that a problem is NP-complete is often taken as evidence that there is no efficient algorithm for solving it, since the existence of such an algorithm would imply the result (considered unlikely) that every problem in NP has an efficient decision algorithm.