International Journal of Computer Discovered Mathematics (IJCDM)
ISSN 2367-7775 ©IJCDM
Volume 3, 2018, pp.118-135.
Received July 30, 2018. Published on-line August 3, 2018
web: http://www.journal-1.eu/

Planar Tilings by Substitution Polykleins

Glenn C. Rhoads
3651 Karen Dr., Chesapeake Beach, MD 20732, USA
e-mail: glennrhoads@gmail.com

Abstract. The outstanding unsolved tiling problem is whether there exists a single tile that admits only nonperiodic tilings of the Euclidean plane. To search for such a tile, we use computer programs to enumerate the tiling behavior of substitution polykleins, a type of intuitively appealing candidate tiles, up through $n = 10$.

Keywords. polyklein, tiling, fundamental domain, aperiodic, nonperiodic

Mathematics Subject Classification (2010). 51-04, 68T01, 68T99.

1. Definitions

A *tiling* of the Euclidean plane is a countable family of closed sets called *tiles*, such that the union of the sets is the entire plane and such that the interiors of the sets are pairwise disjoint. Informally, the tiles fit together in the manner of a jigsaw puzzle; they don't overlap and they fill out the entire Euclidean plane. The tiles we consider will be closed topological disks — a set whose boundary is a single simple closed curve.

An *isometry* is any mapping of the Euclidean plane onto itself which preserves distances. A *symmetry* of a tiling T is an isometry that maps every tile in T onto a tile of T. Informally, think of the tiling as being covered by a clear sheet of paper on which each tile is outlined. A symmetry corresponds to a motion of the paper (including flipping the paper over) such that after the motion, the tracing fits exactly over the original drawing.

A *patch* is a finite collection of non-overlapping tiles such that their union is a closed topological disk. A *translational patch* is a patch such that the tiling consists entirely of a lattice of translations of that patch. A tiling is *periodic* if the group of symmetries contains at least two linearly independent translations, otherwise it is *nonperiodic*. Significantly, a periodic tiling always contains a translational patch. A *fundamental domain* is a translational patch of minimal size. While the

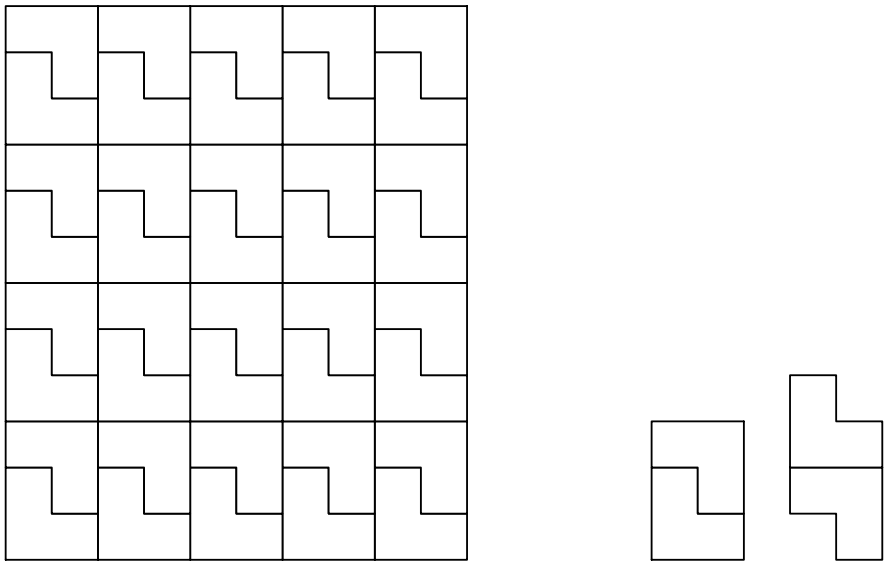
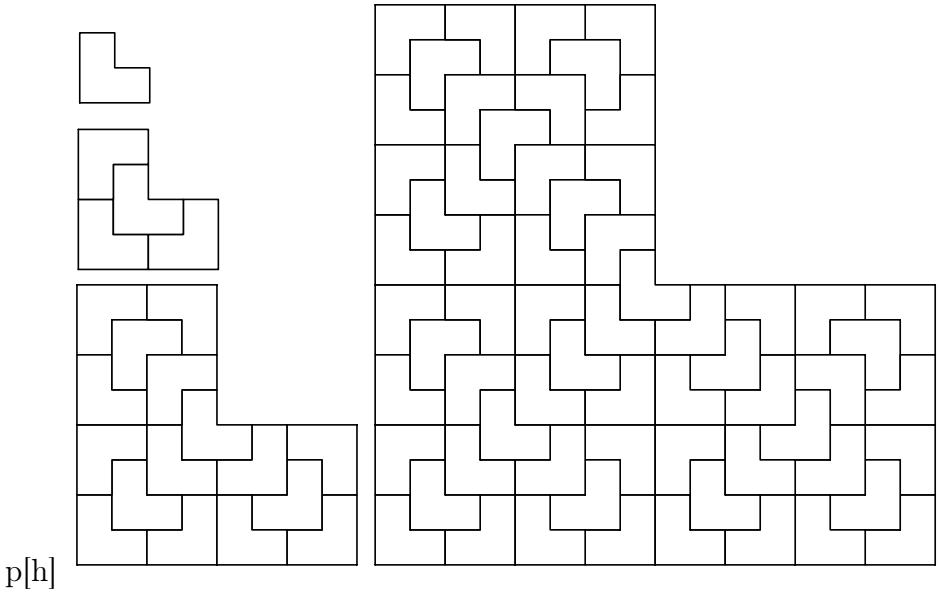


Figure 1. Two Fundamental Domains of a Tiling



p[h]

Figure 2. Successive Compositions of an L-shaped Tile

minimal size is well defined, Figure 1 illustrates that a tiling may have multiple fundamental domains.

Four copies of the L-shaped tile can be arranged to form a patch that has the same exact shape as a single L-tile but at a larger scale. Now treating the patch as a single tile, we can arrange four copies of this patch in the same manner to form a still larger patch that is similar to the original tile. By repeating this process indefinitely, we obtain a nonperiodic tiling of the plane called a *substitution* tiling, see Figure 2 (this tiling appears in many places).

This substitution tiling is nonperiodic ([14],[6]). A set of tiles is *aperiodic* if it admits tilings of the plane and if every such tiling is nonperiodic. While the L-shaped tile admits nonperiodic tilings, it is not an aperiodic set because it also admits periodic tilings. It is not at all obvious whether there are sets of aperiodic

tiles and it had long been assumed that there weren't any. But in 1966, R. Berger ([2]) discovered the first aperiodic set of tiles, a set of 20,246 tiles! Small sets of aperiodic tiles are rare. In 1974, Roger Penrose ([11], [12]) discovered a remarkable aperiodic set consisting of just two prototiles. This set has at least three different forms the most common of which seems to be the *kite* and *dart*.

Since then, the outstanding unsolved tiling problem is whether there exists an aperiodic set containing only a single tile. Socolar and Taylor have recently claimed to find such a tile ([17]) however, their definitions are looser. Their claim is correct only if you allow disconnected tiles or you allow matching conditions (essentially rules which restrict which edges can fit together, see Section 3) that relate non-touching tiles. The usual definition that we insist upon here is that the tiles be topological disks (and hence connected) and that matching conditions be enforceable purely by altering the shapes of the tiles. Under these conditions, the problem is open.

2. POLYKLEINS

A polytile is formed by sticking smaller congruent base tiles together. E.g. polyominoes, polyhexes, and polyiamonds are formed by attaching squares, regular hexagons, and regular triangles respectively along their edges. A **polyklein** is made from an equal number of copies of 1-sqrt(2)-sqrt(2) and 1-sqrt(2)-2 triangles, see Figure 3. For ease of description, we'll call a 1-sqrt(2)-sqrt(2) a **green** triangle and a 1-sqrt(2)-2 triangle a **blue** triangle.

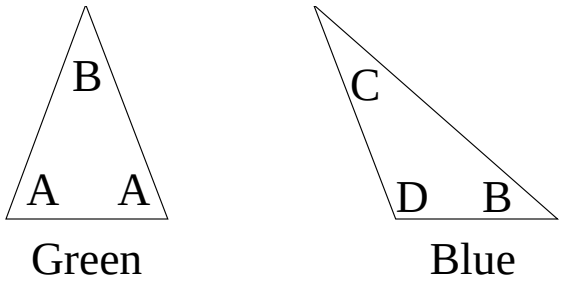


FIGURE 3. Klein Base Triangles

There are 13 distinct ways the angles A, B, C, and D can sum to 360 degrees. But we can simplify this by making use of the two following non-obvious angle identities D = A + B and A = B + C. Using these identities, we can express all angle types in terms of a linear combination of angles B and C. The only combination of angles B and C that sum to 360 degrees is 6 Bs and 4 Cs.

Now if we attach a green and blue triangle along an edge of length sqrt(2), we obtain a triangle similar to the green triangle but scaled up in size by a factor of sqrt(2) and if we attach them along an edge of length 1, we obtain a triangle similar to the blue triangle but scaled up in size again by a factor of sqrt(2). We can repeat this process indefinitely to produce a substitution tiling.

There is an intuitively appealing parallel between these tilings and the Penrose tilings. If you take the Penrose kite and dart, and cut them in half along the axis of symmetry (the matching condition is to make the newly formed edges be directed), you obtain the Penrose triangles which like the kleinian triangles can

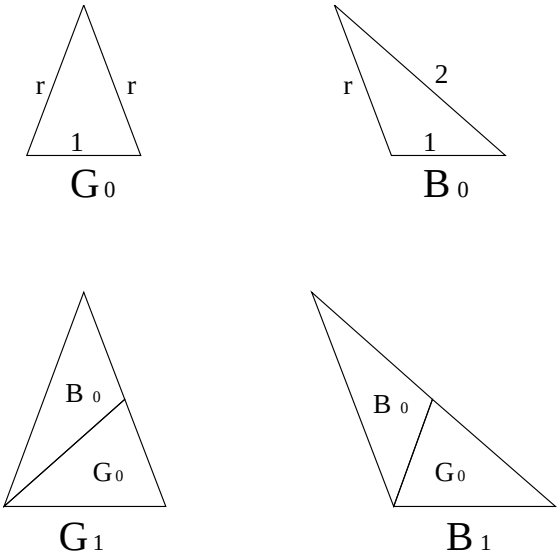


Figure 4. Composition of Kleinian Triangles

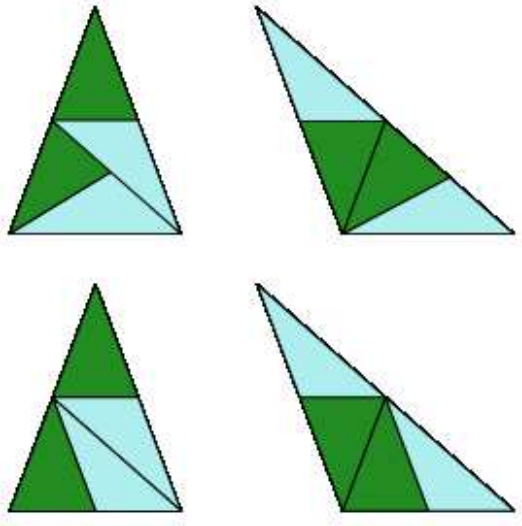


Figure 5. Kleinian Triangles of Level Two

be put together to form larger copies of themselves and hence form a substitution tiling (see Figure 8, D means half-dart and K means half-kite).

But with the Penrose triangles, the ratio of the half-kite to half-dart triangles is irrational (the *golden ratio*!) in *every* penrose tiling. This means it is impossible to form a tiling from a single tile that is made up from a finite number of half-darts and half-kites (the ratio in the entire tiling would be the same as the ratio in such a single finite tile which is rational by definition). However, the ratio of blue to green triangles in any kleinian substitution tiling is 1:1. Hence, there doesn't seem to be any reason why we couldn't form such a tiling from a single tile that is constructed from an equal number of green and blue triangles. The parallel with the Penrose tilings suggests that there may indeed be a single tile that forces such a tiling – i.e. an aperiodic tiler! I.e. they are an intuitively appealing candidate to search for such a tile and we've written some tiling code to do so.

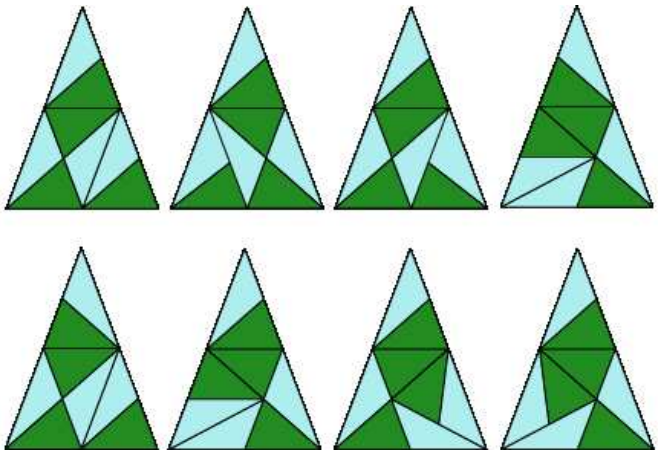


FIGURE 6. Green Kleinian Triangles of Level Three

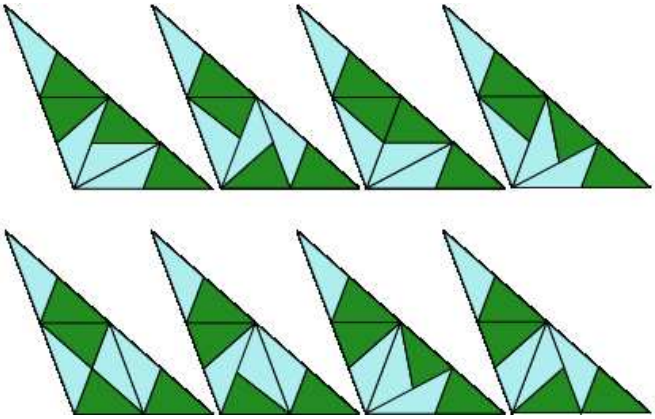


FIGURE 7. Blue Kleinian Triangles of Level Three

Theorem 2.1. *The only non edge-to-edge match that can occur in a Kleinian substitution tiling is to match a base edge of length two against two base edges of length one.*

Note that matching a base edge of length two by the left or right half of a base edge of length two and any other length one edge, including half of a length two edge, is also ruled out by the theorem. We'll omit the proof for brevity (see [14], pp.81–84).

We handle the above situation when constructing the tiles from the base triangles by considering length two edges to be two consecutive edges of length one, and putting a matching condition on the edges originating from length two edges. The matching condition prevents everything ruled out by the theorem and allows everything that is not ruled out.

3. MATCHING CONDITIONS

Small aperiodic sets generally contain matching conditions that specify which equal length edges can be matched. In order to keep the tiles as simple as possible matching conditions are usually specified by techniques such as coloring edges or corners and requiring that colors on adjacent tiles match. It is vital that matching

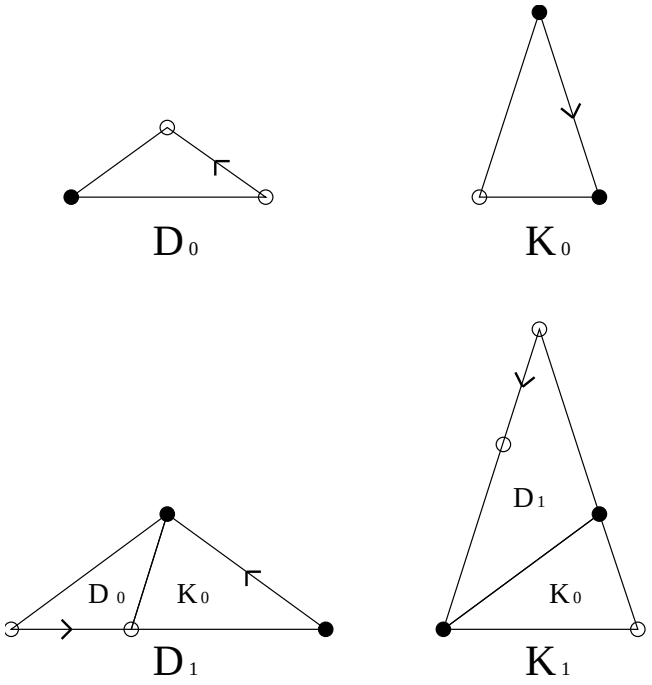


Figure 8. Composition of the Penrose Triangles

conditions be enforceable purely by altering the shapes of the edges. The software can handle three different types of matching conditions.

C-type: An edge has a centrally-symmetric protrusion or the corresponding indentation:

S-type: An edge has a curve (polygonal curves are easier to draw) that is symmetric with respect to 180 degree rotation. We use two distinct S-type markings which are reflections of each other. An edge with an S-type marking matches an edge of the same length with the same marking, since an edge and its match are oriented 180 degrees from each other.

J-type: An edge has one of two asymmetric protrusions which are reflections of each other, or one of the two matching indentations. Note that the matching indentation may at first appear to match instead the reflected form, until you realize that a matching edge is oriented 180 degrees from the original.

The exact shape of a matching condition's protrusion/indentation does not matter. For example, in a C-type matching condition any centrally-symmetric shape could be used. You could attach say a small equilateral triangle to the middle of an edge to produce a protrusion. The matching indentation would be formed by removing a congruent equilateral triangle from the middle of an edge of the same length so that two edges fit together in a lock and key manner. Obviously, there is nothing special about using an equilateral triangle as a protrusion/indentation. Any other centrally symmetric shape would work just as well.

The only edges that needs special consideration during the generation phase are the length one edges that come from an original blue triangle edge of length two. The blue triangle exists in two distinct forms that are reflections of each other. Traverse the edges of both of these blue triangles clockwise. Represent the length two edge as a sequence of two distinct length one edges where the first edge has a J-type protrusion and the second edge has the matching J-type indentation. It is

important to use the same protrusion and indentation on both forms of the base blue triangle. Now you generate polykleins by looking at all ways of putting an equal number of each type of base triangle together allowing only edge-to-edge matches. If both edges of a potential matching pair have a matching condition on them, then you check the matching condition to see if it is allowed. If one or both of the edges does not have a matching condition on it, then you simply ignore the matching condition (i.e. you allow the match if they really do match without the tiles overlapping).

We ignore the matching conditions entirely when determining which ones tile the plane and which ones do not. Then we take the polykleins that tile periodically, and attach matching conditions to the edges and then determine the tiling status of those tiles. If some of the length one edges already have a matching condition on them due to their generation, then we try all possible ways of putting J-type matching conditions on all of the length one edges. For each of these, we try on the length sqrt(2) edges all combinations of C-type matching conditions, all combinations of S-type matching conditions, and all combinations of J-type. If none of the length one edges has a matching condition on it (the base edges with a matching condition could be internal to the tile and not on its outerboundary), then we try all possible combinations of C-type, S-type, and J-type matching conditions on the length one edges and for each of these, we independently try all combinations on the length sqrt(2) edges.

Also, for C-type and J-type matching conditions, the number of protrusions and indentations must be **equal** or else the tile cannot tile the plane. The basic idea of the proof is as follows. Suppose there is an excess of one type of protrusion over the matching indentation. In any patch of tiles, the internal edges must contribute the same number of protrusions as indentations and thus, the entire excess must be taken up by edges on the outerboundary of the patch. The amount of excess is proportional to the number of tiles in the patch and hence, is proportional to the patch's area. The number of edges on the outerboundary depends on the patch's perimeter. Consider a sequence of increasingly large circles that become arbitrarily large. Each circle with radius r contains a circle C whose radius is at least $r-\sqrt{2}$ such that there exists a patch of an equal number of base kleinian tiles that completely contains circle C. The area of the patches increases quadratically with the radius while the perimeter increases only linearly (polytiles have "nice" non-fractal boundaries). Thus, at some point the amount of the excess must exceed the number of outerboundary edges that can take up the excess and this is impossible. To complete the proof you need to take limits in the manner of the well known proof that the ratio of kites to darts in a Penrose tilings is the golden ratio. For brevity, we do not repeat that part of the proof here. As a consequence, there is no need to consider tiles where the number of C and J type protrusions does not equal the corresponding number of matching indentations.

Finally, we did not bother trying to eliminate matching conditions that imply identical tiling behavior. For example, if you take a tile and replace every C-type protrusion with the corresponding indentation and vice-versa, then the tile must have the exact same tiling behavior as before. Trying to eliminate matching conditions that imply identical tiling behavior in complete generality would be a royal pain if it is even doable. It doesn't seem worth the bother of trying.

4. Preprocessing Phase

Theorem 4.1. *([6], chap.3) Let T be a finite set of prototiles each of which is a closed topological disk. In any tiling admitted by T, there must be some tile with no more than 6 neighbors.*

Theorem 4.1 implies that when trying to surround a tile once, we can limit the search depth to 6. Whether this is guaranteed to find all distinct planar tilings depends on what you mean by distinct. To illustrate this, consider the planar tiling with the translational patch shown in Figure 9. This tiling has two equivalence classes of rotationally equivalent tiles, the clear tiles and the shaded ones. Each clear tile is surrounded by 4 tiles and each shaded tile is surrounded by 7 tiles. Thus if you start with an initial tile that is rotationally equivalent to a clear tile, you will not find this tiling by trying to surround the tile with up to 6 copies. However you will encounter the reflection of this tiling. If you imagine flipping this tiling over, then all the shaded tiles will become rotationally equivalent to the initial tile and hence, you will encounter this reflected tiling. One could regard a planar tiling and its reflection as the same tiling. If you do, then limiting the number of tiles for the first surrounding to 6 is guaranteed to find all "distinct" tilings. But if you regard only rotations and translations as distinct, then for each pair of "distinct" reflected tilings, you are guaranteed of finding at least one but not necessarily both tilings.

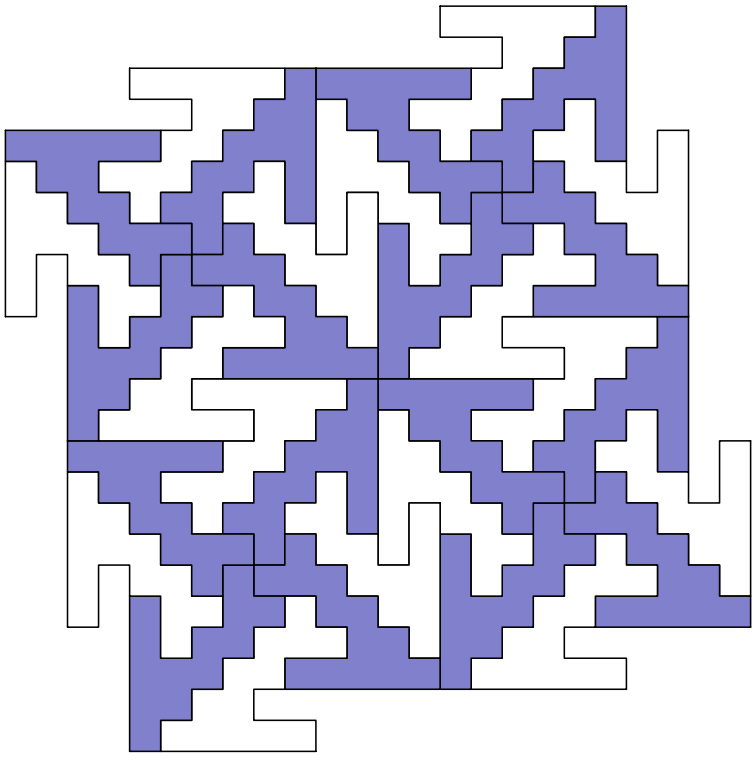


Figure 9. Insufficiency of Examining only Six Neighbors

We can refine the process of surrounding the tile by first doing a preprocessing phase. A *portion* of the initial tile is a set of consecutive edges on its boundary. For each portion, we keep a list of all ways to match that portion. So we try all ways of matching the initial tile with edges of another copy. For each match, determine

the portion of the initial tile and store on that portion's list, the matching edges and orientation of the other tile. Once you have all of the portions of the boundary that can be matched by a single tile, you can combine two adjacent portions to get all the portions of the tile that could potentially be matched by two tiles. Similarly, you can then get all portions that could potentially be matched by 3 tiles, 4 tiles, 5 tiles, and 6 tiles. Whenever the matched portion is the entire boundary of the tile, we store this in the *surround list* which contains all the ways of breaking up the boundary into 6 or fewer portions such that each portion can be matched by some copy of the tile. For each item on the surround list, we search over all ways of matching the individual portions using chronological backtracking. Of course for each portion of the boundary, we must be able to index the corresponding list of all ways to match that portion. (This preprocessing phase was suggested by John Conway ([4]).

5. Conway and Translation Criteria

A closed topological disk satisfies Conway's criterion if you can divide up the boundary into six segments labeled clockwise A, B, C, D, E, F such that

(1) A and D are translations of each other, and
(2) B, C, E, and F are symmetric with respect to a 180 degree rotation about their center point.

At least one edge of each of the pairs B-C and E-F must be non-empty. Also, both segments A and D could be empty if at least three of the remaining four segments are nonempty.

Theorem 5.1. *Any tile satisfying Conway's criterion admits a periodic tiling of the plane (and does so using only translation and* 180° *rotation).*

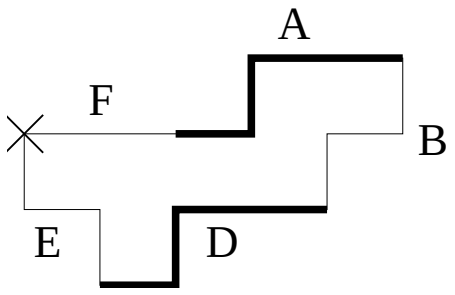


FIGURE 10. A Heptomino Satisfying Conway's Criterion

To illustrate theorem 5.1, consider the heptomino shown in Figure 10. This heptomino satisfies Conway's criterion. Segments A and D correspond to the heavy lines and these segments are translations of each other. Segment B on the right is symmetric with respect to a 180 degree rotation (segment C is empty). The other segment can be broken up into two pieces, E and F, that have 180 degree symmetry – the point where you break up this piece into two is marked with an x.

To tile the plane, translate this heptomino so that segments A and D line up and then repeat this to get the infinite strip shown in the left part of Figure 11. Take a copy of this strip, rotate it 180 degrees (the middle part of Figure 11), and

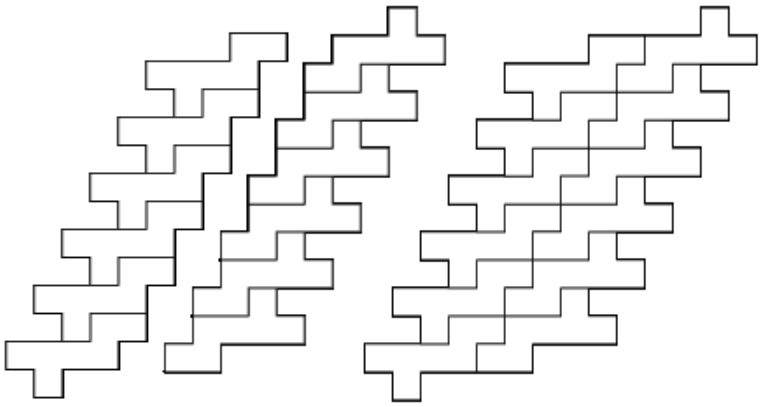


Figure 11. Illustration of Conway's Criterion

fit it next to the original strip so that the segments with 180 degree rotational symmetry line up. Now you have a two-piece wide infinite strip that tiles the plane by translation (the right part of Figure 11). If segments A and D are empty, then you can repeat this same procedure while thinking of segments A and D as being points – see the example heptomino in Figure 12 (It is impossible to divide up the boundary of this heptomino so that it satisfies the Conway criterion where segments A and D are non-empty.)

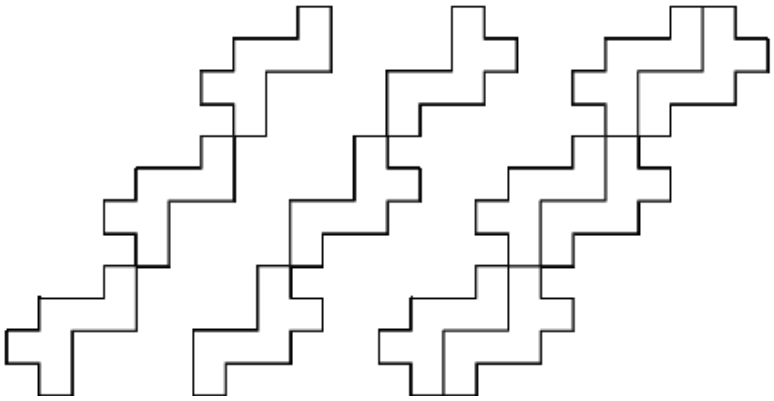


Figure 12. Illustration of Conway's Criterion with Empty Parallel Segments

Conway's criterion is surprisingly powerful. For example, out of the 104 heptominoes that tile the plane, 101 of them satisfy Conway's criterion, and out of the 343 octominoes that tile the plane, 320 of them satisfy Conway's criterion. It is not clear why this criterion is so powerful. The simplicity and power of this criterion makes it a very useful criterion to code up in a computer program.

A closed topological disk satisfies the translation criterion if you can divide up the boundary into six segments labeled clockwise A, B, C, D, E, and F such that each of the three pairs A-D, B-E, and C-F are translations of each other (both edges in one of these pairs may be empty).

Theorem 5.2. *A prototile that is a closed topological disk admits a tiling of the plane by a lattice of translations if and only if the prototile satisfies the translation criterion.*

A prototile satisfying the translation criterion admits a lattice-translation tiling simply by translating copies so that the edges in the each pair line up. If one of the pairs is empty, then the tiling forms a rectangular lattice, otherwise it forms a hexagonal lattice.

6. Coding the Conway and Translation Criteria

We need to check whether distinct portions of the boundary are translations of each other or whether they are centro-symmetric. In order to avoid doing this repeatedly, we precalculate all such portions and store the results. Then when iterating over possible places for dividing up the boundary, we can easily check whether the individual portions satisfy the necessary conditions.

Centro-symmetric portions have a palindromic structure. That is if we assign a distinct symbol for each set of edges that are parallel and of the same length, then a portion is centro-symmetric if and only if the corresponding sequence of symbols is a palindrome.

Identifying the centro-symmetric portions and the pairs of translation portions both require that we identify pairs of edges that are parallel and of the same length. Hence, we check this for each pair of distinct edges and store the result. When identifying centro-symmetric portions, we iterate over possible central places which is either an edge or a vertex, and then extend the portion outwards in both direction as far as possible while the portion remains centro-symmetric. When identifying pairs of translation portions, we iterate over possible starting edges of the first portion. For each starting edge, we iterate over possible ending edges of the second portion. As long as the pair remains a translation of each other, we extend the first portion *forward* by one edge and the second portion *backwards* by one edge. Note that since we are traversing the tile edges clockwise, moving forward from the first portion and backwards from the second portion identifies corresponding edges.

Once we have identified the translation pairs and the centro-symmetric portions, we test for the Conway and translation criteria by iterating over possible starting points for portions A and D, extending each as long as they remain a translation pair and then checking if there is a way to divide up the remainder so that it satisfies either the Conway or the translation criterion (we check the Conway criterion first because it is more powerful).

Finally, when using tiles with matching conditions, we must restrict each pair of translation portions to those where every corresponding pair of edges has matching conditions that "match". When identifying centro-symmetric portions with matching conditions, the corresponding edges must be restricted to those having an S-type matching condition since these are the only type that is symmetric with respect to 180 degree rotation.

7. Numeric Computations

Testing whether edge lengths match and whether angles meet in valid ways at vertices of the tiling is done purely symbolically. With tiles with a regular underlying lattice such as polyominoes and polyhexes, we can nonnumerically test whether a tile and a patch intersect simply by checking whether there is a lattice cell in common to both. Polykleins have the underlying lattice shown in Figure 13 (the

solid lines are the lattice lines and the dashed lines show the additional edges that can occur when a polyklein is placed with their corner points at the lattice points (Also the underlying Kleinian lattice is the inspiration for the term polyklein which is due to John Conway [4]). The tiling behavior of polykleins restricted to the kleinian lattice was investigated in the author's thesis without any interesting tilings being discovered [14]. Restricting the tiles to the underlying lattice results in a finite number of edges types. Hence, these tiles can be handled symbolically. However, polykleins do not in general correspond with the underlying lattice. In the general case, you cannot limit the number of edge angles to a finite number. Hence, testing whether a tile overlaps the patch must be done numerically which is more involved and more computationally expensive.

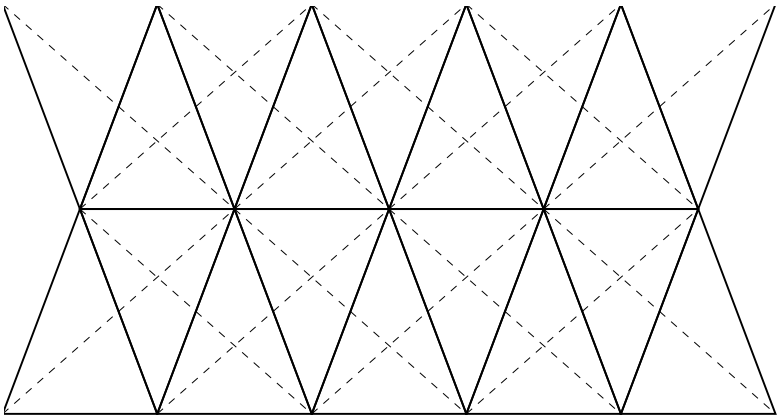


Figure 13. Kleinian Lattice

We assign numeric coordinates to the vertices of the tiles. When checking whether a potential match is valid, we move the tile and numerically check for overlap. Since numeric computations aren't exact, we use a numerical tolerance and when two coordinates differ by less than the tolerance, we regard them as the same. Getting this method to work in geometric applications typically requires great care and vigilance to prevent numerical inaccuracies from causing incorrect results. As an example, we store a separate copy of a polyklein's original coordinates. Whenever we move the polyklein and test for overlap, we reset the polyklein's coordinates back to their original value. This prevents any numerical errors introduced by moving a tile from accumulating. Additionally, we wrote a second version using the high precision QD library ([1]). This library uses four double precision floating point numbers to represent a single number. The QD library extends the usual 16 decimal digits of accuracy to about 64 digits, and the numeric operations are highly optimized for the package's representation. Using the QD package, we increased the numerical tolerance from one part per million to one part per quadrillion and reran the programs for all polyklein sizes except for the last row in Table 1. The results were exactly the same as before which gives us confidence that the numerical precision problems were handled adequately. Lastly, we wrote a program that eliminates the roundoff error in the coordinates of the generated tiles. There are only three types of roundoff error that can occur in the coordinates of the tiles. We can have a sequence of several zeros or several nines where the last two or three digits could something besides a 0 or 9. In these two cases, it is rather obvious what the true value is. The only other type of roundoff error is when the y-coordinate is a rational fraction of the square root of

7. The denominator of these values must be a power of 2. The highest power of 2 attainable depends on the number of base tiles in the polyklein. The difference between two successive true values is far more than necessary to tell which is the correct true value – the rounded off values differ at most in only the last two or three digits.

7.1. **Moving Tiles.** When testing whether a particular edge of a tile matches a particular edge of a patch of tiles, we need to move the tile so that potentially matching edges are aligned. Translating a tile is simple. The standard formula for rotating by angle θ uses both $\cos\theta$ and $\sin\theta$. An important observation is that we can calculate $\cos\theta$ and $\sin\theta$ without using any trig functions and without even determining θ. In fact, we need only the basic arithmetic operations of addition, subtraction, and multiplication! From the law of cosines,

$$\cos\theta = \frac{a^2 + b^2 - c^2}{2ab}$$

In our case, the two legs are of equal length (since the tile edge and the matching patch edge must have the same length), hence this reduces to

$$\cos\theta = \frac{2a^2 - c^2}{2a^2} = 1 - \frac{c^2}{2a^2} = 1 - .5 \star \frac{c^2}{a^2}$$

In our case, the only two possible edge lengths are 1 and $\sqrt{2}$. Hence, $\frac{1}{a^2} = 1$ or $\frac{1}{a^2} = \frac{1}{2}$. In summation, our formula reduces to

$$\cos\theta = 1 - .5 \star l \star c^2$$

where l is either 1 or $\frac{1}{2}$ depending on the value of a. c^2 is easily calculated from the standard distance formula $c^2 = (\Delta X)^2 + (\Delta Y)^2$.

We could calculate $\sin\theta$ from the formula $\sin\theta = \sqrt{1 - \cos^2\theta}$ but $\sin\theta$ can be calculated more efficiently without the numerical precision problem inherent in taking roots by using the formula $\sin\theta = 2lA$ where A is the signed area of the triangle formed by the initial and final location of the matching edge, and the edge opposite angle θ. The signed area of the triangle whose vertices are $(x_1, y_1), (x_2, y_2)$, and (x_3, y_3) is

$$(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - y_1)$$

7.2. **Intersection Test.** There is a standard algorithm for testing whether two polygons intersect which is based on testing for line segment intersections (see e.g. [13]). The algorithm uses the "sweep-line" paradigm where the line segments that intersect the current position of the sweep-line are stored in sorted order in a balanced binary tree (e.g. an AVL or Red-Black tree) with predecessor and successor threads. However, the algorithm doesn't work in the presence of degeneracies. When the tile being matched against the patch has a vertex that is at the same place as a vertex of the patch, we have four line segments that intersect at a single point. When the sweep-line intersects this point, the balanced binary tree will need to store the corresponding line segments in *sorted* order. But it is impossible to define a total ordering among these line segments. Hence, the standard algorithm needs to be modified. Using a balanced binary tree scheme is overkill for this application because the tree will usually hold only a few items.

Even for very large patches with many edges, it's hard to imagine the sweep-line ever intersecting the patch in more than 10 or 12 edges. With this few items, we can maintain a set of sorted items more efficiently and much more simply using a basic scheme. We used a doubly-linked list where equal items are stored consecutively. A doubly-linked list easily lets you access an item's immediate predecessor and successor as required by the algorithm. Also when we need to test a line segment for intersection, we test that particular line segment plus all "equal" line segments. This is the only change required to get the standard algorithm to work in the presence of degeneracies.

A couple of other subtle points need mentioning. First, two edges that are both from the tile being added or both from the patch, are allowed to intersect each other at a common endpoint. Thus, we distinguish tile edges from the patch edges by adding a label to them. Additionally, a tile edge and a corresponding patch edge which are both adjacent to the portion in the common intersection between the tile and patch, intersect at the endpoint of the portion. To allow this we attach a pointer on the patch edge which points to the matching tile edge. Having the pointer run in the opposite direction is insufficient! When the tile fits into a concavity of the patch, it is possible that all edges of the tile except one fit against a patch edge. In this case, the same tile edge is adjacent to *both* endpoints of the matching portion. This edge has an allowable intersection with *two* distinct patch edges. A single pointer attached to the tile edge cannot point to both of these patch edges. Hence, the need to have the pointers run from the patch edge to the matching tile edge. (Actually, we use pointers in both directions which is simpler to code.) Finally, the primitive which tests whether two line segments intersect is surprisingly tricky to get just right in all cases. We use the method described by Professor Paul Bourke which we modified to handle numerical tolerances ([3]).

8. Testing for Matching Edges

0: initialize all potential matches to false

Function: Check-Match()

(1) If the edges have been previously checked, then return false.
(2) If the edges have different lengths, then return false.
(3) Record the pair of edges as having been checked.
(4) Move the tile so that the potentially matching pair of edges are aligned.
(5) Extend the match as far as possible in both directions while recording the pairs of edges in the extensions as being checked. If the potential match cannot be extended (e.g. edges have different lengths or angles), then return false.
(6) If doesn't pass the Tile-Patch intersection test, then return false.
otherwise return true.

When the corresponding edges match (in the function Check-Match), we have to extend the matching region as far as possible to find the matching portions of the corresponding boundaries. When doing this, we may find other pairs of matching edges. So in order to avoid regenerating the same boundary portions for each pair of matching edges, we record these matches in an array. This is the reason for the

initialization in step 0, the check in step 1, and for recording the matching edges in steps 3 and 5.

9. The Basic Algorithm

The basic idea of the algorithm is to place a matching tile against some edge(s) of the starting tile, then place a second copy against the next available clockwise edge(s) of the starting tile, and continue adding tiles in a spiral-like manner using chronological backtracking until you determine whether there is a periodic tiling, no tiling, or you hit an arbitrary limit on the number of added tiles.

For the first surrounding, we use the preprocessing information to backtrack over the ways of dividing up the entire boundary into 6 or fewer potentially matchable portions. We don't start at an arbitrary portion of the boundary. In the preprocessing phase we keep track of the number of ways to match each single portion – i.e. the size of the matching list. We then start the backtracking at the first portion that minimizes the product of the size of the matching list with the size of the next portion's matching list. This heuristic keeps the initial branching factor relatively small which reduces the size of the implicit search tree. If we successfully surround the tile, we then check whether the patch of tiles has an unfillable concavity. We define a concavity as a left turn when traversing the boundary clockwise. Empirically, this unfillable concavity check quickly eliminates the vast majority of patches (see for example [15]).

The main tiling program was split into two versions; a simple version that quickly identifies most tiles either as having a planar tiling or as not having any tiling, and a more powerful version that is used only for those tiles whose status in undetermined by the simple version. Empirically, many of the tiles whose status can be quickly determined make the more powerful version run quite a bit slower, hence the reason for splitting the program into two versions.

The simple version performs three main tasks. First, it tests whether the tile has a periodic tiling using the Conway and translation criteria described in Section ref-Periodic. Then it performs the preprocessing. Finally, it tries to find a way to surround the tile once by backtracking over the ways to divide up the boundary into 6 or fewer portions. Whenever the tile gets surrounded, we check whether the patch has an unfillable concavity. As soon as we find one way to surrounded the tile that doesn't have an unfillable concavity, we put this tile in the undetermined file which will be used as input for the more powerful version.

The more powerful version doesn't check whether the tile satisfies our periodic tiling condition since the simple version already performed this check. The preprocessing is done as before with following changes. For each valid match, we combine the two tiles into a single tile. Then we check whether this tile-pair satisfies our periodic tiling criterion. In this way, we check whether there is a way of putting two copies of the tile together that satisfies our periodic tiling criterion. If the tile-pair isn't periodic, we test whether the tile-pair has an unfillable concavity. This slows up the preprocessing but it reduces the number of potential matches to search over which greatly speeds up the search for more difficult tiles that require a deeper search. We perform the periodic test on the pair first because empirically many of the tiles with a periodic pair are precisely those tiles that cause the unfillable concavity check to run much slower.

The search part of the more powerful version is modified as follows. Whenever we form a patch consisting of three or more tiles, we test whether the patch satisfies the translation criterion. You would naturally think that it would be better to use the Conway criterion but we are performing this check only on those patches where the base tile is such that it does not satisfy the Conway criterion and such that there is not a way of putting two copies of the tile together that satisfies the Conway criterion. For these more difficult tiles, the Conway criterion is empirically next to useless as a periodic test on the patches (see e.g. [15])

Whenever we have surrounded a tile once such that the patch does not have a unfillable concavity, we continue the backtracking search in a spiral-like manner up to an arbitrary pre-defined limit on the number of added tiles. An important subtlety is that when considering potential matches to a patch edge, we must consider whether the tile containing the patch edge is oriented the same way as the starting tile or whether it is in a reflected (i.e. flipped) orientation. If the patch tile is oriented the same way, we traverse over the list of potential matches that *start* at the patch edge, otherwise we traverse the list of potential matches that *end* at the patch edge. These lists are made as part of the preprocessing.

To speed up the backtracking search, we represent the outerboundary of a patch of tiles (or of a single tile) by a circular doubly-linked list of edges. This representation lets you add and remove tiles simply by splicing edges into and out of the list; i.e. this avoids the repeated and expensive copying of data that is usually required by chronological backtracking – you need the copy so that you can back up to the previous state. This idea comes from Donald Knuth's "Dancing Links" paper ([8]).

Another efficiency improvement we use is to avoid the operating system overhead of dynamic memory allocation by allocating large enough data stores at the beginning and then manually controlling the allocation and deallocation from the stores. The one exception to manual control of space occurs during the tile generation. The code was developed on a system with the Windows Vista operating system. This operating system fragments the process space so that although there is enough space available for a free store of tiles, there is not a large enough *contiguous* block (this is a very basic error in the operating system which as of this writing needs to be fixed). Hence, we simply use dynamic memory allocation for the space needed to store the generated tiles.

10. Results and Conclusions

Table 1. Tiling Status of the Polykleins up through order 10

N	period	pairs	nontilers	total	MCper	MCpair	MCnontiler	MCtotal
2	3	1	1	5	16	16	24	56
4	105	23	281	409	4,886	7,762	64,896	77,544
6	2,632	779	41,671	45,082	30,968	12,230	996,594	1,039,792
8	35,360	12,764	5,711,326	5,759,450	3,163,680	4,012,811	2,933,384,759	2,940,561,250
10	85,117	19,855	48,132,403	48,237,375	5,013,086	1,122,676	867,508,250	873,644,012

The meaning of the columns in Table 1 are as follows. N is the number of base triangles in the polyklein; *period* is the number of tiles that satisfy either the Conway or translation criterion; *pairs* is the number of tiles such that two copies of the tile can be combined in a way that satisfies either the Conway or translation

criterion; *nontilers* is the number of tiles that do not have a planar tiling; and *total* is the total number of tiles of size N. The columns with the prefix MC have the same meaning except that the results apply to the tiles with matching conditions, that is, all ways of placing the matching conditions described in Section 3 to the tiles that have a planar tiling.

The first unintuitive result shown in Table 1 is that there are over twice as many tiling 10-kleins as 8-kleins, yet the higher number of tiling 10-kleins results in only about one third as many matching condition tiles as the tiling 8-kleins do! This result seems so wrong that we reran the program on an entirely different machine and obtained the identical results. However, upon examination of tiling 8 and 10 kleins, the reason becomes clear. All of the tiling 8-kleins have an even number of edges on their outerboundary while all of the tiling 10-kleins have an odd number. For C and J type matching conditions, the number of protrusions must equal the number of matching indentation. Hence, the number of edges with C or J type matching conditions must be even. Since the tiling 10-kleins have an odd number of edges on their boundary, you must use an odd number of S type matching conditions. There are two possibilities, there are an odd number of length 1 edges each of which must have an S type matching condition or there are an odd number of length sqrt(2) edges each of which must have an S type matching condition. This restriction greatly reduces the number of possible matching conditions.

The other counter-intuitive result is that the ways a polyklein can tile the plane are disappointingly simple. Every tile up through N=10 that tiles the plane does so because it satisfies either the Conway or translation criterion or because there is a way to combine two copies of the tile that satisfies one of the criteria. This is not true for other types of tiles made from combining copies of base tiles. For instance, there are polyhexes of size N=7 that tile the plane but the tile does not satisfy either the Conway or translation criterion nor is there a way of combining two copies that satisfies one of the criteria. I.e. all tilings are more complicated. At only N=11, we encounter a mind-blowing polyhex where the fundamental domain of its unique tiling contains an astonishing 36 tiles! ([15]). So the obvious question is why aren't there tiling polykleins that do not have a simple tiling? There is no apparent reason why such tiles shouldn't exist; at present this is a real mystery.

The difficulty with extending the polykleins to larger sizes is the space required for generating the tiles. With polyominoes, polyhexes, and polyiamonds, all tiles correspond to a regular lattice and hence, you can generate one copy of each distinct tile without storing all the tiles ([10]). But with polykleins, the tiles do not have to correspond to any lattice and hence, there does not seem to be a way to generate all the tiles of a given size without explicitly storing them (you need to store them in order to determine whether a newly generated tile is distinct from the previously generated tiles).

Due to this space problem, a natural thought is to try to restrict the tiles to a very small subset that is likely to contain the interesting tiles whose simplest tiling is complicated. A natural idea is to restrict the polykleins to those whose corner points are aligned with the kleinian lattice. This has already been done ([14]) up through N=12 without any interesting tilings being discovered. Unfortunately, it doesn't seem possible to restrict the tiles to only a very small subset that contains the most interesting tiles. When you examine the polyominoes, polyhexes, and polyiamonds ([9]), the most interesting tiles do not seem to have any distinguishing

properties or qualities; they look pretty much like any other tile. There is no reason to suspect that this wouldn't also be true of the polykleins and hence, it seems unlikely that you could put your finger on the most interesting examples without examining most or all of the tiles.

Finally, all of the software used to obtain the results shown in Table 1 are available upon request.

11. Acknowledgments

A special thanks to Laurent Bartholdi for providing the computational resources to determine the $n = 10$ case using my software.

References

[1] David H. Bailey, Yozo Hida, and Xiaoye S. Li. Lawrence Berkley National Laboratory (Univ. of Calif.). http://crd.lbl.gov/ dhbailey/mpdist/

[2] R. Berger. The undecidability of the domino problem. *Memoirs American Mathematical Society*, 66, 1966.

[3] Paul Bourke. http://local.wasp.uwa.edu.au/pbourke/geometry/lineline2d

[4] John H. Conway. personal communication.

[5] Martin Gardner. *Penrose Tiles to Trapdoor Ciphers*. W. H. Freeman and Company, 1989.

[6] Branko Grünbaum and G. C. Shepard. *Tilings and Patterns*. W. H. Freeman and Company, 1987.

[7] H. Heesch. *Reguläres Parkettierungsproblem*. Westdeutscher, Cologne and Opladen, 1968.

[8] Donald E. Knuth. http://www-cs-faculty.standford.edu/ uno/preprints.html.

[9] Joseph Myers. http://www.srcf.ucam.org/ jsm28/tiling/.

[10] D. H. Redelmeier. Counting polyominoes: Yet another attack. *Discrete Mathematics*, 36:191–203, 1981.

[11] Roger Penrose. The role of aesthetics in pure and applied mathematical research. *Bulletin Inst. of Mathematics and its Applications*, 10:266–271, 1974.

[12] Roger Penrose. Remarks on a tiling: Details of a $(1 + \epsilon + \epsilon^2$-aperiodic set. In R.Ṽ. Moody, editor, *Proc. of NATO ASI Series C 489 – The Mathematics of Long Range Aperiodic Order*, pages 467–497, 1997.

[13] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry*. Springer-Verlag, New York, 1989.

[14] Glenn C. Rhoads. Planar tilings and the search for an aperiodic prototile. *Doctoral Dissertation at Rutgers University*, May 2003.

[15] Glenn C. Rhoads. Planar tilings by polyominoes, polyhexes, and polyiamonds. *Journal of Computational and Applied Mathematics*, 174:329–353, 2005.

[16] Doris Schattschneider. *Visions of Symmetry*. W. H. Freeman and Company, New York, 1990.

[17] Joshua E. S. Socolar and Joan M. Taylor. An aperiodic hexagonal tile. Submitted to the *Journal of Combinatorial Theory*, Series A.