Journal of Mathematics and the Arts
Vol. 00, No. 00, Month 200x, 1–20

RESEARCH ARTICLE

On formal descriptions for knitting recursive patterns

Anna Bernasconi, Chiara Bodei and Linda Pagli*
Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo, 3, I-56127, Pisa, Italy.
Tel: +39 050 22 12 735 - Fax: +39 050 22 12 726
(*Received 00 Month 200x; final version received 00 Month 200x*)

We investigate relationships between knitting and formal systems. We show how formal languages, grammars, and algorithms give rise to powerful tools that can be used in both descriptive and prescriptive ways that have not yet been fully explored for designing, then knitting, complex patterns. In particular, we generate knitting meta charts, i.e., abstract schemes, which admit different knitting charts according to the knitting techniques chosen. We thus produce new intricate and recursive patterns that in our opinion also exhibit a sense of artistry and beauty.

Keywords: Knitting Charts; Formal Grammars; Recursive algorithms; Kolmogorov Complexity; Knitting Complexity.

AMS Subject Classification: AMS 2000 Mathematical Subject Classification codes: 68W05, 68Q30, 68Q42.

1 Knitting, Computer Science and Mathematics

We use the term *creative knitting* to refer to the act of designing and executing "beautiful" new patterns for use in knitting. Actually, the knitted objects we show are not finished artworks, but only "swatches" of our patterns. They could become part of a pullover, a scarf, a blanket or of another unconventional object. Our idea is that their aesthetic attraction can contribute to the overall beauty and harmony of a piece of finished artwork, together with the overall design, the quality of the wool and many other variables. In particular, the quality of the chosen yarn in terms of consistency, thickness and softness, transmits other sensations than the pattern *in se*, addressing not only the sense of sight, but also the one of touch.

Creative knitting requires both an artistic sense and some technical knowledge. We believe many aesthetically promising patterns possess great regularity, which can be exploited to compress their description.

In a preliminary study [2] we showed how concepts from the theory of formal languages can be helpful in analyzing knitting. We realized knitting — its rules and its infinite design capabilities — can be formally modeled and studied abstractly.

Here, we consider the problem of generating the knitting charts that are needed to execute complex knitted patterns. We proceed in two stages: (1) the generation of a knitting meta chart, i.e., an abstract pattern scheme for use in knitting; and (2) the selection of the specific knitting techniques for executing the pattern. We therefore separate, temporally and conceptually, the abstract design phase from the concrete realization phase, i.e., "what to do" which does not change, from "how to do" which can change. This allows us to design charts in a hierarchical way following the top-down principle of abstraction: at a high level, a chart just represents a pattern, while at a lower one, the pattern is made of parts that in turn must be obtained by

*Corresponding author. Email: pagli@di.unipi.it

ISSN: 1751-3472 print/ISSN 1751-3480 online

DOI: 10.1080/1751347YYxxxxxxxx
http://www.informaworld.com

selecting suitable stitches or combinations of stitches. Furthermore, our patterns and selection rules can be automatically generated.

To the best of our knowledge, few attempts to link concepts and methods found in Computer Science to knitting have been made. In [9] genetic algorithms were applied to develop new lace patterns. Cellular automata have been applied to knitting; indeed, they allow the design of new patterns starting from only one initial row of stitches and a stitch-decision rule. In her book [8], Debbie New provides many applications using such techniques to produce new and unexpected artistic knitting patterns.

Knitting has also been examined from the point of view of Mathematics. There are many examples of knitted objects that are also elegant mathematical structures such as fractals, Celtic Knots [3], and other complex two-dimensional structures. An exhaustive survey can be found at the web site *The Home of Mathematical Knitting* [17].

For the sake of convenience, we focus here on flat monochromatic knitting, made with a single pair of needles. *Knitting charts* (see Figure 1) are diagrams that provide a compact, graphical representation of stitch patterns. A chart can be viewed as a matrix, where each symbol appearing as an entry represents a stitch, and matrix rows represent rows of stitches. A knitting chart gives the specification, row by row, of the elementary stitches to be knitted and thus represents the *algorithm* to be executed to realize the piece of work.

We will consider several aspects of the creative knitting process, including some which we first introduced in [2]. We begin by using regular grammars [**?**] and collage grammars [5, 10] to describe and generate patterns for knitting charts. Both kinds of grammars use recursive mechanisms which provide simple, elegant and concise representations of pattern generation. Recursion is further exploited by designing algorithms to generate patterns for knitting charts. We obtain some surprising new results, as shown by the swatches of knitted recursive patterns we executed that, in our opinion, have some aesthetic merit. Many of our patterns are also suitable for use in other settings such as quilting and paper collage. Applying the well-known concept of *Kolmogorov Complexity*, we give an argument to show that our recursive knitting patterns have low *"knitting complexity"*, while their standard descriptions have a higher complexity. Kolmogorov complexity was used previously in a study of drawn faces [15], where it was argued that low Kolmogorov complexity captures our instinctive perception of the "beauty" of a face.

The paper is organized as follows. In Section 2, we present our underlying model for knitting. We introduce grammars for the generation of patterns in Section 3. In Section 4 we show how knitting patterns can be obtained through recursive algorithms. In Section 5, from previously generated examples we show how to create new recursive patterns. In Section 6, Kolmogorov Complexity is applied to recursively defined knitting patterns. We then conclude in Section 7 with a discussion of our approach.

2 Our formal model of knitting

2.1 *Stitch patterns*

Stitch patterns determine the characteristics the knitted object will have. They are based on repetitions — the repetition of stitches (giving the horizontal motif) and the repetition of rows (giving the vertical motif) [1].

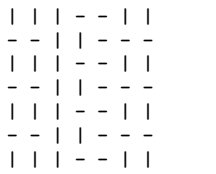
In the literature, stitch patterns are provided both in written form and in chart form. The written description is compressed horizontally by inserting the repetitions needed between two stars, and compressed vertically by indicating the rows to be repeated. In Figure 1 directions for knitting Mistake Stitch Ribbing are given for the first row, since the other rows follow the same schema. To knit row 1 knit 2 stitches, purl 2 stitches, knit 2 stitches, purl 2 stitches, and so on until the end of the row which is then completed by knitting 3 further stitches.

A knitting chart is a matrix, where each element corresponds to a single stitch and every row corresponds to a needle i.e., a row of stitches. Each element is a symbol whose meaning

Cast on over a number of stitches multiple of 4 plus 3.

Row 1: * knit 2, purl 2; repeat from *: to last 3 sts, knit 3.

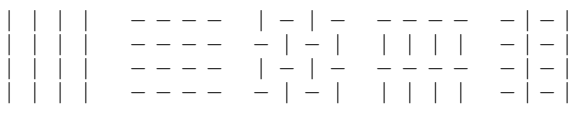
Repeat row 1.



| = Knit on the right side, purl on the wrong side

– = Purl on the right side, knit on the wrong side

Figure 1. *Written description (top) and knitting chart (bottom) of Mistake Rib.*



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

| = Knit on the right side, purl on the wrong side

– = Purl on the right side, knit on the wrong side

Figure 2. (a) *Knitting Charts for Stockinette,* (b) *Reverse Stockinette,* (c) *Seed Stitch,* (d) *Garter Stitch, and* (e) *Ribbing 1x1, respectively.*

is clarified by a suitable key in the chart legend that indicates how to work the corresponding stitch. In order to obtain flat knitting, we must knit back and forth in rows. Knitting charts must therefore be read from the bottom to the top, the odd rows from right to left and the even ones in the opposite direction, i.e., a "boustrophedonic" reading (from ancient Greek $\beta o \upsilon \varsigma$, "ox" + $\sigma \tau \rho \epsilon \phi \epsilon \iota \nu$, "to turn", imitating the ox ploughing the field, back and forth). In knitting, this order comes naturally, since at the end of each needle the side is inverted. Charts represent the pattern of the knitted fabric as we are looking at it, i.e., the right side of the fabric. As a consequence, when knitting the wrong side rows (from left to right) the symbols must be read and executed in complementary way, as recalled by the chart legend. In this way, on the right side, they appear as required and the chart resembles its realization.

The basic knit pattern is called *stockinette*, or stockingnette, commonly found in T-shirt fabric. It is obtained by alternating rows of knit stitches on the *right* side with rows of purl stitches on the *wrong* side (see the corresponding chart in Figure 2 (a)). The visual effect is a grid of smooth V shapes. On the *wrong* side the pattern has a different texture. The effect is a grid of $\sim$ shapes. This *reverse stockinette* (see Figure 2 (b)) is considered a pattern in its own right. Another common pattern is the *seed stitch* pattern (see Figure 2 (c)) that is obtained by alternating knits and purls in a row. The *Garter stitch* (see Figure 2(d)) differs by having knits (or purls) in every row.

2.2 *Knit textures*

The most basic patterns yield *textures* which in turn can be combined to obtain more complex stitch patterns. These textures are used to make: horizontal stripes or welts: e.g., by alternating stripes belonging to different textures, and varying the widths; vertical stripes or ribs: e.g., 1x1 ribbing (see Figure 2 (e)); diagonal stripes: e.g., by making stripes with seed stitch on a garter stitch background; variations on the above themes, such as zigzag stripes; checkerboard patterns. There are also ways of combining textures to obtain patterns in the form of baskets (basket-weave), diamonds, triangles, windmills, and so on, plus all their possible variations.

A Checkerboard pattern, as in Figure 3, is composed of a square-tiled board, where the squares are of two different alternating textures, such as stockinette stitch and reverse stockinette stitch.

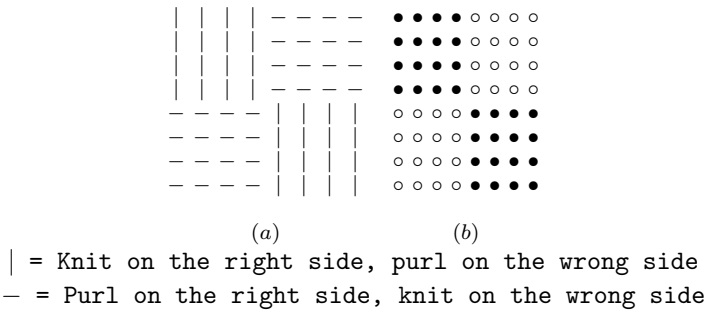


Figure 3. *Checkerboard: knitting chart (a), meta chart (b).*

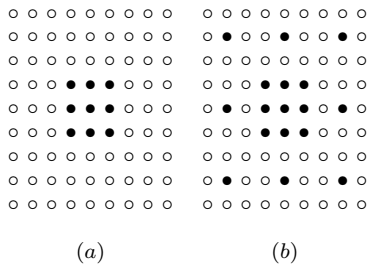


Figure 4. *Sierpinski meta charts of increasing resolution.*

The only variable is the dimension of the squares, i.e., the number of stitches across and the number of rows down. This pattern can be seen as a generalization of *seed stitch*, by considering it to be a Checkerboard where the dimension of the squares is just one.

Whenever a pattern can be partitioned into distinct regions where different textures can be used, we can introduce special symbols for the generic point of each texture, e.g., " • " for the generic point of the stockinette texture, and " ∘ " for the generic point of the reverse stockinette (see the right-hand side of Figure 3). A similar convention is often used in knitting charts. In particular, we are using an abstraction of the texture, where each matrix entry is a suitable basic texture unit. We call these charts *meta charts*. Of course, we can also assign " • " and " ∘ " to other pairs of textures and obtain new pattern combinations. The legend indicates the symbol assignment. In order not to lose generality, henceforth, in our figures with meta charts, we leave the legend unspecified, so that in a second, separate step, the desired textures can be mapped to the corresponding areas of the patterns, often by using suitable transformation rules (see e.g., Figure 10).

Fractals offer further possibilities for pattern design. Of course, as in nature, we can only approximate fractals by displaying the self-similar structure over an extended, but *finite*, scale range. Consider the planar fractal known as the *Sierpinski carpet*. It was first described by Wacław Sierpiński in 1916. The construction of the Sierpinski carpet begins with a square. The original square is divided into 9 congruent squares in a 3-by-3 grid, and the central square is *removed*. The same procedure is then applied recursively to the remaining 8 squares, and this process is repeated until the desired resolution is obtained. To realize such a pattern with our tools, i.e., needles and handwork, we only have to decide how to "remove" a square. This could be done, for example, by using stockinette for the background, and reverse stockinette for the removed squares, or vice-versa. Examples of knitting meta charts induced by the Sierpinski carpet are shown in Figure 4. So far we have only made use of the basic knit and purl stitches. There are other stitches and techniques that can be used to extend the possibilities for creating patterns. These include using slipping stitches, knitting into the stitch below, making a yarn over, or using twisted stitches. Such techniques can be used also to increase or decrease the number of stitches: in this case, the number of stitches in a row may not be constant. (In knitting charts, when the number of stitches in a row changes, a special "no-stitch" symbol is often used to keep the dimension of the rows the same.) Using such methods, we can obtain, for example, "bobbles" (localized sets of stitches forming bumps in relief) that offer us a further

way of decorating fabrics. With such methods, knit fabrics can also include knitted laces that are obtained by combining moves of yarn over and of decreases in such a way that increases and decreases compensate for each other. Finally, the technique of crossing groups of stitches leads to the formation of "cables", opening another rich source of patterns.

3 The grammar of knitting

Within knitting charts, each separate stitch is represented by a special knitting symbol. In other words, there is a finite set of symbols to work with. Explanations for the rows found in written knitting instructions for patterns resemble the production rules in a regular grammar (see Chapter 3 in [11]). In such a grammar, we start with an *initial symbol* S and we apply a set of string rewriting rules, called *productions*, which tell us how to replace previous symbols with new ones. Symbols are further distinguished as being either *non terminal* symbols, i.e., meta symbols to be replaced, or *terminal* symbols that do not need further processing. At each step a non terminal symbol is replaced, by using one of the productions, which may be chosen non-deterministically. The process, or *derivation* is complete when all symbols are terminals. The set of all strings derived from the initial non terminal symbol S forms the language generated by the grammar. In our case knitting symbols will be the terminal symbols.

Definition 3.1: A *regular grammar* is a tuple (T, N, S, P) given by: (i) a finite set of terminal symbols T; (ii) a finite set of nonterminal symbols N; (iii) an initial symbol S in N; (iv) a finite set of production rules P in the form $A \to Bw$ or $A \to w$, where $A, B \in N$ and w is a possibly empty string of terminal symbols in T.

In our knitting framework, each row of a chart is viewed as the result corresponding to a derivation in the grammar. Consider the first row in the diagram for Mistake Rib shown in Figure 1. It can be obtained using the grammar where $N = \{S\}$, $T = \{|, -\}$, and A consists of:

$$\begin{array}{lll} (1) & S & \to \quad S\ --|| \\ (2) & S & \to \quad ||| \end{array}$$

Given S, by directly applying production (2) we obtain the string $|||$; if instead we use production (1) we obtain the string $S\ --||$ that, in turn, can lead to $|||\ --||$, by applying (2), or to $S\ --||\ --||$, by applying (1), and so on. In other words, we can obtain all the strings in the form $|||\ (--\ ||)^i$, with $i \geq 0$, of length $3 + 4i$, where i corresponds to the number of times production (1) is applied. Formally, this grammar generates the following language of regular expressions $|||\ \{--\ ||\}^* = \{|||, |||--||, |||--||--||, ...\}$, where $*$ is the closure operator called Kleene star [11], defined in this case to be $\{--\ ||\}^* = \{(--\ ||)^i : i \geq 0\}$. It is interesting to observe that the written instructions in Figure 1 use the star in the same sense as the Kleene star.

Grammars can also be used in the generation of meta charts. The first row of the Checkerboard meta chart in Figure 3 (c), can be generated from:

$$\begin{array}{lll} (1) & S & \to S\ \circ\ \circ\ \circ\ \circ\ \bullet\ \bullet\ \bullet\ \bullet \\ (2) & S & \to \quad \circ\ \circ\ \circ\ \circ\ \bullet\ \bullet\ \bullet\ \bullet \end{array}$$

We can introduce intermediate production rules, which use additional non terminal symbols, that allow us to obtain the desired dimension of the tiles. For the checkerboard, we use two general rules, valid for all patterns in that family:

$$\begin{array}{lll} (1) & S & \to \quad SBA \\ (2) & S & \to \quad BA \end{array}$$

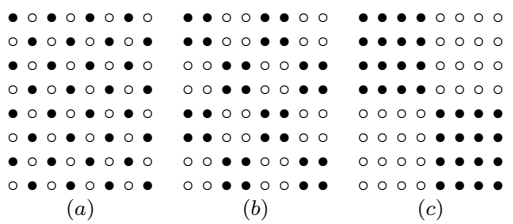


Figure 5. *Checkerboard meta charts with square tiles of increasing dimension.*

and then by adding just two more rules as indicated below we can obtain production rules for the charts (a), (b), and (c) in Figure 5, respectively:

A	$\rightarrow$	$\bullet$	B	$\rightarrow$	$\circ$		seed stitch
A	$\rightarrow$	$\bullet\ \bullet$	B	$\rightarrow$	$\circ\ \circ$		2x2 checkerboard
A	$\rightarrow$	$\bullet\ \bullet\ \bullet\ \bullet$	B	$\rightarrow$	$\circ\ \circ\ \circ\ \circ$		4x4 checkerboard

These examples show why grammars appear to be a good tool for modeling patterns, as they provide simple, elegant and concise representations for pattern generation. Note that the grammars in our framework are recursive: each non terminal symbol can be replaced by a string that includes it.

To this point, we have considered stitch patterns row by row, the way they are read when using their written or chart representations. But a knitting pattern is two-dimensional, and its motifs are more naturally developed by taking into account both dimensions at once. Therefore, we now consider generating the knitting chart as a whole, instead of in single rows. Row by row instructions can be easily obtained from the resulting chart in a second phase. This requires us to generalize our regular linear grammar approach to allow for two-dimensional grammars. There are several kinds of grammars that have been introduced for image generation that can be exploited for this purpose. Following [6], we will define and use a two-dimensional grammar inspired by *collage grammars* [5, 10]. The underlying idea is to augment pieces of patterns with non terminal place holders that can be replaced by other pieces of patterns, through a typical context-free mechanism i.e., using only productions which substitute for a single non terminal at each step in the derivation. In a collage grammar each pattern corresponds to a derivation tree: the derivation tree describes the pattern syntactic structure, while its evaluation yields the actual pattern. We will introduce these grammars through two specific examples.

3.1 *A two-dimensional grammar for Checkerboard patterns*

We are able to generate checkerboard patterns where the basic square tile has any desired dimensions. For simplicity though, we show how to generate the seed stitch (where the dimension of the tile is 1). Increasing the dimension of the tile can be easily obtained by using the same procedure we illustrated above for generating checkerboard by rows with linear grammars.

A seed stitch pattern can be seen as a collage of tiles of two basic kinds: a $\bullet$ (tile 1), or a $\circ$ (tile 2). It can be considered as a fractal, composed of four alternating tiles, that in turn can be decomposed in four further alternating tiles, and so on.

The collage grammar we consider is defined as follows:

- Non Terminal Symbols: $N = \{C, \overline{C}\}$;
- Terminal Symbols: $T = \{\bullet, \circ\}$;
- Initial axiom: $S = d[C, \overline{C}, \overline{C}, C]$;
- Production rules $P = \begin{cases} (1)\ C \rightarrow d[C, \overline{C}, \overline{C}, C] & (2)\ C \rightarrow \bullet \\ (3)\ \overline{C} \rightarrow d[C, \overline{C}, \overline{C}, C] & (4)\ \overline{C} \rightarrow \circ \end{cases}$

The initial axiom gives the starting tile composition. The 4-ary operation d stands for drawing

four tiles, using the four arguments $C, \overline{C}, \overline{C}, C$, according to the following spatial arrangement:

$$\begin{array}{cc} C & \overline{C} \\ \overline{C} & C \end{array}$$

In the derivation trees in Figure 6, the first (respectively, second) element in the last level refers to the tile in the upper left (respectively, right) corner, while the third (respectively, fourth) refers to the tile in the lower left (respectively, right) corner. Similarly in Figure 7. The terminal symbols $\bullet$ and $\circ$ stand for tile 1 and tile 2, respectively. Their evaluation simply draws the corresponding tile. If we want to have a more general checkerboard pattern, we only need to replace the rules (2) and (4), e.g., to obtain a 4×4 checkerboard pattern, rule (2) becomes $C \rightarrow \bullet\ \bullet\ \bullet\ \bullet$ and rule (4) becomes $\overline{C} \rightarrow \circ\ \circ\ \circ\ \circ$.

A derivation starts with the initial axiom and proceeds by replacing a non terminal with the root of the tree described by the right-hand side of the applied rule. We only consider maximum parallel derivations, i.e., we will replace all non terminals in parallel for each step. Furthermore, in order to obtain balanced trees, we regulate derivations in such a way that all the terminating rules (i.e., those that lead to the completion of the derivation), in our case rules (2) and (4), are applied at the same time. To achieve this, we resort to the notion of *table-driven* or *TOL grammars* [7, 14]. In these grammars, rules are organized in sets or tables. In each derivation step, all the non terminal replacements are performed in parallel, according to the rules of a single table. In our case, we can partition our four rules into two tables: T_1 is composed by the odd rules $\{(1), (3)\}$ and T_2 by the even ones $\{(2), (4)\}$. We use the first table to add one level to the tree and the second one to complete the tree derivation. We show some derivations in Figure 6 and in Figure 7, where we indicate with $\overset{\Rightarrow}{T_i}$ the derivation step obtained by applying the productions in table T_i, with $i = 1, 2$.

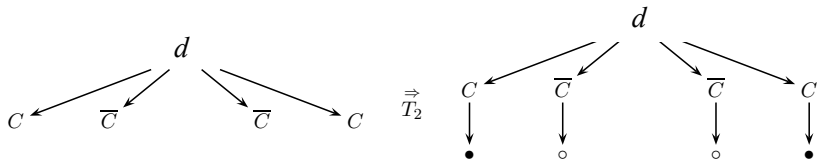


Figure 6. *A derivation tree obtained by applying to the initial axiom the productions in table T_2.*

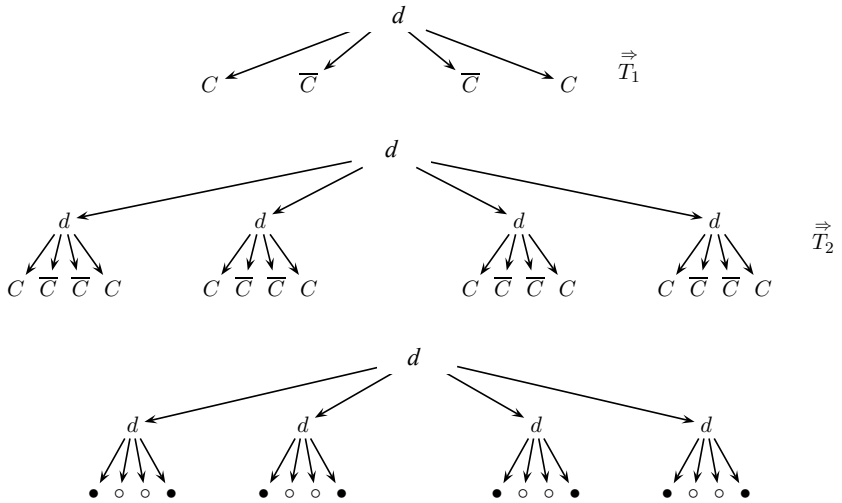


Figure 7. *A derivation tree obtained, by first applying the productions in table T_1 and afterwards by applying the productions in T_2.*

In Figure 6, applying the productions in T_2 amounts to replacing C with $\bullet$ and $\overline{C}$ with $\circ$, consequently concluding the derivation process, since there are no non terminal symbols any longer. Instead, in Figure 7, by first applying the productions in T_1, we substitute $d[C, \overline{C}, \overline{C}, C]$ for C and $d[C, \overline{C}, \overline{C}, C]$ for $\overline{C}$. In the second step, we apply the productions in T_2, as seen above, by completing the derivation process.

3.2 *A two-dimensional grammar for the Sierpinski carpet*

In a similar way, we can describe a pattern based on the Sierpinski carpet.

- Non Terminal Symbols: $N = \{C, \overline{C}\}$;
- Terminal Symbols: $T = \{\bullet, \circ\}$;
- Initial axiom: $S = d[C, C, C, C, \overline{C}, C, C, C, C]$;
- Production rules $P = \begin{cases} (1)\ C \to d[C, C, C, C, \overline{C}, C, C, C, C] & (2)\ C \to \circ \\ (3)\ \overline{C} \to \overline{d}[\overline{C}, \overline{C}, \overline{C}, \overline{C}, \overline{C}, \overline{C}, \overline{C}, \overline{C}, \overline{C}] & (4)\ \overline{C} \to \bullet \end{cases}$

Here, the 9-ary operation d stands for drawing nine squares, using the arguments of the operation, according to the following spatial arrangement of the corresponding elements:

$$\begin{array}{ccc} C & C & C \\ C & \overline{C} & C \\ C & C & C \end{array}$$

The 9-ary operation $\overline{d}$ stands instead for drawing nine squares, using the arguments of the operation, as follows.

$$\begin{array}{ccc} \overline{C} & \overline{C} & \overline{C} \\ \overline{C} & \overline{C} & \overline{C} \\ \overline{C} & \overline{C} & \overline{C} \end{array}$$

Also in this case, we can use the rules in table $T_1 = \{(1), (3)\}$ to add one level to the tree, and the rules in $T_2 = \{(2), (4)\}$ to complete the derivation. For example, to generate the pattern in Figure 4 (b), we have to apply first the rules in T_1 and then those in T_2.

4 Recursive knitting: an algorithmic description

In this section, we continue to exploit the power of recursion to design some new knitting patterns. As with grammar descriptions, one additional advantage of algorithmic descriptions over standard ones is that from one algorithm it is possible to obtain families of new patterns by simply changing the initial conditions and the basic stitches.

We give some examples of families of patterns of apparently increasing *difficulty* that can be recursively generated. As we will see in Section 6, the difficulty is only with respect to a human executor and not intrinsic to the pattern. The first two algorithm examples cover the two patterns introduced in the previous sections: the Checkerboard and the Sierpinski carpet. The next two are knitted versions of the *Binary tree pattern* and the *Butterfly network*.

Without loss of generality, we consider the algorithmic generation of square or rectangular knitting meta charts, each represented as a matrix of texture units. Note that it is difficult for a human who is not familiar with recursive algorithms to execute these patterns. However, they can be automatically transformed into knitting charts or written instructions, by some kind of *knitting compiler* (see [16]). To give a better intuition of the recursive strategy, we provide a high level description of our algorithms. More detailed descriptions of these algorithms can also be found in [2].

4.1 Checkerboard patterns

For the Checkerboard patterns, the inputs are the dimension $n = 2^k$ of the meta chart to generate, and the dimension d of the base case of the pattern. Following the same idea employed for the collage grammars, a $n \times n$ square is divided into four $n/2 \times n/2$ squares, and once the first square has been recursively computed, the overall chart is obtained by compositing four identical copies of it. The recursive calls end when the dimension of the squares equals the dimension d of the base case, at which point the algorithm returns the following square

$$\begin{matrix} K & P \\ P & K \end{matrix}$$

where K and P are two $d/2 \times d/2$ matrices of symbols $\bullet$ and $\circ$, respectively.

Checkerboard(n, d)
  if $(n == d)$ **return** BaseCase(d);
  else
    $C =$ Checkerboard$(n/2, d)$;
    return $\begin{pmatrix} C & C \\ C & C \end{pmatrix}$;

BaseCase(d)
  $K =$ new $(d/2 \times d/2)$ matrix of $\bullet$;
  $P =$ new $(d/2 \times d/2)$ matrix of $\circ$;
  return $\begin{pmatrix} K & P \\ P & K \end{pmatrix}$;

Observe that by varying the value of d from its maximum value of n down to its minimum value of 2, we obtain patterns with progressively increasing resolution.

Of course, for such an elementary pattern an iterative algorithmic description is more natural and less restrictive than the recursive one; moreover it would more easily admit square matrices of arbitrary size, not just powers of two, as well as rectangular domains as follows:

IterativeCheckerboard$(n1, n2, d1, d2)$
  $A =$ new $(n1 \times n2)$ matrix;
  for (int $i = 0$; $i < n1$; $i++$)
    for (int $j = 0$; $j < n2$; $j++$)
      if $(i/d1 \bmod 2 == j/d2 \bmod 2)$ $A[i][j] = \bullet$;
      else $A[i][j] = \circ$;

Examples of patterns of the Checkerboard family generated by this algorithm are shown in Figure 5. Figure 8 shows a written description of a Checkerboard pattern with $d = 2$. Although it may appear simpler than the algorithmic ones, note that it cannot be as easily adapted to express the family patterns according to the value of d, while the algorithms describe families of patterns. Thus while the Checkerboard pattern may not be a convincing example of the power of recursion, it is a good illustration of the mechanism.

4.2 Sierpinski carpet

The Sierpinski carpet pattern has as inputs the dimension $n = 3^k$ of the meta chart to generate, and the dimension d of the base case of the pattern. The $n \times n$ square is divided into nine $n/3 \times n/3$ squares; the central square is filled with $\bullet$, while the eight squares on the border are

`Cast on over a number` n `of stitches, multiple of 4.`

Row 1: `* knit 2, purl 2; repeat from *.`

Row 2: `repeat row 1.`

Row 3: `* purl 2, knit 2; repeat from *.`

Row 4: `repeat row 2.`

`Repeat rows 1, 2, 3, and 4 for` $n/4$ `times.`

Figure 8. *Written description of a Checkerboard pattern for the case* $d = 2$.

generated recursively. Again, the recursive calls end when the dimension of the squares equals d, at which point the algorithm returns the following base case square

$$\begin{matrix} P\ P\ P \\ P\ K\ P \\ P\ P\ P \end{matrix}$$

where K and P are $d/3 \times d/3$ matrices of symbols $\bullet$ and $\circ$.

Sierpinski(n, d)
  if $(n == d)$ **return** BaseCase(d);
  else
    $S =$ Sierpinski$(n/3, d)$;
    $H =$ new $(n/3 \times n/3)$ matrix of $\bullet$;
    return $\begin{pmatrix} S\ S\ S \\ S\ H\ S \\ S\ S\ S \end{pmatrix}$;

BaseCase(d)
  $K =$ new $(d/3 \times d/3)$ matrix of $\bullet$;
  $P =$ new $(d/3 \times d/3)$ matrix of $\circ$;
  return $\begin{pmatrix} P\ P\ P \\ P\ K\ P \\ P\ P\ P \end{pmatrix}$;

Again, by decreasing d from n down to 3, we obtain patterns of progressively increasing resolution. For examples of the Sierpinski carpet family see Figure 4.

Whereas in the previous example we were able to give a short written description for a fixed value of d, as we will see in Section 6, it is difficult to obtain a similar concise written description for the Sierpinski carpet. This is due to the lack of a corresponding simple iterative description for this "fractal-style pattern".

4.3 Binary tree pattern

Binary trees are graphs defined recursively as follows (see [4]):

Definition 4.1: A *binary tree* T is a structure defined on a finite set of nodes that either contains no nodes, or is composed of three disjoint sets of nodes: a *root* node, a binary tree called its *left subtree*, and a binary tree called its *right subtree*.

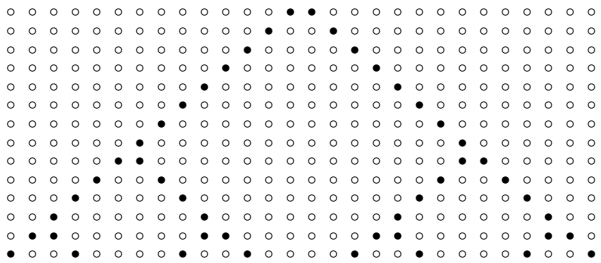


Figure 9. *Meta chart of a Binary tree pattern of dimension* $d = 4$ *and resolution* $\ell = 3$.

A tree with no nodes is called *empty*. If a subtree is nonempty its root is called a *child*. Nodes with one or two children are called *internal nodes*, while nodes without children are called *leaves*. The length of the path from the root of a tree T to a node v is called the *depth* of v in T. For our pattern, we consider only *complete binary trees*:

Definition 4.2: A *complete binary tree* is a binary tree in which all leaves have the same depth and all internal nodes have exactly two children.

Our recursive algorithm generates a matrix representing the meta chart of a complete binary tree, as shown in Figure 9. The inputs are d and ℓ, the dimension of the pattern, and its resolution, respectively. That is, the pattern will be represented as an $n \times 2n$ matrix $T(d, \ell)$, where $n = 2^d - 2$, and the resolution ℓ must be strictly less than d, since ℓ represents the depth of the leaves. The matrix $T(d, \ell)$ can be decomposed into two submatrices C and R:

$$T(d, \ell) = \begin{pmatrix} C \\ R \end{pmatrix},$$

of dimension $(n/2 + 1) \times 2n$ and $(n/2 - 1) \times 2n$, respectively. The upper matrix C is generated during the *combine* step of the *divide et impera* algorithm, while the lower matrix R is constructed after the recursive call. In fact, we have

$$R = \begin{pmatrix} T(d-1,\, \ell-1) & P & T(d-1,\, \ell-1) \end{pmatrix},$$

where, in turn, $T(d-1,\, \ell-1)$ is the matrix representing a binary tree of dimension $d-1$ and resolution $\ell - 1$, and P is a $(n/2 - 1) \times 4$ matrix, filled with $\circ$ symbols. The recursive calls end when $\ell = 1$, as the algorithm returns a simple binary tree composed by the root and two leaves.

CompleteTree(d, ℓ)
  $n = 2^d - 2$;
  if $(\ell == 1)$
    $D =$ new $n \times n$ matrix of $\circ$;
    for $(i = 0;\ i < n;\ i++)$ $\quad D[i][i] = \bullet$;
    $A = D$ with columns in reverse (i.e., right to left) order;
    return $\begin{pmatrix} A & D \end{pmatrix}$;
  else
    $T =$ CompleteTree$(d-1, \ell-1)$;
    $P =$ new $(n/2 - 1) \times 4$ matrix of $\circ$;
    $R = (T\ P\ T)$;
    $C =$ Combine(n);
    return $\begin{pmatrix} C \\ R \end{pmatrix}$;

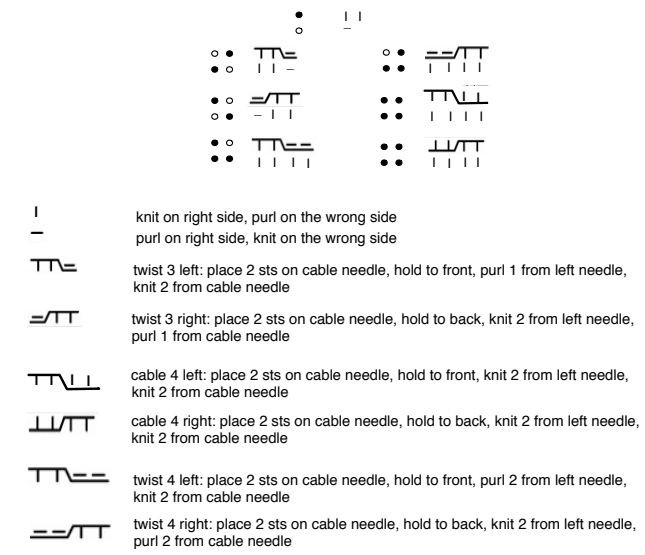


Figure 10. *Transformation rules from meta chart to knitting chart (top) and legend for the cabled charts of the Binary tree and of the Butterfly patterns (bottom).*

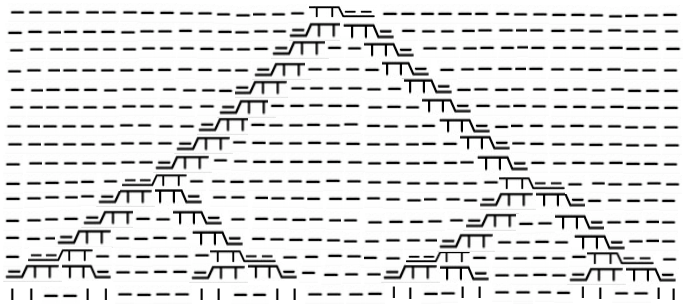


Figure 11. *Chart for cabled realization of the Binary tree pattern (even rows only, stitches in odd rows are knitted as they appear).*

Combine(n)

$D =$ new $(n/2+1) \times n$ matrix of $\circ$;
for $(i = 0;\ i <= n/2;\ i++)$ $\quad D[i][i] = \bullet$;
$A = D$ with columns in reverse (i.e., right to left) order;
return $\begin{pmatrix} A & D \end{pmatrix}$;

By increasing ℓ from 1 to $d-1$, we obtain patterns of progressively increasing resolution.

This pattern can be executed with needles in various ways, depending on the ability of the knitter; beginners could use the basic knit and purl stitches, while skilled knitters could "implement" the tree scheme using more sophisticated stitches. Figure 12 shows a swatch with the pattern executed using the cable stitch. This was based on the knitting chart in Figure 11 obtained from the meta chart according to the legend and the transformation rules in Figure 10. These transformation rules take into account: (1) that the two strands of the cable grow diagonally (twist 3 left and twist 3 right in Figure 10), and (2) they are cabled when they meet and after cabling only one of them survives (twist 4 left and twist 4 right). The choice of whether to cable left or right is purely aesthetic. Note that the knitting chart obtained from the meta chart can be automatically generated by a computer program.

Figure 12. *An "advanced" realization of the Binary tree pattern. 23 cm × 10 cm. Photo* ©*Chiara Bodei.*

4.4 Butterfly pattern

Our last example of a recursive knitting pattern is based on the *Butterfly network* (see [12]):

Definition 4.3: A Butterfly network of degree d (or d-Butterfly) has $(d+1)2^d$ nodes and $d\,2^{d+1}$ edges. The nodes correspond to pairs (w, i) where i is the level of the node ($0 \le i \le d$) and w is a d-bit binary number that denotes the row of the node. Two nodes (w, i) and (w', i') are joined by an edge if and only if $i' = i + 1$ and either w and w' are identical *(straight edge)* or they differ in precisely the i'th bit *(cross edge)*.

As in the previous example, a *divide et impera* algorithm can be used to generate a meta chart representing a butterfly (see Figure 13). For aesthetic reasons, we only consider cross edges.

The input to the algorithm is the degree d of the butterfly network to be generated. The pattern will be represented as an $n \times n$ matrix $B(d)$, where $n = 4(2^d - 1)$. The matrix $B(d)$ is decomposed into two submatrices:

$$B(d) = \begin{pmatrix} R \\ C \end{pmatrix},$$

of dimension $(n/2 - 2) \times n$ and $(n/2 + 2) \times n$, respectively. The matrix C, which represents a multiple cross of 2^d diagonal lines, is generated in the *combine* step of the *divide et impera* algorithm below. The matrix R can be represented as follows:

$$R = \begin{pmatrix} B(d-1) & P & B(d-1) \end{pmatrix},$$

where $B(d-1)$ is the matrix representing a butterfly of degree $d-1$, and P is a $(n/2-2) \times 4$ matrix filled with $\circ$. Due to its structure, R can be conveniently constructed following the recursive call. The recursive calls end when $d = 1$, i.e., $n = 4$, and the algorithm returns a 4×4 matrix representing a single cross.

Butterfly(d)

if $(d == 1)$ **return** $\begin{pmatrix} \bullet & \circ & \circ & \bullet \\ \circ & \bullet & \bullet & \circ \\ \circ & \bullet & \bullet & \circ \\ \bullet & \circ & \circ & \bullet \end{pmatrix}$;

else

$n = 4(2^d - 1)$;

$B =$ Butterfly$(d-1)$;

$P =$ new $(n/2-2) \times 4$ matrix of $\circ$;

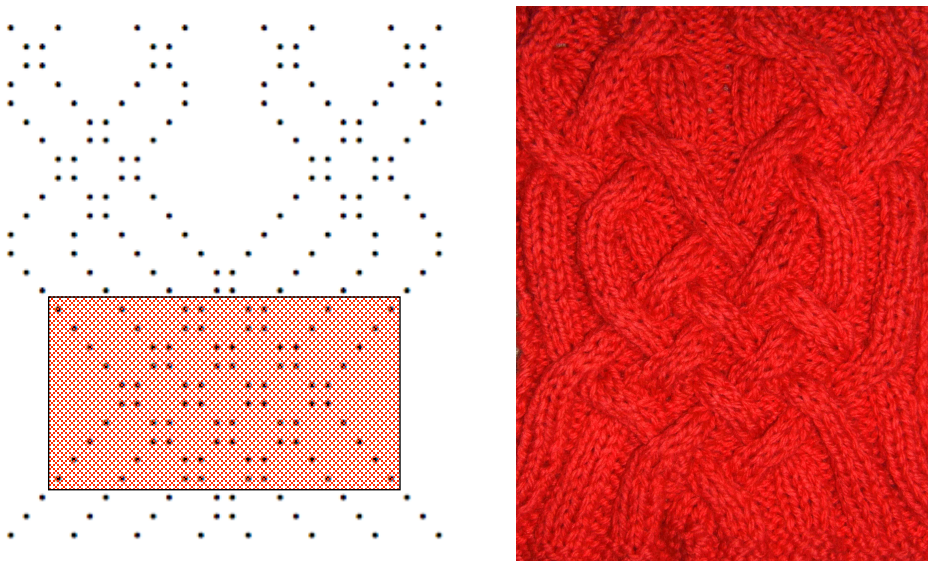


Figure 13. *A 3-Butterfly meta chart recursively generated (left) and its cabled realization (right), 19 cm × 25 cm. Photo* ©*Chiara Bodei. For readability reasons, the symbol ∘ has been replaced by a blank. The highlighted region is expanded in Figure 14.*

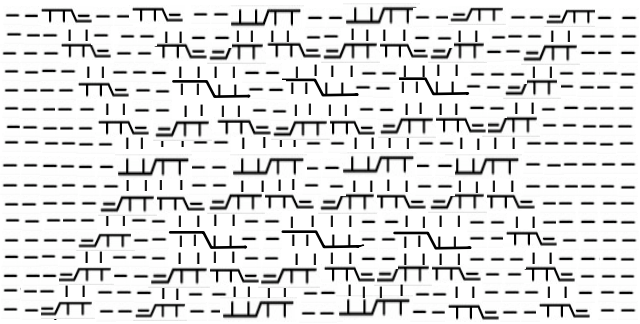


Figure 14. *Chart for cabled realization of a portion of the Butterfly network.*

$R = (B \;\; P \;\; B)$;
$C = \text{Combine}(n)$;
return $\begin{pmatrix} R \\ C \end{pmatrix}$;

Combine(n)

C = new $(n/2+2) \times n$ matrix;
for $(i = 0;\ i < n/2+2;\ i++)$
  for $(j = 0;\ j < n;\ j++)$
    if $((i\text{-}j) \bmod 4 == 0$ **and** $0 \le (j\text{-}i) < n/2)$ $\quad C[i][j] = \bullet$;
    else if $((n\text{-}1\text{-}i\text{-}j) \bmod 4 == 0$ **and** $n/2 \le i{+}j < n)$ $\quad C[i][j] = \bullet$;
    else $\quad C[i][j] = \circ$;

Like the Binary tree, the Butterfly pattern can be realized using just the two basic stitches to obtain the diagonal lines representing its edges or, even better, it can be realized with the cable stitch, as shown in Figure 13, by using part of the chart in Figure 14 in conjunction with the legend and the transformation rules in Figure 10. The choice made for aesthetic reasons between the two different ways of cabling, occurs at the level of the transformation rules. All the cable stitches lying on the same row are of the same type. A row of right cabling will follow one with

left cabling and vice-versa. See for instance row 1 and row 5, from the bottom, in the chart of Figure 14.

5 The Kronecker product and recursive patterns

In this section, we use recursion to generate intricate patterns that we call "fancy patterns" starting from a small initial meta chart together with a recombination rule that operates on stitches. The idea is very simple. An initial meta chart, represented as a $d \times d$ matrix B of symbols $\bullet$ and $\circ$, can either be randomly generated, or chosen by the user according to her/his aesthetic preferences. This small meta chart is then amplified to reach the desired size, by applying a *Kronecker product* type operation, suitably defined for knitting meta charts.

If A is a $m \times n$ matrix and B is a $p \times q$ matrix, recall that the Kronecker product $A \otimes B$ is the $mp \times nq$ block matrix

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix},$$

where $a_{ij}B$ is the matrix obtained by multiplying all elements of B by a_{ij}. In order to apply the Kronecker product to matrices representing meta charts, we must decide how to "multiply" the symbols $\bullet$ and $\circ$, that is, we must define a *binary operation* $op : \{\bullet, \circ\} \times \{\bullet, \circ\} \to \{\bullet, \circ\}$. This binary operation can be chosen in 16 different ways. For example, we could map $\bullet$ to 1, and $\circ$ to 0 and use the logical AND operator. In this case, $A \otimes B$ represents the meta chart obtained by substituting each occurrence of $\bullet$ in A with B, and each occurrence of $\circ$ with a $p \times q$ matrix of $\circ$. If instead we map $\bullet$ to 0 and $\circ$ to 1, and we use the exclusive OR operator, computing $A \otimes B$ means substituting each occurrences of $\bullet$ with B, and each occurrence of $\circ$ with the complement of B, i.e.,the matrix obtained from B by changing every $\circ$ to $\bullet$, and vice-versa.

Once the initial $d \times d$ meta chart B and the operation op has been chosen, the final pattern can be constructed using a recursive algorithm whose inputs are B, op, the dimension n of the pattern to create, and the resolution ℓ ($\ell \leq \lfloor \log_d n \rfloor$).

FancyPattern(B, op, n, ℓ)
  $\otimes_{op}$ = Kronecker product based on the operation op;
  if $(\ell == 1)$
    K = new $(n/d \times n/d)$ matrix of $\bullet$;
    return $K \otimes_{op} B$
  else
    P = FancyPattern$(B, op, n/d, \ell - 1)$;
    return $P \otimes_{op} B$;

The Kronecker product in the base case of the algorithm is used to scale the meta chart B up to the desired dimension n. Observe that even if the initial meta chart B is randomly generated, if we choose a sufficiently large value for the resolution ℓ, i.e., if we execute a considerable number of recursive calls, the final pattern will show a certain regularity.

In Figure 15 we show two meta charts, together with their knitted realizations, generated using this algorithm. In the first example we started from the following 3×3 meta chart

$$\begin{matrix} \bullet & \bullet & \bullet \\ \circ & \bullet & \bullet \\ \bullet & \circ & \bullet \end{matrix}$$

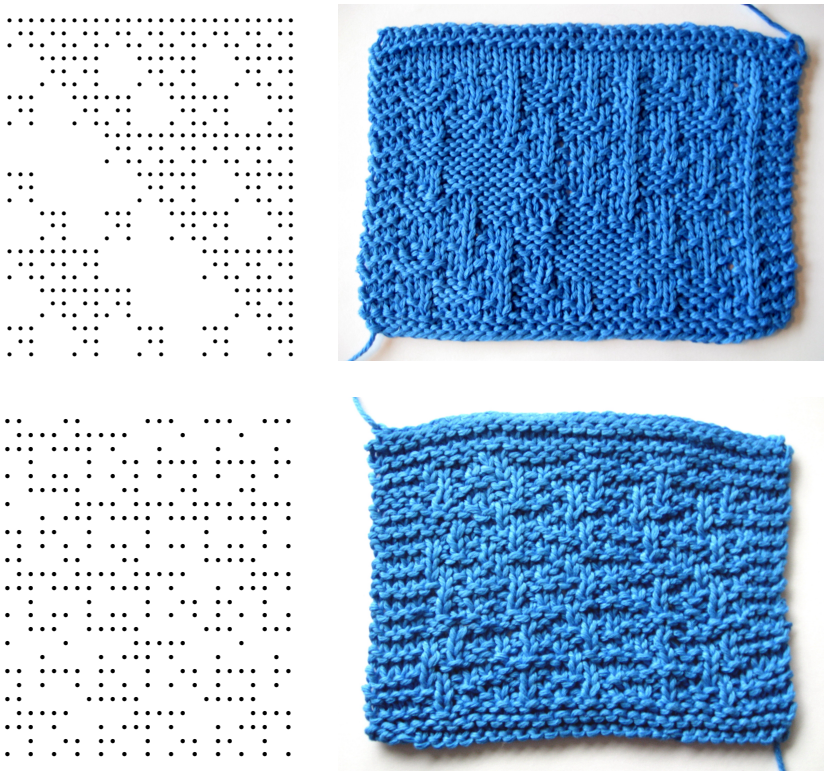


Figure 15. *Fancy recursive knitting meta charts (left) and their knitted realizations (right). For readability reasons, the symbol ∘ has been replaced by a blank. 15 cm × 11 cm (both). Photos ©Anna Bernasconi.*

and set $n = 27$ and $\ell = 3$. We then alternated two different binary operations when computing the Kronecker product in the recursive calls: for odd values of ℓ we used the binary AND operation, while for $\ell = 2$ we used the exclusive OR. In the second example we started from the following 5×5 meta chart randomly generated:

$$\begin{array}{ccccc} \bullet & \bullet & \circ & \circ & \circ \\ \circ & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \circ & \bullet \\ \circ & \circ & \bullet & \circ & \circ \\ \bullet & \circ & \bullet & \circ & \bullet \end{array}$$

we set $n = 25$ and executed two recursive steps ($\ell = 2$), using the exclusive OR operation when computing the Kronecker product. In both examples the two swatches have been obtained mapping the two symbols $\bullet$ and $\circ$ into knit and purl stitches, respectively.

6 Knitting complexity

Whether introduced consciously or unconsciously by the artist, many art works show regularities. Through symmetry and repetition, it is often possible to express the whole in terms of only some of the parts. As we have seen in our examples of knitting patterns, this allows us to give short descriptions. The *Kolmogorov complexity* (also known as algorithmic entropy, or program-size complexity) of an object is a measure of the computational resources needed to specify the object [13]. In [15], Kolmogorov complexity was used to analyze works of art under the assumption that low Kolmogorov complexity was a measure of "beauty", because a short description helps capture the essence of a work of art. We will show that many of the complex patterns previously introduced, described by grammars and recursive algorithms, exhibit low Kolmogorov complexity. By analogy with the standard definition, we introduce the following:

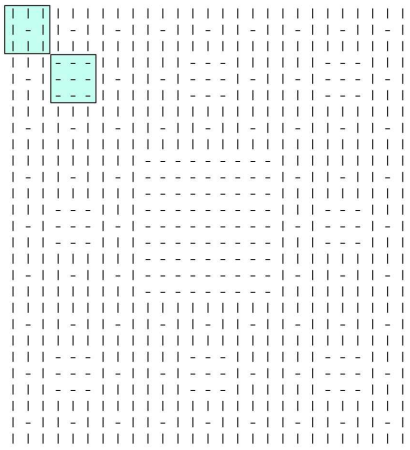


Figure 16. *A Sierpinsky meta chart of size $n = 27$ and $d = 3$, where for readability symbols $\circ$ and $\bullet$ have been replaced by knit and purl symbols.*

Definition 6.1: The *knitting complexity* of a knitting pattern is the length in bits, of the shortest description of its knitting chart.

It is immediate that for an $n \times m$ pattern the knitting complexity is always at least of order $\log n + \log m$ since this many bits are required simply to specify the dimensions of the knitting area. If a pattern does not present any structural regularity, the shortest description of its knitting chart will consist of the chart itself, and since a (binary) chart has $n \times m$ bits, the knitting complexity is always at most of order $n \times m$.

We will now analyze the knitting complexity of the four examples described in Section 4. We have

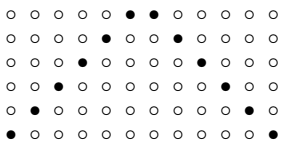
Proposition 6.2: *The Checkerboard, Sierpinski, Binary tree, and Butterfly patterns all have knitting complexity of order $c + \log n$, for some constant c; therefore their recursive descriptions approach the lower bound for knitting complexity as n increases.*

Proof First, observe that for the Binary tree pattern $m = 2n$, while for all the others $m = n$. The result stated is obtained by referring to the algorithmic descriptions of these four patterns specified in Section 4. In fact, each algorithm consists of pseudo-code of constant length c, and since the values for the variables and input parameters are all bounded above by n, their descriptions require at most $\log n$ bits and the result follows. The optimality assertion immediately follows from the fact that we know that at least order of $\log n$ bits are always required and, as n increases, the logarithmic term in $c + \log n$ becomes dominant. □

The problem of finding the minimal knitting complexity is not a trivial task. In general, according to the result of *Non computability of Kolmogorov complexity* [13], it is impossible to derive an algorithm that generates the shortest description for an arbitrary pattern. For our patterns we are able to argue for their optimality only for large values of n.

Is it possible to find any descriptions more succinct than our recursive ones? Consider using the standard technique involving written instructions. As we have already seen, the written Checkerboard pattern can be expressed in a constant number of instructions (see Figure 8), hence it can be expressed succinctly by a non recursive description. For all the other patterns, although we are able to find short descriptions, they are not as succinct as the recursive ones. Moreover, these new descriptions were found only after the recursive nature of the patterns was understood.

To be useful as knitting patterns, n must be small, and consequently the logarithmic value must be small, so we should also consider the role played by the constants when trying to compare recursive and non recursive descriptions. For the Sierpinski carpet, a very short written description can be easily derived when the resolution is low, i.e., when the dimension d of the



`Cast on over an even number` $2n$ `of stitches.`

Row i $(1 \leq i \leq n)$**:** `purl` $i-1$`, knit 1, purl` $2(n-i)$`, knit 1, purl` $i-1$`.`

Figure 17. *Meta chart (top) and written description (bottom) of a Binary tree pattern with resolution* $\ell = 1$.

base case square is equal to n. In all other cases, we have:

Proposition 6.3: *A Sierpinsky carpet of resolution* d *where* $d < n$*, can be described by written instructions in number of bits proportional to* $(n/d)^2 + c' + \log d$*, where* c' *is constant.*

Proof For $d < n$, we know that the pattern can be realized as the composition $d \times d$ squares of two different types, as shown in Figure 16. For fixed d, due to the nature of their regularity, each of these two squares can be described using $c' + \log d$ bits. The meta chart can therefore now be described as an $n/d \times n/d$ matrix of squares, and to each of them we have to assign one of two possible charts. In this way we obtain a description of size proportional to $(n/d)^2 + c' + \log d$. □

Note that, even if the base case routine must be invoked for each $d \times d$ square, it can be described only once, because in a non recursive program, the routine is inserted inside a *for* or a *while* cycle.

Let RS be the recursive Sierpinsky algorithm given in Section 4, and S be the algorithm formulated according to the proof of Proposition 6.3. Let c be the constant found in the knitting complexity for RS and c' the one for S. By ignoring the logarithmic terms, we have that RS compares favorably with S when $n^2/d^2 > c - c'$, that is for decreasing values of d, and therefore for increasing resolution.

The Binary tree pattern can also be described by a constant number of written instructions when the resolution is very low, i.e., when ℓ is a constant. For instance, if $\ell = 1$, we can describe the pattern, realized with knit and purl stitches, as shown in Figure 17. For increasing resolution, however, the description becomes more involved and we were not able to find a description more succinct than the recursive one. A similar result holds for the Butterfly pattern.

We finally observe that the recursive patterns described in Section 5 have knitting complexity of order ranging from $c + \log n + \log d$ to $c' + \log n + d^2$, where c and c' are constants, and d is the dimension of the initial meta chart B. The lowest value is attained when B presents a very regular structure, while the highest value is attained when B is a random meta chart whose smallest description is the meta chart itself.

We have shown that the recursive formulation of complex patterns can give succinct and elegant descriptions. On the other hand, the comprehensibility of a description decreases with the use of recursion. Compare for instance the two descriptions of the Checkerboard pattern, the non recursive formulation is much clearer for most knitters, with possibly the exception of computer programmers!

7 Conclusions and Future Work

Summarizing, we have shown that:

- Complex knitting patterns can be described succinctly by grammars from the theory of formal languages or collage grammars typically used for two-dimensional pattern generation.
- Complex knitting patterns can be also generated using recursive and iterative algorithms.

- The knitting complexity of a pattern description allows for the comparison of its different descriptions, in order to establish in some sense which is the most effective.
- Algorithmic tools can be used for the creation of new patterns.

As future work we plan to extend the recursive pattern generator discussed in Section 5, by allowing a richer set of initial choices, combination rules and binary operations, setting up a nice graphical user interface, so that the user could set up her/his personal pattern generator.

Apart from their theoretical interest, the above results also have a practical impact. In fact, using large values for recursion depth and high values for resolution, we can automate in simple way, the design of arbitrarily complex patterns that, to our knowledge, have never before been considered. We can also automatically generate their knitting charts. Although these charts may exceed the skill and patience of human knitters, they are amenable to knitting machines whose execution time is independent of the difficulty of the chart.

Although visually complex, almost all our patterns exhibit low knitting complexity: this parameter can be formally computed and, if we are to believe [15], is associated with the concept of beauty. Perhaps, then, low knitting complexity can be used as a guideline when designing new patterns.

Acknowledgements

We intend to thank our anonymous referees and our editor for their careful reading of our preliminary draft and for their constructive comments and suggestions.

References

[1] P. Allen, T. Malcolm, R. Tennant, and C. Fall, *Knitting for Dummies*, Wiley Publishing, Inc., Indianapolis, Indiana, 2002.
[2] A. Bernasconi, C. Bodei and L. Pagli. *Knitting for Fun: A Recursive Sweater*. Proceedings of Fun with Algorithms, 4th International Conference, FUN 2007, Castiglioncello, Italy, Lecture Notes in Computer Science 4475, pp. 53-65, Springer, 2007.
[3] *Celtic Knot Cable Stitch*, `http://pinebaskets.tripod.com/knitting/celticknotcable.html`. (Accessed, 28 April 2008).
[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms, second edition*, The MIT Press, Cambridge, MA, 2001.
[5] F. Drewes, *Tree-based picture generation*, Theoretical Computer Science 246(1-2) (2000), pp. 1-51.
[6] F. Drewes and R. Klempien-Hinrichs, *Picking Knots from Trees – The Syntactic Structure of Celtic Knotwork*, Proceedings of the First International Conference, Diagrams 2000, Edinburgh, Scotland, UK, Lecture Notes in Computer Science 1889, pp. 89-104, Springer-Verlag, 2000.
[7] F. Drewes, R. Klempien-Hinrichs and H.J. Kreowski. *Table-driven and context-sensitive collage languages*. Developments in Language Theory (1999), pp. 326-337.
[8] D. New, *Unexpected knitting*, Schoolhouse Press, Pittswille, 2007.
[9] A. Ekart, *Genetic Programming for the Design of Lace Knitting Stitch Patterns*, Proceedings of Genetic and Evolutionary Computation Conference (GECCO), London, United Kingdom, Bremen, Germany, Lecture Notes in Computer Science 2278, Springer, pp. 2457-2461, 2007.
[10] A. Habel, H.J. Kreowski. *Collage Grammars* Proceedings of Graph-Grammars and Their Application to Computer Science, Lecture Notes in Computer Science 532, Springer, pp. 411-429, 1990.
[11] J.E. Hopcroft, R. Motwani, J.D. Ullman *Introduction to Automata Theory, Languages, and Computation*, III edition, Pearson Education, Inc., Old Tappan, NJ, 2007.
[12] F.T. Leighton, *Parallel Algorithms and Architectures: Array, Trees, Hypercubes*, Morgan Kaufmann Publishers, San Mateo, CA (1992).
[13] M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*, Springer Verlag, New York (2005).
[14] G. Rozenberg. *TOL Systems and languages* Information and Control, 23(4), (1973), pp. 262-283.
[15] J. Schmidhuber, *Low-Complexity Art*, Leonardo, Journal of the International Society for the Arts, Science and Technology, vol. 30:2, MIT Press (1997), pp. 97-103.
[16] D. Suzuki, T. Miyazaki, K. Yamada, T. Nakamura, and H. Itoh, *A Supporting System for Colored Knitting Design*, Proceedings of the 13th Int. Conf. on Industrial and engineering applications of artificial intelligence and expert systems, IEA/AIE, New Orleans, Louisiana, USA, Lecture Notes in Computer Science 1821, Springer-Verlag, 2000.
[17] *The Home of Mathematical Knitting*, `http://www.toroidalsnark.net/mathknit.html`. (Accessed, 28 April 2008).
[18] *The girl from auntie*, `http://www.thegirlfromauntie.com/patterns/celtic`. (Accessed, 28 April 2008).

List of Figure Captions

- Figure 1: Written description (top) and knitting chart (bottom) of Mistake Rib.
- Figure 2: Knitting Charts for Stockinette (a), Reverse Stockinette (b), Seed Stitch (c), Garter Stitch (d), Ribbing 1x1 (e), respectively.
- Figure 3: Checkerboard: knitting chart (a), meta chart (b).
- Figure 4: Sierpinski meta charts with increasing resolution.
- Figure 5: Checkerboard meta charts with subsquares of increasing dimension.
- Figure 6: A derivation tree obtained by applying to the initial axiom the productions in table T_2.
- Figure 7: A derivation tree obtained, by first applying the productions in table T_1 and afterwards by applying the productions in T_2.
- Figure 8: Written description of a Checkerboard pattern for a fixed value d = 2.
- Figure 9: Meta chart of a Binary tree pattern of dimension $d = 4$ and resolution $\ell = 3$.
- Figure 10: Transformation rules from meta chart to knitting chart (top) and legend for the cabled charts of the Binary tree and of the Butterfly patterns (bottom).
- Figure 11: Chart for cabled realization of the Binary tree pattern (even rows only, stitches in odd rows are knitted as they appear).
- Figure 12: An "advanced" realization of the Binary tree pattern. 23 cm × 10 cm. Photo ©Chiara Bodei.
- Figure 13: A 3-Butterfly meta chart recursively generated (left) and its cabled realization (right), 19 cm × 25 cm. Photo ©Chiara Bodei. For readability reasons, the symbol ∘ has been replaced by a blank. The highlighted region is expanded in Figure 14.
- Figure 14: Chart for cabled realization of a portion of the Butterfly network.
- Figure 15: Fancy recursive knitting meta charts (left) and their knitted realizations (right). For readability reasons, the symbol ∘ has been replaced by a blank. 15 cm × 11 cm (both). Photos ©Anna Bernasconi.
- Figure 16: A Sierpinsky meta chart of size $n = 27$ and $d = 3$, where for readability symbols ∘ and • have been replaced by knit and purl symbols.
- Figure 17: Meta chart (top) and written description (bottom) of a Binary tree pattern with resolution $\ell = 1$.