Picking Knots from Trees[*]
The Syntactic Structure of Celtic Knotwork

Frank Drewes[1] and Renate Klempien-Hinrichs[2]

[1] Department of Computing Science, Umeå University
S-901 87 Umeå (Sweden)
Email: drewes@cs.umu.se

[2] Department of Computer Science, University of Bremen
P.O.Box 33 04 40, D-28334 Bremen (Germany)
Email: rena@informatik.uni-bremen.de

Abstract. Interlacing knotwork forms a significant part of celtic art. From the perspective of computer science, it is a visual language following mathematically precise rules of construction. In this paper, we study the syntactic generation of celtic knots using collage grammars. Several syntactic regulation mechanisms are employed in order to ensure that only consistent designs are generated.

1 Introduction

A typical characteristic of visual languages is that the diagrams in such a language are related by a common structure and layout. In other words, the language is defined by a set of syntactic visual rules yielding the acceptable pictures. Formal picture-generating methods help to understand the structure of the languages in question, to classify them, and to generate them automatically by means of computer programs.

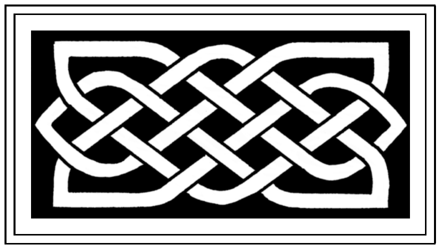


Fig. 1. A celtic knot

Artists from many cultures have been using visual rules since ancient times in order to design diagrams of various sorts. Celtic diagrams, and in particular

[*] Research partially supported by the *Deutsche Forschungsgesellschaft* (DFG) under grant no. Kr-964/6-1, the EC TMR Network GETGRATS (General Theory of Graph Transformation Systems), and the ESPRIT Basic Research Working Group APPLIGRAPH (Applications of Graph Transformation).

celtic knotwork, are a famous visual language of this type. Figure 1 shows an example of a celtic knot. Basically, such a diagram is the two-dimensional picture of one or more continuous strands weaved in a particular fashion. If we look at two successive crossings along a particular strand, this strand lies on top of the first strand crossed if (and only if) it passes below the second. Furthermore, the distances and angles which appear are determined by an invisible grid.

Traditional methods to construct celtic knotwork are described in e.g. [Bai51, Mee91,Buz]. Mainly, one first draws a grid of squares, triangles, or a similar pattern. This grid is used to obtain a plait, i.e., a knot with regular interlacing, which is subsequently modified by breaking crossings and connecting the loose ends. In the last step, the style of the strands and the background is determined. Although one can identify certain characteristic features of celtic knots as well as typical ways to construct them, the description of celtic knotwork as a visual language remains informal.

In [Slo95], Sloss presents an algorithmic way to generate knots. An introduction to celtic knotwork from the perspective of computer graphics is given by Glassner in [Gla99a,Gla99b,Gla00]. In the present paper, a formal description for celtic knotwork is provided by means of *collage grammars* [HK91,HKT93, DK99], one of the picture-generating devices studied in computer science. Using collage grammars in the form of [Dre00b], we shall describe (a picture of) a knot by a term, i.e., an expression over graphical operations and primitives. Such a term corresponds to a derivation tree in a collage grammar, the value of the term (the result of the derivation tree) being the generated knot. Thus, the tree describes the syntactic structure of the knot, whereas the evaluation of the tree yields the actual knot. In particular, two knots may share their syntactic structure although their visual appearance differs—which is reflected in the formal model by the fact that the underlying trees are identical, but the symbols are given different interpretations as picture operations.

Reasoning about the generation of picture languages is at the same time easier and more enjoyable if there is a suitable system supporting the work. For this paper, all knotwork designs (except for the first, hand-drawn, knot) were produced using the system TREEBAG [Dre98]. With the exception of some very basic examples, we shall not present in detail the grammars we implemented, because that would soon degenerate into a tedious enumeration of symbols and rules. Instead, we try to convey the basic ideas and principles behind them. Readers who are interested in the details or would like to make their own experiments are invited to download TREEBAG from the internet at http://www.informatik.uni-bremen.de/~drewes/treebag. The distribution contains all examples presented in this paper (and many more from other areas).

The paper is organised as follows. In Section 2, we develop a first grammatical description of plaitwork, which is the basic form of knotwork, in terms of table-driven collage grammars. Sections 3 and 4 build upon this in order to generate square knots (designs with breaks in the interlacing) resp. plaits in the so-called carpet-page design. In Section 5, we turn to rectangular knots. Section 6 is devoted to knots composed of triangular primitives. In Section 7, variations on drawing structurally identical knots are discussed. Finally, Section 8 contains some concluding remarks.

2 Plaits

Celtic knotwork is based on plaiting, where the strands are interwoven so that they turn only at the border of a design. A square plait is shown in Figure 2. Being square, it can be quartered by a horizontal and a vertical line through its centre, see Figure 3, and the quarters can be obtained from each other by a rotation about the centre of the whole. This observation offers already a method

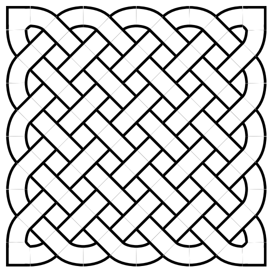


Fig. 2. A square plait

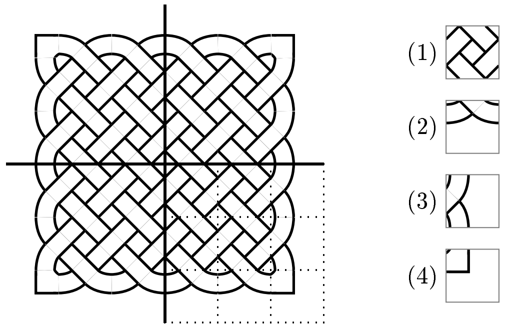


Fig. 3. Dividing the plait into tiles

to construct a description for the whole plait from a description of one of its quarters. Now consider the lower right quarter. The dotted lines in Figure 3 indicate that it is made up of four basic designs shown at the right: four copies of design (1) are used in the inner part, designs (2) and (3) yield the edges, and design (4) the corner. As a tiling of the plane can be obtained by seamlessly repeating design (1), we will from now on call a basic design a tile.

Turning the quarter clockwise by 45 degrees as in Figure 4 makes it easy to see that the tree to its right describes its structure. For this, the symbols occurring in the tree are considered as operation symbols, which turns the tree into an expression that can be evaluated. In order to produce the quarter, the

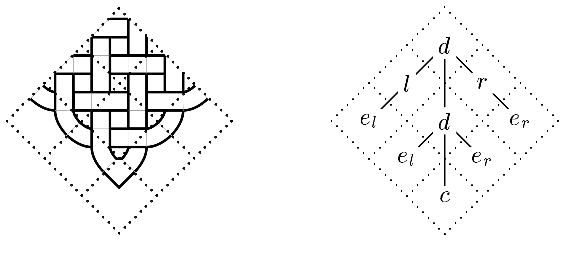


Fig. 4. Constructing a tree for the plait quarter

following meaning is given to the symbols. The nullary symbols e_l, e_r, and c stand for the tiles (2), (3), and (4). Their evaluation simply draws the respective tile (with the upper corner placed at the origin, say). The ternary operation d stands for

(a) drawing tile (1),
(b) shifting the design of the first argument to the left square below tile (1) and, respectively, the third argument down to the right square, and
(c) shifting the second argument to the square directly below tile (1).

Similarly, the unary operation l (resp. r) denotes shifting the design of its argument to the lower left (resp. right) square, and drawing tile (1). Plugging four copies of the tree into a 4-ary operation *initial* which denotes, as discussed above, rotating the design of its ith argument by $(i-1) \cdot 90$ degrees yields a tree which denotes the whole plait of Figure 2.

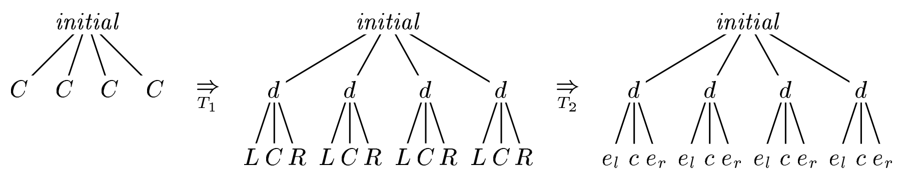
Formally, the collection of the operation symbols $\mathit{initial}, d, l, r$ of respective arities 4, 3, 1, 1 and the constants e_r, e_l, c (which are simply operation symbols of arity 0) is a signature, the tree denoting our plait is a term over that signature (which can be denoted as $d[l[e_l], d[e_l, c, e_r], r[e_r]]$ in linear notation), and the interpretation of the symbols yields an algebra of pictures containing, among others, the desired square plaits. Note that there are many trees over these symbols which denote inconsistent designs. Clearly, a tree denoting a consistent square plait must at least be balanced, as is the case with the tree denoting the plait of Figure 2.

Until now, we have discussed a single tree whose evaluation yields a particular plait. But how can we describe *the set of all square plaits* in a formal way? With the signature and the algebra given as above, we only need a way to generate suitable trees over that signature. This can be done by means of a grammar which has three nonterminal symbols C, L, R, the axiom $\mathit{initial}[C, C, C, C]$, and the following six rules:

$$\begin{array}{ll} C \to d[L, C, R], & C \to c, \\ L \to l[L], & L \to e_l, \\ R \to r[R], & R \to e_r. \end{array}$$

Such a rule is applied by plugging the root of the right-hand side tree into the place of the replaced nonterminal. A derivation starts with the axiom and proceeds in a nondeterministic way until there is no nonterminal symbol left. For convenience (and since it does not affect the resulting trees), we shall always consider maximum parallel derivations, i.e., in each step all nonterminals are replaced in parallel.

Using the rules from above, some of the generated trees are balanced, and thus denote square plaits, whereas others are not. Unbalanced trees are generated by derivations in which the terminating rules are applied too early resp. too late in some places. In order to exclude such derivations, we have to regulate the generation process in an appropriate way. For this, the notion of a table-driven,



initial
C C C C
$\overset{\Rightarrow}{T_1}$
initial
d d d d
L C R L C R L C R L C R
$\overset{\Rightarrow}{T_2}$
initial
d d d d
e_l c e_r e_l c e_r e_l c e_r e_l c e_r

Fig. 5. A derivation with the tables T_1 and T_2

or TOL, grammar [Roz73,PL90,KRS97] (see also [DKK] for the case of collage grammars) can be put to use. In such a grammar, there may be more than one set, called table, of rules (with every table containing at least one rule for each nonterminal), and a derivation step from a tree t to a tree t' consists of choosing one table, and rewriting in parallel every nonterminal in t with an appropriate rule in that table. As an example, we may use two tables T_1 and T_2, where T_1 contains the three nonterminating rules on the left, and T_2 contains the three terminating ones on the right. A derivation using these two tables is shown in Figure 5. Every application of T_1 adds one level to the tree, whereas T_2 finishes the derivation as it contains only terminating rules. The interpretation of the tree generated by this derivation results in the plait of Figure 6.

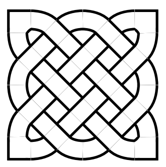


Fig. 6. Interpreting the tree generated in Figure 5

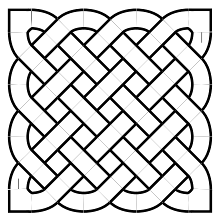
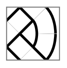
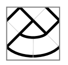
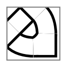


Fig. 7. The square plait of intermediate size

Note that there is a square plait, shown in Figure 7, whose size lies between those of Figures 2 and 6. It can be produced by adding three constants e'_r, e'_l, c' to the signature, which are interpreted as the tiles

(5)  (6)  (7) 

respectively, and a table T_3 analogous to T_2 to the grammar. Tiles (5) and (6) will reappear in Section 3, where they are used to generate knots rather than plaits.

When the grammars get more complicated than the one presented above (and they soon will!), it is usually too tedious to discuss their rules and the operations of their associated algebras separately and in all detail. In fact, once the basic idea of the construction is understood, it tends to be a straightforward (though sometimes time-consuming) programming exercise to devise appropriate rules and define the required operations. Therefore, we will rather explain the overall behaviour of a typical derivation than discuss every single rule and each operation used. Furthermore, while the distinction between trees and their evaluation is a very useful principle from a conceptual point of view, it is normally easier to understand the idea behind a particular grammar when it is presented in terms of pictures. Fortunately, there is an elegant way of doing so: We just have to extend the considered algebra *so that it interprets nonterminal symbols as primitive pictures* (special tiles, for instance). As an immediate result, the nonterminal trees occurring in a derivation can be evaluated as well, yielding a pictorial representation of the derivation.

If we use this idea to visualise the derivation of the plait in Figure 2, we get a sequence of pictures as shown in Figure 8 if nonterminals are interpreted as grey

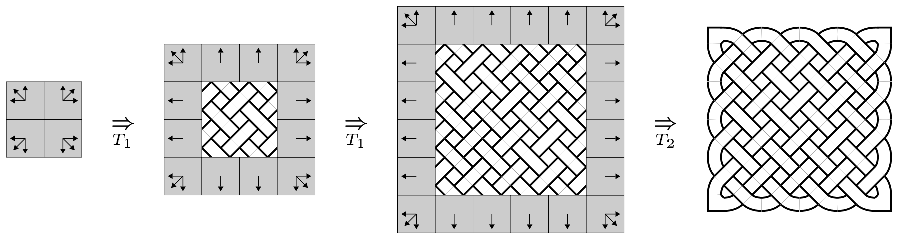


Fig. 8. Deriving the plait in Figure 2

squares with arrows inside, indicating the direction into which a nonterminal is propagated by the rules. Note that the information provided by this interpretation of nonterminals is not completely sufficient to reconstruct the grammar. In particular, it does not reveal that L and R are two distinct nonterminals, as the interpretation of one of them is simply a 90-degree rotation of the other. One could, of course, change the interpretation of L or R in order to make the difference visible. However, in the present case L and R behave more or less alike, whereas the behaviour of C is different, and this is reflected by the chosen interpretation.

3 Square knots

In a knot, strands turn not only at the border but also in between, at so-called breaklines. The traditional construction of a knot usually starts with a plait (at least conceptually). Afterwards, one repeatedly chooses a crossing of strands s and s' and cuts both strands, yielding loose ends s_1 and s_2 respectively s'_1 and s'_2. Then, s_1 is connected with s'_1 and s_2 is connected with s'_2, as illustrated in Figure 9. As a result, the crossing has disappeared and two new turns have emerged.

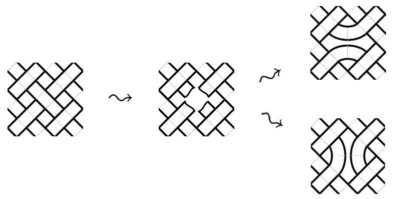
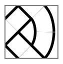
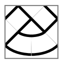
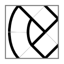
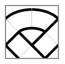
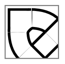


Fig. 9. Introducing breaks by cutting and reconnecting strands

Let us use this method in order to turn the square plaits of Section 2 into knots. In doing so, we need five additional tiles, which we obtain by modifying tile (1):

(1a)  (1b)  (1c)  (1d)  (1e) 

Note that (1a)=(5), (1b)=(6), and tiles (1c), (1d), (1e) can be obtained from tiles (5), (6), (7), respectively, by a 180 degree rotation (but *not* by a reflection).

If we want to use these tiles in order to break some of the crossings in a plait like the one shown in Figure 2, we have to face two major difficulties. First, in order to obtain a consistent design it must be ensured that, for instance, whenever tile (1c) is used in some place, it is accompanied by tile (1a) on its left. Second, celtic knots are usually symmetric with respect to the placement of breaks, which we therefore wish to ensure as well. Again, tables turn out to be a useful mechanism for dealing with both requirements.

In order to add breaks in a systematic way, we increase the number of tables. The tables T_1 and T_2 introduced in Section 2 remain useful and play the same rôle as before. A few rules must be added to these tables in order to deal with new nonterminals, but these rules can be inferred from the existing ones in a straightforward way. Besides the old ones, there are two new tables T_{a} and T_{b}. Intuitively, an application of table T_{a} 'informs' the four nonterminals C in the corners of the square that they and their descending chains of L's and R's are *allowed* to insert breaks. This is implemented by turning C into a new nonterminal C_{b}. If C_{b} is replaced in the next step, it produces an L_{b}, C, and R_{b} instead of L, C, and R. In this way, it propagates the information that breaks are allowed downwards to its outermost L- and R-descendants. The table T_{b} is the one which actually inserts breaks. While its rules treat L and R in the same way as the first table, it inserts tiles (1c) and (1d) when L_{b} resp. R_{b} are replaced. Similarly, T_{b} uses tile (1e) when replacing C_{b}.

However, we still have to ensure that the tiles (1a) and (1b) always occur next to (1c) and (1d), respectively.[1] We implement this by a slight change of the original production for C. Instead of generating an L and an R when replacing a C, we now generate new nonterminals L_1 and R_1 in order to keep track of the fact that they lie next to a C. Table T_{a}, which turns C into C_{b}, also turns L_1

[1] Actually, this problem occurs only if the breaks happen to lie *inside* a quarter. Otherwise, the requirement is automatically satisfied since the quarters are rotations of each other.

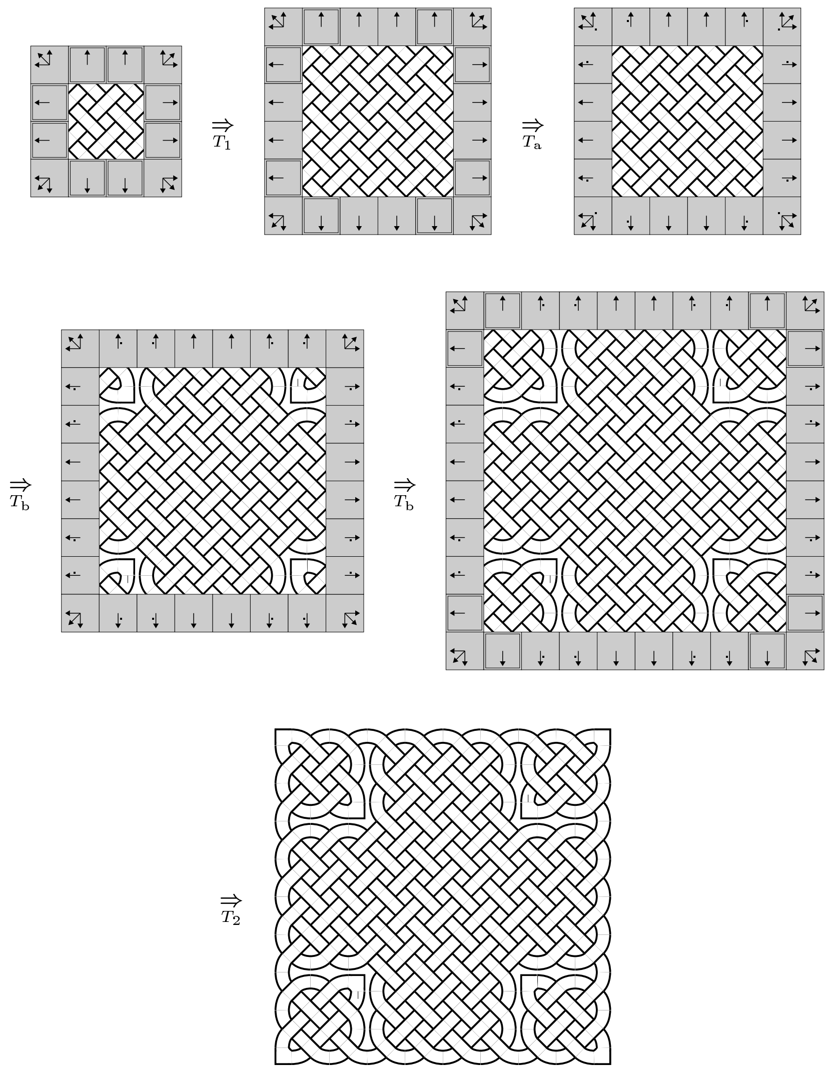


Fig. 10. Deriving a knot

into $\overline{L_b}$ and R_1 into $\overline{R_b}$. These nonterminals act like L_b and R_b, except that the opposite tiles (1a) and (1b) are used by the respective rules in T_b.

If this sounds complicated, have a look at the pictorial representation of a sample derivation in Figure 10 (where the first picture, which equals the first one in Figure 8, is left out). The nonterminal symbols C, L, and R are depicted as in Figure 8, L_1 and R_1 are indicated by an additional frame, and the sides at which $L_b, \overline{L_b}, R_b, \overline{R_b}$ produce breaks are indicated by small black squares. Likewise, C_b is distinguished from C by the addition of such a black square.

Some of the square knots generated by the discussed grammar are depicted in Figure 11. A careful look at the pictures reveals a necessary restriction that has

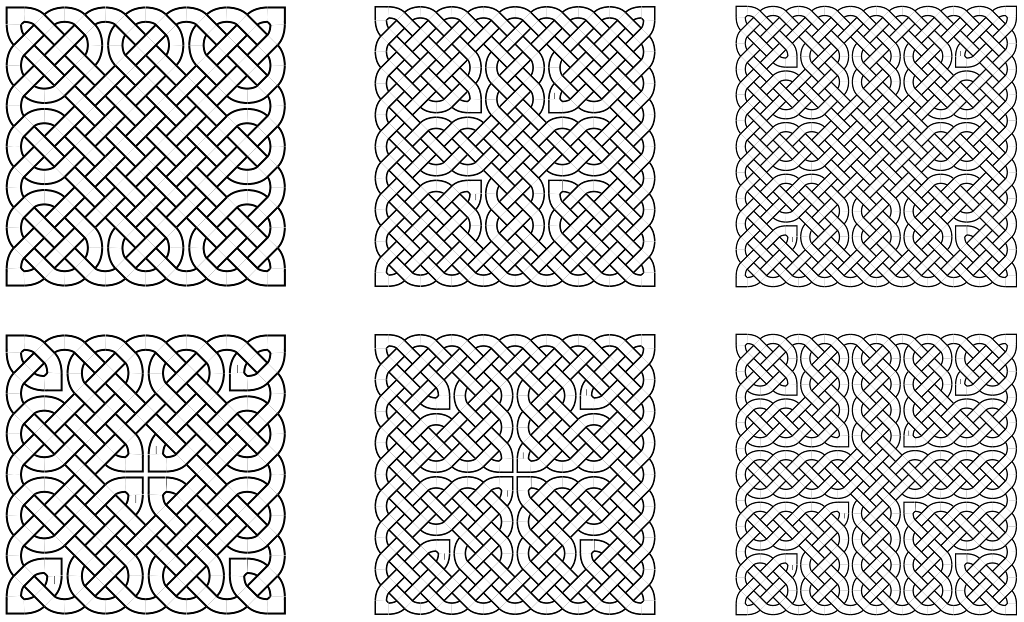


Fig. 11. Some derivable square knots

not been mentioned so far. Whenever the table T_a has been used, we have to wait for at least two steps until T_a can be used again. Otherwise, inconsistent knots would occur (as long as we do not add another type of tiles). Thus, the sequence of tables must not be an arbitrary one—it should be taken from a particular regular language. Another, somewhat clumsy solution would be to use further nonterminals in order to memorize the fact that T_a has just been applied, so that another application of T_a can be skipped if it happens to occur too early.

4 Elements of the carpet-page design

The basic design of square plaits can be varied by interrupting the plaitwork with regularly placed holes. The celtic artist would use these holes for further decoration, creating illustrations in the so-called carpet-page design. The simplest form consists of a square hole in the middle of a square plait, resulting in a closed plaitwork border such as the one shown in Figure 12(a). Placing square holes or L-shaped holes (called L-holes in the following) at the four corners of a square plait yields crosslets (Figure 12(b)), and a cross panel is obtained if these holes are inside rather than on the boundary of the plait (Figure 12(c)).

Cross panels with L-holes may be perceived as variations of square plaits as follows. Recall from Section 2, Figures 3 and 4 that square plaits can be divided into four quarters. Each quarter is made up of n inner rows, resulting from n applications of table T_1, and one edge row as the result of once applying table T_2, see the schema in Figure 13 where the edge row is represented in light grey. Analogously, the schema of Figure 14 illustrates the structure of a cross panel

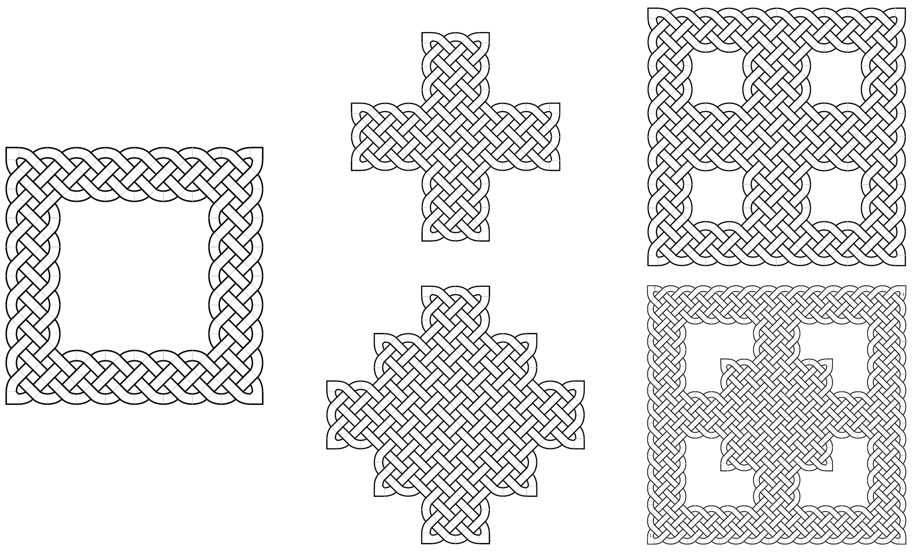
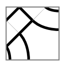


Fig. 12. Square plaits with holes: (a) square border (b) crosslets (c) cross panels

with L-holes, where the dark grey refers to the hole and suitably extended tables T_1 and T_2 are used. As for plaits, the first i rows of the panel are created by i applications of T_1. Then one application of a table T_{m} marks the position of the current corner nonterminal C as the intersection point of the lines going through the left and right edges of the future hole by replacing it with $d[L_{\mathrm{m}}, C_{\mathrm{h}}, R_{\mathrm{m}}]$. During the following j applications of T_1, the outermost nonterminal descendants of that C are L_{m} and R_{m}, and for the part in between, nonterminals $C_{\mathrm{h}}, L_{\mathrm{h}}, R_{\mathrm{h}}$ are used. Now one application of a table T_{u} forms the upper edge of the hole by putting at the positions of nonterminals L_{m}, L_{h}, C_{h}, R_{h}, R_{m}, respectively, tiles (8), (2), (4), (3), (8), where tile (8) is

(8)  ,

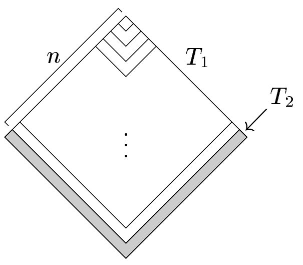


Fig. 13. Schema of a quarter plait

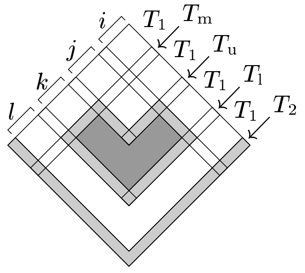


Fig. 14. Schema of a quarter cross panel with L-holes

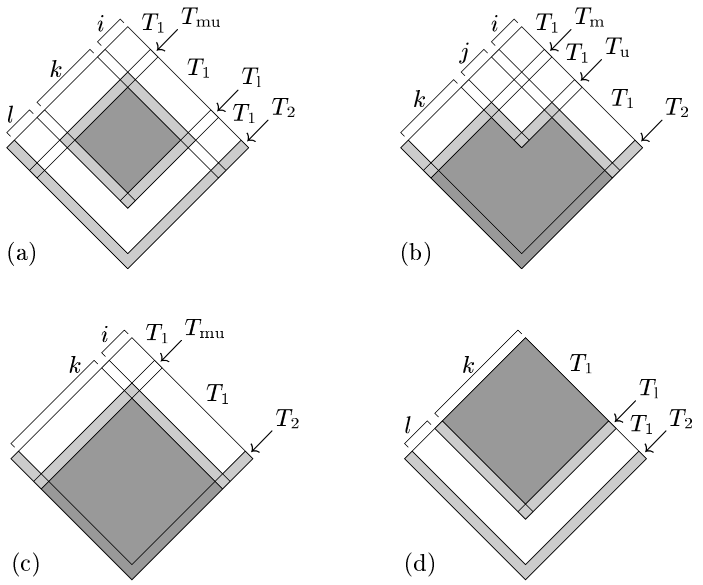
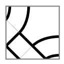
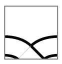
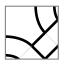
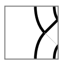
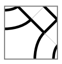


Fig. 15. Schemata of design quarters: (a) cross panel with square holes, (b) crosslet with L-'holes', (c) crosslet with square 'holes', (d) border

and using descendants L_{e}, $L_{\diamond}$, $(L_{\diamond}, C_{\diamond}, R_{\diamond})$, $R_{\diamond}$, R_{e} which will, during the following k applications of T_1, create the left edge (nonterminal L_{e} and tile (3)), the right edge (nonterminal R_{e} and tile (2)), and the blank part (nonterminals $L_{\diamond}$, $C_{\diamond}$, $R_{\diamond}$ and the empty tile (0)) of the hole. The lower edge is provided by one application of a table T_{l} which uses for the nonterminals L_{e}, $L_{\diamond}$, $C_{\diamond}$, $R_{\diamond}$, R_{e} tiles

(9)  (10)  (11)  (12)  (13) 

and descendants L_{m}, L_{h}, $(L_{\mathrm{h}}, C_{\mathrm{h}}, R_{\mathrm{h}})$, R_{h}, R_{m}, respectively. Subsequent l applications of T_1 and one application of T_2 then finish the whole cross panel.

As illustrated in Figure 15, the construction of the other designs can be seen as variants of that for cross panels with L-holes:

(a) with $j = 0$ and tables T_{m} and T_{u} amalgamated into a table T_{mu} which marks the corner nonterminal and immediately starts the upper edge of the hole, a cross panel with square holes is obtained;
(b) letting $l = 0$ and omitting T_{l}, a crosslet with L-'holes' is generated (where T_2 uses the corner tile (4) for the edge nonterminals L_{e} and R_{e});
(c) combining the two variations above leads to a crosslet with square 'holes'; and
(d) a plaitwork border is the result if the unique nonterminal of the first row is already $C_{\diamond}$, i.e., $i + j = 0$ and tables T_{m}, T_{u} are omitted.

While the language of all square borders, crosslets, and cross panels in the forms just discussed may be interesting in itself, the panels shown in Figure 16 illustrate the appealing effects of repeating the hole-generating process in one

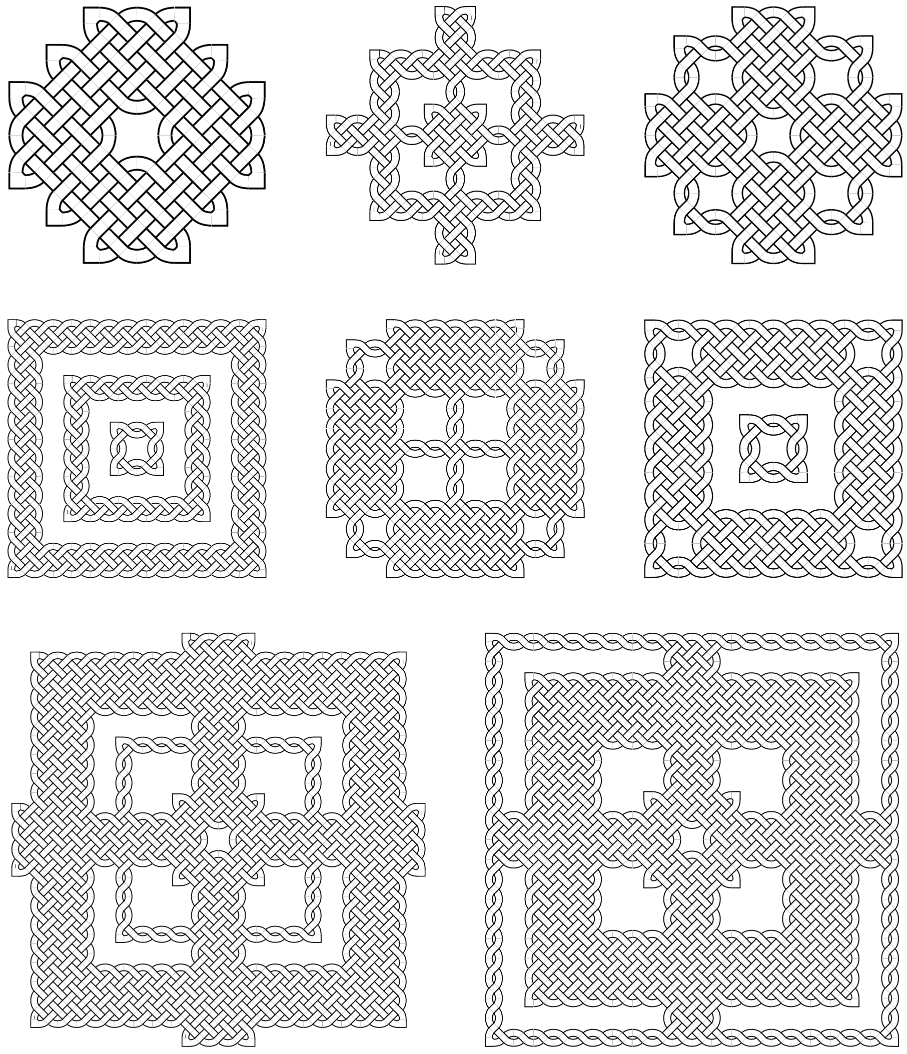


Fig. 16. Panels in the carpet-page design

design. For such an iteration, one can either re-use the left and right edge lines of the preceding hole, or insert a new intersection point after the preceding hole has been completed. An example of a table sequence is $\ldots T_{\mathrm{m}} T_{\mathrm{u}} T_{\mathrm{l}} T_{\mathrm{u}} T_{\mathrm{l}} \ldots$ for the former choice and $\ldots T_{\mathrm{mu}} T_{\mathrm{l}} T_{\mathrm{m}} T_{\mathrm{u}} \ldots$ for the latter, where in addition table T_{l} may occur arbitrarily often and at any position. This leads to a further observation: as for the knots of the previous section, admissible table sequences, i.e., sequences which generate consistent designs, are elements of a certain regular language.

Note, moreover, that the tables of our grammar can be extended so that the intersection point of a second hole may be positioned before the outer edge of the first hole is generated, which can produce, among others, the effects sketched in Figure 17. By iterating that extension and accepting the resulting explosion in the number of nonterminals, rules, and tables, it is clearly possible to handle

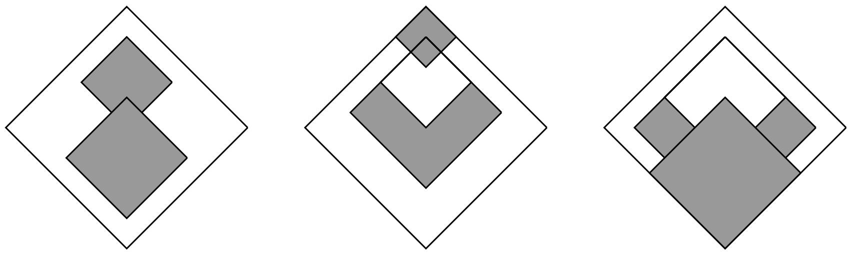


Fig. 17. Placing the intersection point of the second hole before the end of the first

up to n intersection points within the area of one hole, where n is some finite natural number. However, we do not know whether there is a grammar which can achieve an arbitrary number of intersection points in the area of one hole.

Let us mention one further, and simpler, modification of our original grammar, which was used e.g. for the panel in the centre of the upper row in Figure 16. Dividing this design into tiles reveals that its outermost edge is fashioned from rotations of tile (5) instead of (2), which implies for the general case that one must choose for the remaining parts of edges tile (1) instead of tiles (8), (9), (11), (13), and tile (7) instead of tile (4). The only restriction which has to be observed here is that all tiles in one continuous edge—which may go over several corners—must be of either the one or the other type, an information that can be stored in the nonterminals while an edge is being constructed.

5 Rectangular knots

So far, only square knots have been discussed. Let us now turn to the more general case of rectangular knots. In order to generate such knotwork, one could in fact use the operations of the previous sections together with a more sophisticated grammar, but it seems more instructive to consider another sort of tiles.

In Figure 18, an alternative division of the interior of a plait into tiles is shown. Two basic types of tiles occur, one of them being a 90-degree rotation of the other. We shall use them in order to generate the desired rectangular plaits (and, afterwards, knots). The basic idea is to split the generation into two phases each of which is implemented by a table. In the first phase, the width of the plait is determined: the plait is extended horizontally to either side by one tile in each

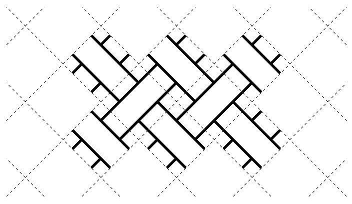


Fig. 18. Alternative segmentation of a plait

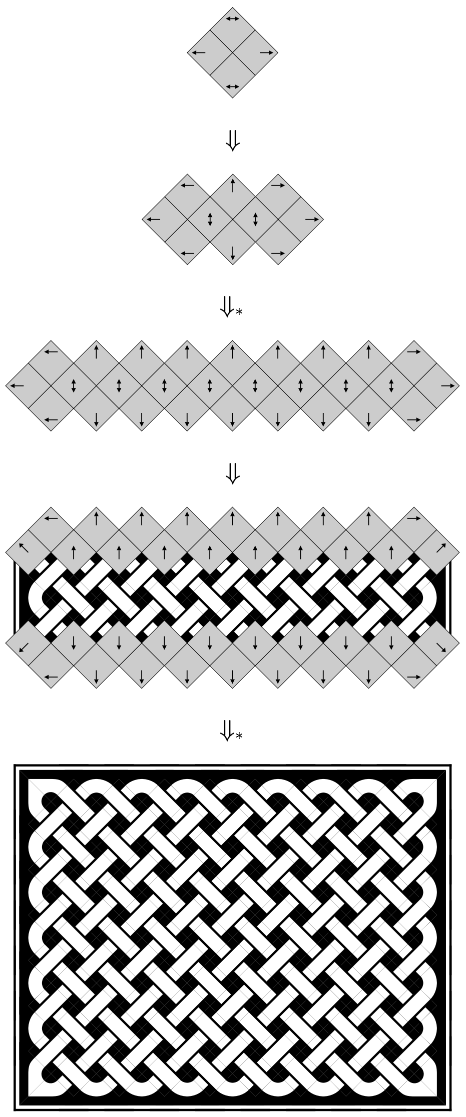


Fig. 19. Derivation of a rectangular plait (as usual, a starred arrow denotes a sequence of derivation steps)

step. The second phase yields the vertical extension. Thus, the width (height) is determined by the number of applications of the first table (respectively second table), which yields rectangular plaits with arbitray width/height ratios. Depicting nonterminals as grey squares with inscribed arrows of various descriptions, Figure 19 depicts a typical derivation (where we have used tiles with a black background and added a frame to the border tiles). Based on this picture, the necessary operations and rules can be defined in a straightforward way. In order to reduce the number of nonterminals and rules as far as possible, it is useful to

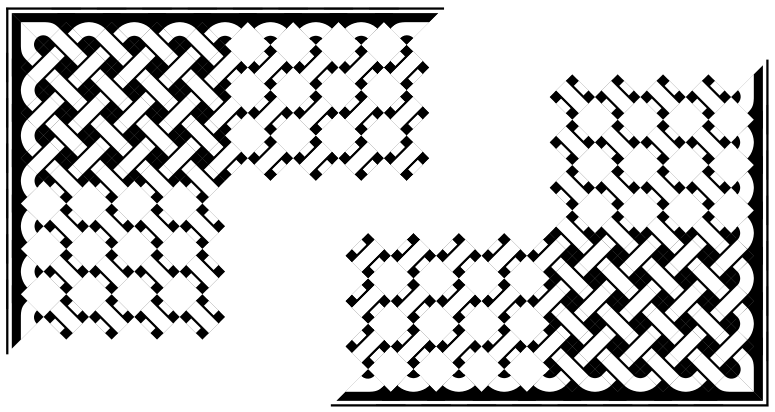


Fig. 20. The two halves of a rectangular plait

observe that the pictures (terminal as well as nonterminal ones) can be split into two parts one of which is a 180-degree rotation of the other, as is illustrated in Figure 20. As a consequence, the plaits can be generated using 9 nonterminals. A trickier approach might perhaps reduce this number even further. As mentioned in the introduction, the reader is invited to fetch the grammar and the Treebag system from the web and make their own experiments. One could, for instance, use two further tables in order to make the plaits grow either to the left, right, top, or bottom in each step (which, however, seems to require numerous additional nonterminals). In this way, one could also generate rectangular plaits with an even number of rows and/or columns (which can, of course, also be achieved with the current grammar if we allow for the use of several axioms).

As any crossing can be broken just by replacing the corresponding tile, the new set of tiles lends itself to the generation of knots. Obviously, consistency is always preserved. Thus, no additional consistency constraints must be observed—the plait grammar can be turned into one that generates rectangular knots such as the one shown in Figure 21 by inserting breaks at random. The required modifications of the grammar are almost trivial. It suffices to make two copies of each rule whose right-hand side contains a crossing, and to replace the crossing

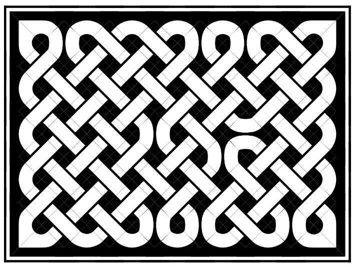


Fig. 21. A rectangular knot obtained by inserting breaks at random

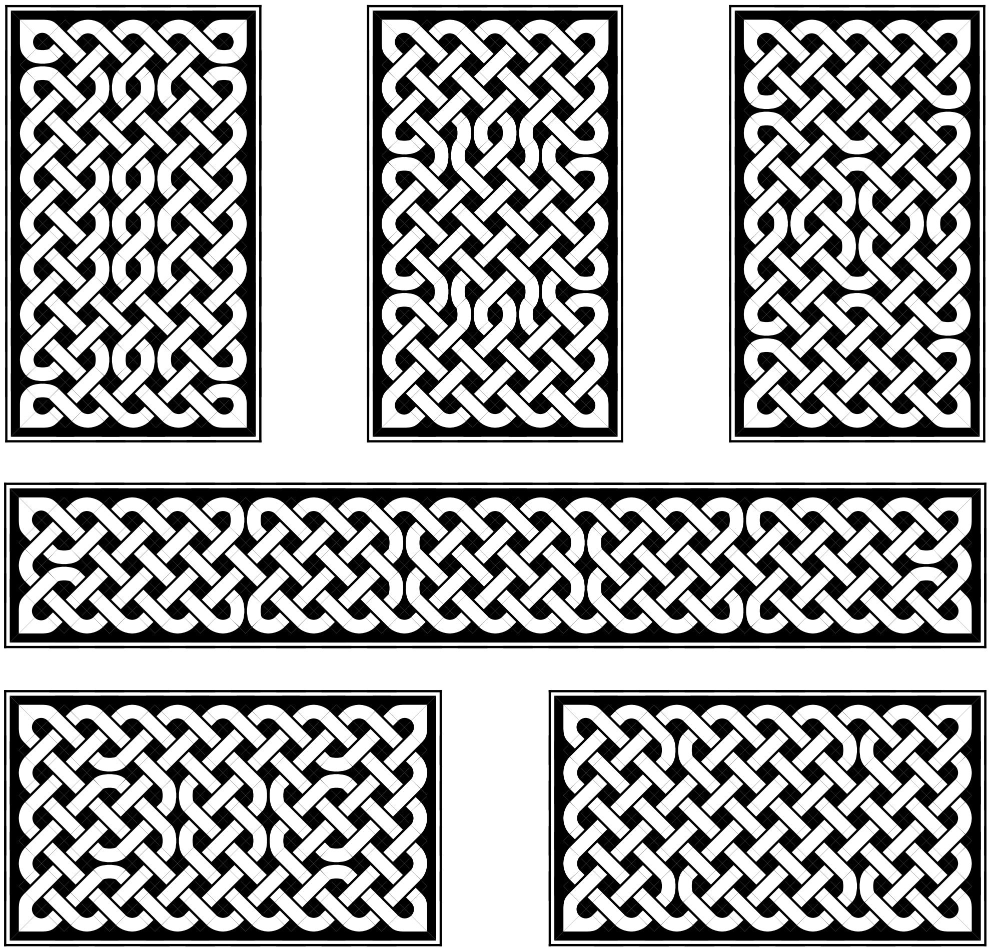


Fig. 22. Symmetric rectangular knots

by a horizontal resp. vertical break. The disadvantage of this method is that symmetric knots are encountered only by chance. Aiming at symmetry, we have to search for a more controlled approach.

A rather ambitious task is to develop a device that generates *all* knots which can be obtained from a rectangular plait by replacing some of the crossings with horizontal or vertical breaks in such a way that the pattern of breaks is symmetric with respect to both middle axes. Some knots of this type are pictured in Figure 22. To generate the set of all these knots, tables do not seem to be powerful enough.[2] This is because tables can only provide a global control while, here, the structure of the dependencies is more complicated. A possible solution is provided by a technique which was introduced in [Dre00b] (see also the picture generator $\mathcal{G}_{\mathrm{Mosaic}}$ in [Dre00a, Section 6]). It provides a regulation principle

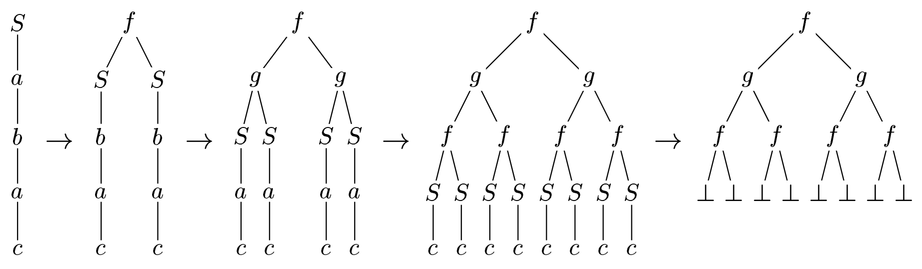
[2] The authors strongly conjecture that this is true in a formal sense, but an exact proof seems to be tremendously difficult and would probably require proof techniques far beyond the known ones (cf. [DKL97,DKK]).

which is perhaps best described as *branching tables*. In order to explain this, an excursion to the theory of tree transducers is necessary.

Suppose we are given a tree-generating grammar whose productions (in linear notation) are $S \to f[S,S]$, $S \to g[S,S]$, and $S \to \perp$. The grammar (with S taken as its axiom) generates all binary trees whose internal nodes are labelled with f or g and whose leaves are labelled with the symbol $\perp$. Now, if we use three separate tables for these productions, we get only those trees in which nodes at equal distance from the root have identical labels. The same effect can be obtained by replacing the grammar with a *tree transformation*, also called a *tree transduction*, as follows. As input trees we use all trees over the unary symbols a, b and the constant c. These are transformed into output trees using rules which are similar to the rules of the grammar, except that their 'nonterminals', called *states* now, are unary symbols placed on top of the input tree, which they consume symbol by symbol. These are the rules:

$$\begin{aligned} S[a[x]] &\to f[S[x], S[x]], \\ S[b[x]] &\to g[S[x], S[x]], \\ S[c] \quad &\to \perp. \end{aligned}$$

Here, x is a variable, i.e., the rules are term rewrite rules in which x stands for an arbitrary tree over the considered input signature. A derivation is carried out by choosing an input tree, placing the state S (which must be the initial one since there is no other) on top of it, and then applying these rules as long as possible. For example, the input tree $a[b[a[c]]]$ gives rise to the (parallel) derivation



producing as its result the tree the grammar generates by applying the first, second, first, and third table (in that order). Intuitively, the symbols in the input tree correspond to the tables. Since the input tree is monadic (i.e., it consists entirely of unary symbols, except for the leaf), in the n-th parallel step all states process copies of the n-th input symbol. In other words, all rules applied in one step belong to the same 'table'. It was shown in [ERS80] that this generation principle is indeed equivalent to the use of tables. In fact, our implementations of examples within TREEBAG use exactly this method in order to produce the effect of tables.

What happens if we allow n-ary symbols (binary ones, say) to occur in the input trees?—We gain the freedom to decide in every step whether the subderivations shall transform copies of the same subtree, or use different ones. Thus, finer control strategies become available. Consider, for example, the same situation as before, except that the arity of input symbol b is now 2 and the corresponding rule is turned into $S[b[x,y]] \to g[S[x],S[y]]$. As a consequence, the device generates all trees over f, g, and $\perp$ such that the two subtrees of each f-labelled node are identical (as before), but the subtrees of a g-labelled node may differ. We could also turn g into a symbol of arity 3 and use the rule $S[b[x,y]] \to g[S[x],S[y],S[x]]$. Then, in the resulting trees every g-labelled node would have identical left and right subtrees whereas the subtree in the middle could be different.

The same technique can be used to generate (the trees denoting) the knots we are aiming at. The input tree contains binary symbols which determine whether to use a crossing or a break in any given place. In the initial picture shown in Figure 19 (which is in fact the second one, being derived from the initial state), the two states represented by the top respectively bottom square use the same control tree, and similarly for the two remaining ones. In the next step, when the topmost square yields the three topmost ones of the second picture, the left and right square are supplied with the same subtree while the one in the middle is controlled by the other subtree. This reflects (and ensures) that the knot development is symmetrical with respect to the vertical axis, but the placement of breaks within the middle column is independent of those to the left and right.

In order to be precise, it should be mentioned that the binary input trees must themselves be generated in a table-driven way to ensure that they are balanced. Thus, altogether, we need two tree transducers—one which transforms monadic input trees into balanced trees of branching tables, and one which uses these as input trees to generate (trees representing) symmetric knots of the form presented in Figure 22.

6 Knots based on triangular tiles

Although square or, more generally, rectangular tiles are suitable to produce a large number of different classes of knots, other types of tiles can be useful as well. In this section, we concentrate on triangular tiles, which are particularly well suited to generate triangular or hexagonal knots.

We discuss two examples of this kind. The first is based on the famous *triquetra* or *trinity knot*, shown in Figure 23(a). Since the trinity knot is a complete knot on its own, it cannot be used very nicely in order to generate larger knots—the latter would simply be sets of individual trinity knots. Therefore, we turn the trinity knot into an 'open' tile as in Figure 23(b). Now, we can assemble arbitrarily large knots of the type shown in Figure 24, using tiles (a), (c), and (d)

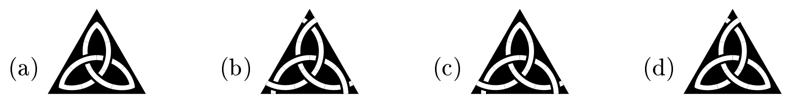


Fig. 23. The trinity knot (a) and three tiles obtained from it (b)–(d)

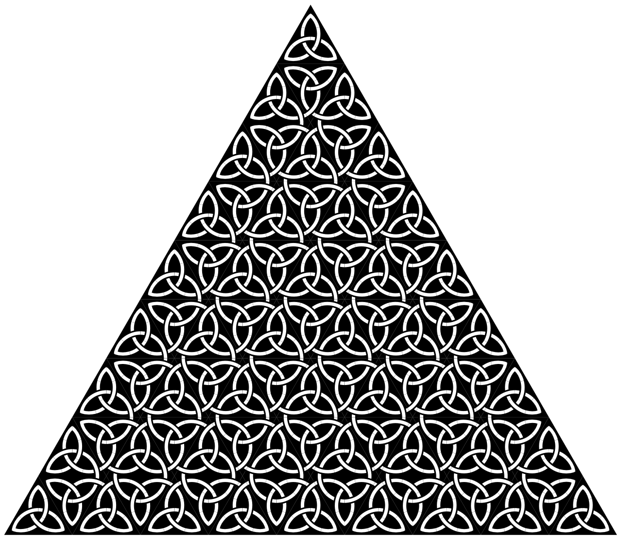


Fig. 24. A triangular knot composed of the tiles in Figure 23

for the boundary. It is quite easy to construct a grammar for this type of knots, using tables and a strategy which generates a knot starting at the top corner and working downwards. Conversely, we can also start in the middle of the base line and grow the knot upwards, as depicted in Figure 25. Intuitively, in this case the nonterminals 'sit' on the left and right edges, and in the upper corner of the

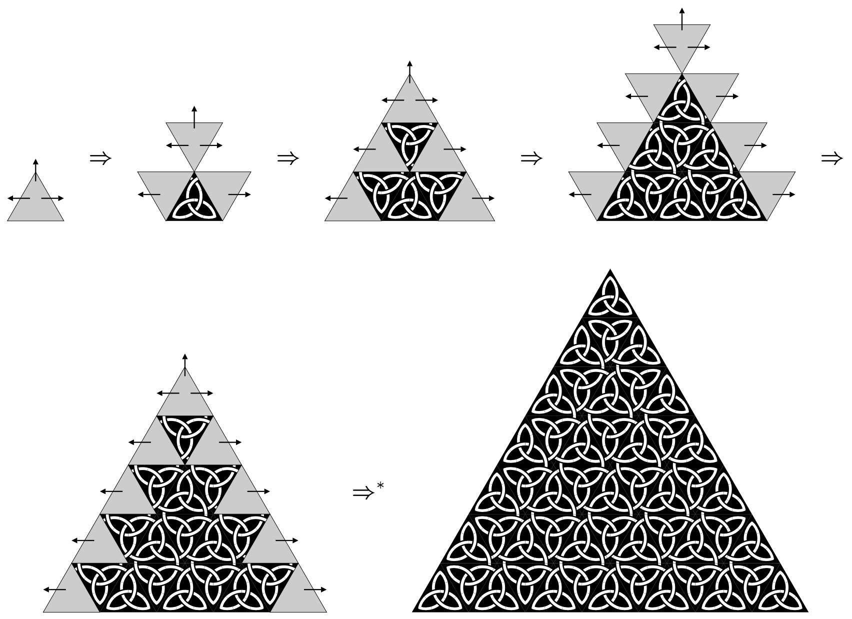


Fig. 25. Derivation of a triangular knot using the tiles in Figure 23

part which has already been generated. In each step, the nonterminals on the edges extend the knot horizontally by one row of tiles. At the same time, the nonterminal in the upper corner propagates upwards and 'emits' a left- and a right-edge nonterminal.

Based on this grammar, more sophisticated variations can be developed. By adding further tables (and nonterminals), one can generate knots with triangular, horizontally centered holes. A derivation of such a knot is depicted in Figure 26. The corresponding grammar has one table which makes the topmost nonterminal switch to a 'hole-generating' state, and one which makes it switch back again.

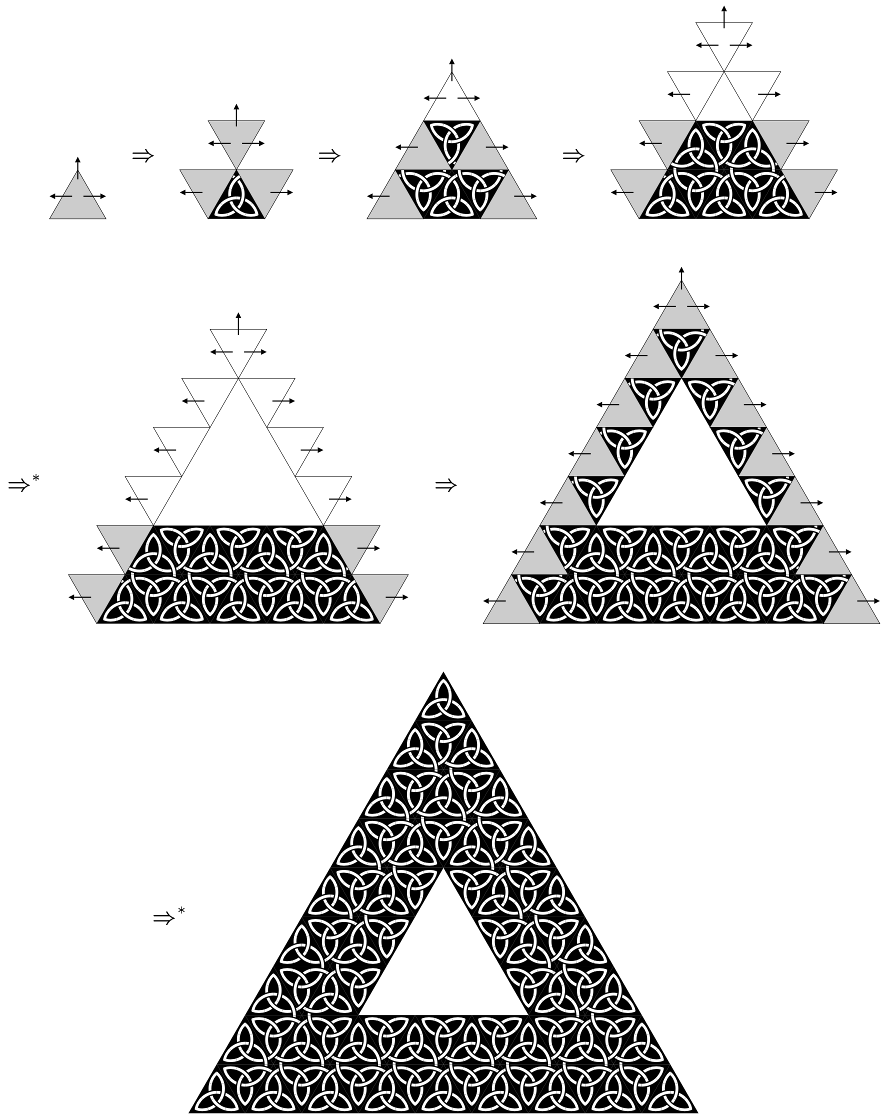


Fig. 26. Derivation of a triangular knot with a hole

Our second example of triangular tiles, shown in Figure 27(a), is based on a motif mentioned in [Dav89, Plate 2] and occurring similarly on the Rosemarkie stone, Ross-shire. Using six rotated copies of this tile, the knot in Figure 27(b) is obtained. We can extend this hexagon to a larger one by adding further layers,

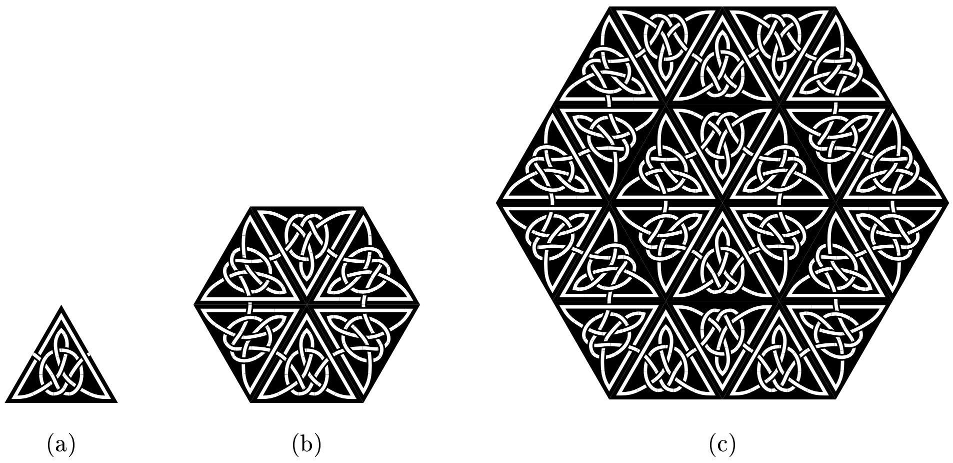
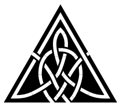


Fig. 27. A second triangular tile (a) and its use to create hexagonal knots (b), (c)

as shown in Figure 27(c). Unfortunately, this yields only knots consisting of separate, non-interwoven layers, which is not that nice. As a remedy, one can modify the tile so that consecutive layers may be connected. Coming from the original tile in Figure 27(a), one of the possibilities is to cut one of the strands twice, and to take the four loose ends down to the basis of the triangle, leading to the following tile:



Using this tile in addition, more interesting knots of any size can be generated. However, how can we make sure that the resulting pattern of tiles is symmetric while keeping as much nondeterminism as possible? Again, branching tables are a possible solution. The generation of a knot starts in the centre of the hexagon, initially using six nonterminals. The purpose of each of them is to generate one of the six triangular parts of the hexagon. The tables are used to ensure that (a) all six triangles are identical and (b) each triangle is symmetric with respect to the bisector of the angle at the centre of the hexagon. Except for these restrictions, the tiles are chosen nondeterministically. One of the resulting knots is presented in Figure 28.

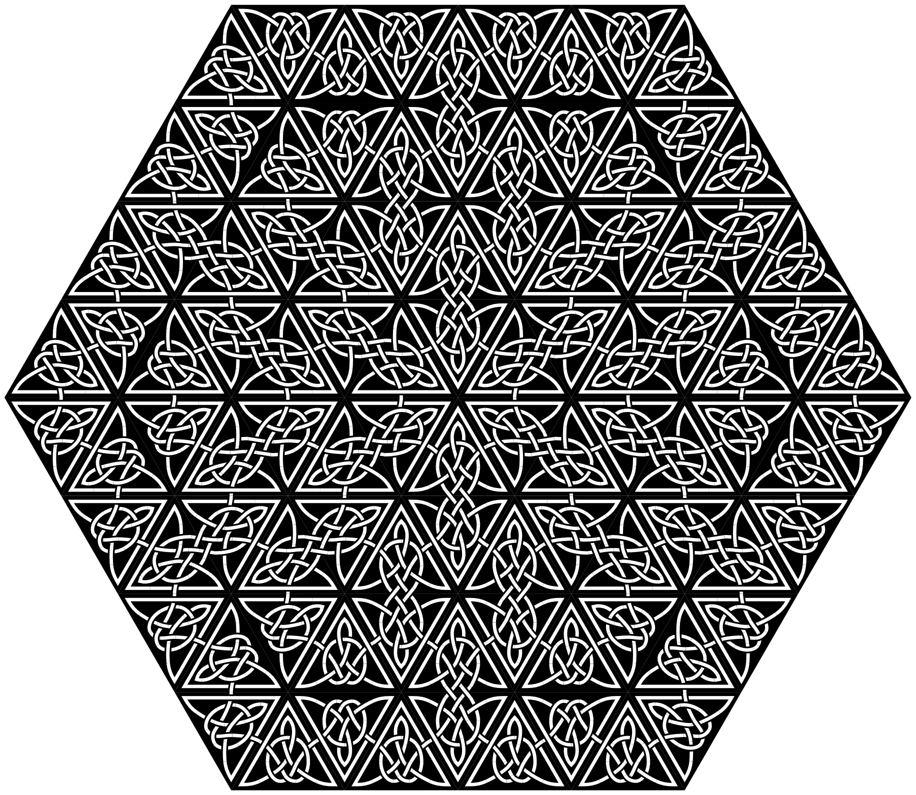


Fig. 28. A knot generated by using binary tables and two types of tiles

7 The final treatment of a knot

In the preceding sections we have seen how, given some type of knotwork, one can try to deduce syntactic rules which allow to describe the structure of the sample knot. At the same time, the interpretation of that structural description was already determined by the sample. In this section, two examples are given to illustrate the flexibility gained by the possibility to choose various algebras for a signature.

When drawing a knot by hand, the particular style of ribbon can be decided upon when the design is already quite advanced. In contrast, the form of the underlying grid has to be fixed in the very first stage of the drawing. Well known for knots are the Pictish proportions 3 by 4, resp. 4 by 3. The knot of Figure 29 implements the second ratio. Its algebra is derived from the algebra in Section 5 by interpreting the symbol at the root of a tree so that the rotation of each argument is followed by a scaling of 4 by 3. Note, however, that unless the ribbon part of each tile is implemented as one line of fixed width, such a scaling causes the ribbon to be broader at the vertical bends. It therefore may be preferable to adjust all operations to the rectangular grid and use the appropriate tiles.

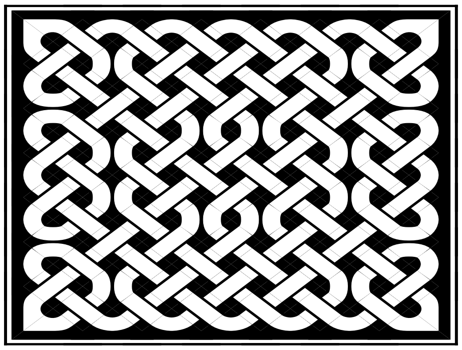
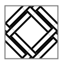


Fig. 29. Knot in 4 by 3 proportion

Now consider again the signature(s) and algebra(s) developed in Sections 2–4. There, each strand of the knot is drawn as one single broad ribbon. A lacier effect can be achieved by splitting that ribbon using the so-called swastika method; the basic inner tile is then

.

With this treatment, the knots of Figure 30 are obtained.

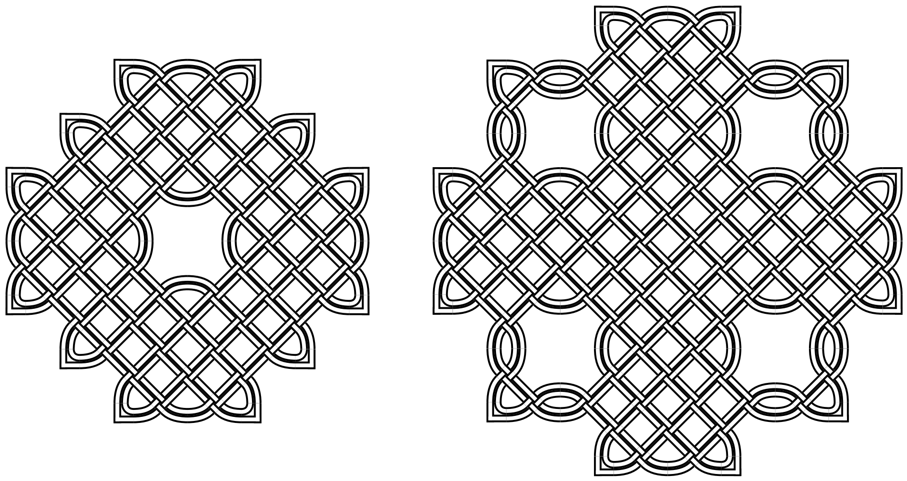


Fig. 30. Knots with split ribbons

Note that iterating the process of splitting the ribbons—whether into two, three, or more ribbons—can be formalised by a collage grammar as well, allowing to generate more and more refined knots.

8 Conclusion

In this paper, we have explored the syntactic structure of celtic knotwork. The construction of various classes of knots can be modelled by table-driven collage grammars with regulated derivations. The formal model has the advantage that the structure of a knot can be separated from its representation, permitting to treat the two individually and independently. Quite intricate structural dependencies can be expressed if the basic regulation provided by tables is enhanced by controlling the admissible sequences of applied tables, such as requiring that they belong to some regular language, or even by arranging the tables in the structure of a tree, as is done with branching tables. Given a structural representation of a knot, its pictorial representation can be changed easily by choosing a different set of tiles. The knots which we have generated are as yet of somewhat basic type, but there are several directions open to further investigation.

Firstly, one may think of using more complex tiles in the generation of knots. For instance, adding a third dimension yields knot models which could practically serve as jewellery designs. Pleasing effects can also be achieved by supplying the knotwork ribbons with colour.

Secondly, there are phenomena not yet provided for in our framework, such as e.g. the wide corner arcs of the knot in Figure 1, or knots with a circular or an irregular contour. Moreover, while we have to some extent integrated the method of regularly placed breaklines to untangle crossing strands, the breaklines occurring in a celtic knot can form rather more complex, but still regular patterns, and it is not yet clear how more involved structural properties of this kind can be expressed.

On the other hand, the theory of formal languages offers sophisticated tools to control the generation of objects. Thinking of the table-driven grammars we use, further refining the regulation techniques for the application of the tables is one possibility. Furthermore, the grammars admit only a top-down generation of trees, i.e., information cannot be propagated bottom-up. More powerful grammars may prove to be helpful e.g. for the generation of knots in the carpet-page design where the holes are not quite so uniform.

A question which naturally arises when designing a knot is the number of strands. In celtic art, the whole knot often consists of one continuous strand; a higher number may be useful e.g. for colouring. A description of a knot in terms of its syntactic structure should allow to compute that number.

The aim of the work reported here was to develop a formal model for celtic knotwork as close to the original as possible. It should nevertheless be noted that such form languages develop in time. Moreover, the methods employed here have proved to be applicable to other visual languages such as fractals or Escher-like pictures, and we believe that combining aspects of these languages may be quite pleasurable, both in results and in the doing.

References

[Bai51] George Bain. *Celtic Art. The Methods of Construction.* Constable, London, 1951.

[Buz] Cari Buziak. Aon celtic art and illumination. `http://www.aon-celtic.com/`.

[Dav89] Courtney Davis. *Celtic Iron-On Transfer Patterns.* Dover Publications, New York, 1989.

[DK99] Frank Drewes and Hans-Jörg Kreowski. Picture generation by collage grammars. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages, and Tools*, chapter 11, pages 397–457. World Scientific, 1999.

[DKK] Frank Drewes, Renate Klempien-Hinrichs, and Hans-Jörg Kreowski. Table-driven and context-sensitive collage languages. In G. Rozenberg and W. Thomas, editors, *Proc. Developments in Language Theory (DLT'99).* World Scientific. To appear.

[DKL97] Frank Drewes, Hans-Jörg Kreowski, and Denis Lapoire. Criteria to disprove context-freeness of collage languages. In B.S. Chlebus and L. Czaja, editors, *Proc. Fundamentals of Computation Theory XI*, volume 1279 of *Lecture Notes in Computer Science*, pages 169–178, 1997.

[Dre00a] Frank Drewes. Tree-based generation of languages of fractals. *Theoretical Computer Science.* To appear.

[Dre00b] Frank Drewes. Tree-based picture generation. *Theoretical Computer Science.* To appear.

[Dre98] Frank Drewes. TREEBAG—a tree-based generator for objects of various types. Report 1/98, Univ. Bremen, 1998.

[ERS80] Joost Engelfriet, Grzegorz Rozenberg, and Giora Slutzki. Tree transducers, L systems, and two-way machines. *Journal of Computer and System Sciences*, 20:150–202, 1980.

[Gla99a] Andrew Glassner. Andrew Glassner's notebook: Celtic knotwork, part 1. *IEEE Computer Graphics and Applications*, 19(5):78–84, 1999.

[Gla99b] Andrew Glassner. Andrew Glassner's notebook: Celtic knotwork, part 2. *IEEE Computer Graphics and Applications*, 19(6):82–86, 1999.

[Gla00] Andrew Glassner. Andrew Glassner's notebook: Celtic knotwork, part 3. *IEEE Computer Graphics and Applications*, 20(1):70–75, 2000.

[HK91] Annegret Habel and Hans-Jörg Kreowski. Collage grammars. In H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. Fourth Intl. Workshop on Graph Grammars and Their Application to Comp. Sci.*, volume 532 of *Lecture Notes in Computer Science*, pages 411–429. Springer, 1991.

[HKT93] Annegret Habel, Hans-Jörg Kreowski, and Stefan Taubenberger. Collages and patterns generated by hyperedge replacement. *Languages of Design*, 1:125–145, 1993.

[KRS97] Lila Kari, Grzegorz Rozenberg, and Arto Salomaa. L systems. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages. Vol. I: Word, Language, Grammar*, chapter 5, pages 253–328. Springer, 1997.

[Mee91] Aidan Meehan. *Knotwork. The Secret Method of the Scribes.* Thames and Hudson, New York, 1991.

[PL90] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants.* Springer-Verlag, New York, 1990.

[Roz73] Grzegorz Rozenberg. T0L systems and languages. *Information and Control*, 23:262–283, 1973.

[Slo95] Andy Sloss. *How to Draw Celtic Knotwork: A Practical Handbook.* Blandford Press, 1995.