Chapter 14

Walkabout: Transition graphs

I think I’ll go on a walkabout
find out what it’s all about [...] take a ride to the other side
—Red Hot Chili Peppers, ‘Walkabout’

In Chapters 11 and 12 we learned that invariant manifolds partition the state
space in invariant way, and how to name distinct orbits. We have established
and related the temporally and spatially ordered topological dynamics for a
class of ‘stretch & fold’ dynamical systems, and discussed pruning of inadmissi-
ble trajectories.

Here we shall use these results to generate the totality of admissible itineraries.
This task will be particularly easy for repellers with complete Smale horseshoes
and for subshifts of finite type, for which the admissible itineraries are generated
by finite transition matrices, and the topological dynamics can be visualized by
means of finite transition graphs. We shall then turn topological dynamics into a
linear multiplicative operation on the state space partitions by means of transition
matrices, the simplest examples of ‘evolution operators.’ They will enable us – in
chapter 15 – to count the distinct orbits.

14.1 Matrix representations of topological dynamics

The allowed transitions between the regions of a partition \( \{M_1, M_2, \cdots, M_m\} \) are
encoded in the \([m \times m]\)-dimensional transition matrix whose elements take values

\[
T_{ij} = \begin{cases} 
1 & \text{if the transition } M_j \to M_i \text{ is possible} \\
0 & \text{otherwise}.
\end{cases}
\]  

(14.1)

The transition matrix is an explicit linear representation of topological dynam-
icity. If the partition is a dynamically invariant partition constructed from sta-
ble/unstable manifolds, it encodes the topological dynamics as an invariant law
FIGURE 14.1: Points from the region \( M_{21} \) reach regions \( \{M_{10}, M_{11}, M_{12}\} \), and no other regions, in one time step. Labeling exemplifies the ‘shift map’ of example 11.7 and (11.20).

of motion, with the allowed transitions at any instant independent of the trajectory history, requiring no memory.

Several related matrices as well will be needed in what follows. Often it is convenient to distinguish between two or more paths connecting the same two regions; that is encoded by the adjacency matrix with non-negative integer entries,

\[
A_{ij} = \begin{cases} 
  k & \text{if a transition } M_j \to M_i \text{ is possible in } k \text{ ways} \\
  0 & \text{otherwise} 
\end{cases} \quad (14.2)
\]

More generally, we shall encounter \([m \times m]\) matrices which assign different real or complex weights to different transitions,

\[
L_{ij} = \begin{cases} 
  L_{ij} \in \mathbb{R} \text{ or } \mathbb{C} & \text{if } M_j \to M_i \text{ is allowed} \\
  0 & \text{otherwise} 
\end{cases} \quad (14.3)
\]

As in statistical physics, we shall refer to these as transfer matrices.

\( M_i \) is accessible from \( M_j \) in \( k \) steps if \( (L^k)_{ij} \neq 0 \). A matrix \( L \) is called reducible if there exists one or more index pairs \( \{i, j\} \) such that \( (L^k)_{ij} = 0 \) for all \( k \), otherwise the matrix is irreducible. This means that a trajectory starting in any partition region eventually reaches all of the partition regions, i.e., the partition is dynamically transitive or indecomposable, as assumed in (2.2). The notion of topological transitivity is crucial in ergodic theory: a mapping is transitive if it has a dense orbit. If that is not the case, state space decomposes into disconnected pieces, each of which can be analyzed separately by a separate irreducible matrix.

Region \( M_i \) is said to be transient if no trajectory returns to it. Region \( M_j \) is said to be absorbing if no trajectory leaves it, \( L_{jj} \neq 0, L_{ij} = 0 \) for all \( i \neq j \). Hence it suffices to restrict our considerations to irreducible matrices.

If \( L \) has strictly positive entries, \( L_{ij} > 0 \), the matrix is called positive; if \( L_{ij} \geq 0 \), the matrix is called non-negative. Matrix \( L \) is said to be eventually positive or Perron-Frobenius if \( L^k \) is positive for some power \( k \) (as a consequence, the matrix is transitive as well). A non-negative matrix whose columns conserve probability, \( \sum_j L_{ij} = 1 \), is called Markov, probability or stochastic matrix.
Figure 14.2: Topological dynamics: shrink each state space partition region figure 14.1 to a node, and indicate the possibility of reaching a region by a directed link. The links stand for transition matrix elements $T_{10,21} = T_{11,21} = T_{12,21} = 1$; remaining $T_{i,j} = 0$.

Example 14.1 Markov chain. The Google PageRank of a webpage is computed by a Markov chain, with a rather large Markov matrix $M$.

A subshift (11.22) of finite type is a topological dynamical system $(\Sigma, \sigma)$, where the shift $\sigma$ acts on the space of all admissible itineraries $(s_k)$

$$
\Sigma = \{(s_k)_{k \in \mathbb{Z}} : T_{s_{k+1}, s_k} = 1 \text{ for all } k\}, \quad s_k \in \{a, b, c, \ldots, z\}. \quad (14.4)
$$

The task of generating the totality of admissible itineraries is particularly easy for subshifts of finite type, for which the admissible itineraries are generated by finite transition matrices, and the topological dynamics can be visualized by means of finite transition graphs.

### 14.2 Transition graphs: wander from node to node

Let us abstract from a state space partition such as figure 14.1 its topological essence: indicate a partition region $M_a$ by a node, and indicate the possibility of reaching the region $M_b, L_{ba} \neq 0$ by a directed link, as in figure 14.2. Do this for all nodes. The result is a transition graph.

A transition graph (or digraph, or simply ‘graph’) consists of a set of nodes (or vertices, or states), one for each letter in the alphabet $\mathcal{A} = \{a, b, c, \ldots, z\}$, connected by a set of directed links (edges, arcs, arrows). A directed link starts out from node $j$ and terminates at node $i$ whenever the matrix element (14.3) takes value $L_{ij} \neq 0$. A link connects two nodes, or originates and terminates on the same node (a ‘self-loop’). For example, if a partition includes regions labeled $\{\cdots, M_{101}, M_{110}, \cdots\}$, the transition matrix element connecting the two is drawn as $L_{101,110} = \textcircled{101} \rightarrow \textcircled{110}$, whereas $L_{0,0} = \textcircled{0} \rightarrow \textcircled{0}$. Here a dotted link indicates that the shift $\sigma(x_{011\cdots}) = x_{11\cdots}$ involves symbol 0, and a full one a shift $\sigma(x_{110\cdots}) = x_{10\cdots}$ that involves 1. A $j \rightarrow \cdots \rightarrow k$ walk (path, itinerary) traverses a connected set of directed links, starting at node $j$ and ending at node $k$. A loop (periodic orbit, cycle) is a walk that ends at the starting node (which can be any node along the loop), for example

$$
t_{011} = L_{110,011} L_{011,101} L_{101,110} = \textcircled{101} \rightarrow \textcircled{110} \rightarrow \textcircled{011} \rightarrow \textcircled{110}. \quad (14.5)
$$
Our convention for ordering indices is that the successive steps in a visitation sequence \( j \to i \to k \) are generated by matrix multiplication from the left, \( T_{kj} = \sum T_{ki}T_{ij} \). Two graphs are isomorphic if one can be obtained from the other by relabeling links and nodes. As we are interested in recurrent (transitive, indecomposable) dynamics, we restrict our attention to irreducible or strongly connected graphs, i.e., graphs for which there is a path from any node to any other node.

A transition graph describes compactly the ways in which the state space regions map into each other, accounts for finite memory effects in dynamics, and generates the totality of admissible trajectories as the set of all possible walks along its links.

Construction of a good transition graph is, like combinatorics, unexplainable. The only way to learn is by some diagrammatic gymnastics, so we work our way through a sequence of exercises in lieu of plethora of baffling definitions.

Example 14.2 Full binary shift. Consider a full shift on two-state partition \( \mathcal{A} = \{0, 1\} \), with no pruning restrictions. The transition matrix and the corresponding transition graph are

\[
T = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = \begin{array}{ccc}
& 0 & \\
& 0 & \\
& 0 & \\
0 & 0 & 0
\end{array}
\]  

(14.6)

Dotted links correspond to shifts originating in region 0, and the full ones to shifts originating in 1. The admissible itineraries are generated as walks on this transition graph. (continued in example 14.8)

Example 14.3 Complete \( N \)-ary dynamics: If all transition matrix entries equal unity (one can reach any region from any other region in one step),

\[
T_c = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ 1 & 1 & \ldots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \ldots & 1 \end{pmatrix}
\]  

(14.7)

the symbolic dynamics is called complete, or a full shift. The corresponding transition graph is obvious, but a bit tedious to draw for arbitrary \( N \).

Example 14.4 Pruning rules for a 3-disk alphabet: As the disks are convex, there can be no two consecutive reflections off the same disk, hence the covering symbolic dynamics consists of all sequences which include no symbol repetitions 11, 22, 33. This is a finite set of finite length pruning rules, hence, the dynamics is a subshift of finite type (see (11.23) for definition), with the transition matrix / graph given by

\[
T = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{array}{ccc}
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0
\end{array}
\]  

(14.8)
The complete unrestricted symbolic dynamics is too simple to be illuminating, so we turn next to the simplest example of pruned symbolic dynamics, the finite subshift obtained by prohibition of repeats of one of the symbols, let us say 1L. This situation arises, for example, in studies of the circle maps, where this kind of symbolic dynamics describes “golden mean” rotations.

Example 14.5 ‘Golden mean’ pruning. Consider a subshift on two-state partition \( \mathcal{A} = \{0, 1\} \), with the simplest grammar \( G \) possible, a single pruned block \( b = 11 \) (consecutive repeat of symbol 1 is inadmissible): the state \( M_0 \) maps both onto \( M_0 \) and \( M_1 \), but the state \( M_1 \) maps only onto \( M_0 \). The transition matrix and the corresponding transition graph are

\[
T = \begin{pmatrix}
1 & 1 \\
1 & 0
\end{pmatrix} = \quad \begin{array}{c}
0 \\
1
\end{array}
\begin{array}{c}
0 \\
1
\end{array}
\]

(14.9)

Admissible itineraries correspond to walks on this finite transition graph. (continued in example 14.9)

In the complete \( N \)-ary symbolic dynamics case (see example 14.3) the choice of the next symbol requires no memory of the previous ones. However, any further refinement of the state space partition requires finite memory.

Example 14.6 Finite memory transition graphs. For the binary labeled repeller with complete binary symbolic dynamics, we might chose to partition the state space into four regions \( \{M_{00}, M_{01}, M_{10}, M_{11}\} \), a 1-step refinement of the initial partition \( \{M_0, M_1\} \). Such partitions are drawn in figure 12.3, as well as figure 1.9. Topologically \( f \) acts as a left shift (12.11), and its action on the rectangle \([0.1]\) is to move the decimal point to the right, to \([0.1]\), forget the past, \([.1]\), and land in either of the two rectangles \(\{[10], [11]\}\). Filling in the matrix elements for the other three initial states we obtain the 1-step memory transition matrix/graph acting on the 4-regions partition

\[
T = \begin{pmatrix}
T_{00,00} & 0 & T_{00,10} & 0 \\
T_{01,00} & 0 & T_{01,10} & 0 \\
0 & T_{10,01} & 0 & T_{10,11} \\
0 & T_{11,01} & 0 & T_{11,11}
\end{pmatrix} = \quad \begin{array}{c}
0 \\
1 \\
0 \\
1
\end{array}
\begin{array}{c}
0 \\
1 \\
0 \\
1
\end{array}
\]

(14.10)

(continued in example 15.7)

By the same token, for \( M \)-step memory the only nonvanishing matrix elements are of the form \( T_{s_1s_2...s_{M+1},s_{M+1}} \), \( s_1s_2...s_{M+1} \in \{0, 1\} \). This is a sparse matrix, as the only non vanishing entries in the \( a = s_0s_1...s_M \) column of \( T_{ba} \) are in the rows \( b = s_1...s_M0 \) and \( b = s_1...s_M1 \). If we increase the number of steps remembered, the transition matrix grows large quickly, as the \( N \)-ary dynamics with \( M \)-step memory requires an \([N^{M+1} \times N^{M+1}]\) matrix. Since the matrix is very sparse, it pays to find a compact representation for \( T \). Such representation is afforded by transition graphs, which are not only compact, but also give us an intuitive picture of the topological dynamics.
Figure 14.3: Transition graph (graph whose links correspond to the nonzero elements of a transition matrix $T_{ba}$) describes which regions $b$ can be reached from the region $a$ in one time step. The 7 nodes correspond to the 7 regions of the partition (14.11). The links represent non-vanishing transition matrix elements, such as $T_{101,110} = \frac{1}{2}$. Dotted links correspond to a shift by symbol 0, and the full ones by symbol 1.

Figure 14.4: The self-similarity of the complete binary symbolic dynamics represented by a binary tree: trees originating in nodes $B, C, \ldots$ (actually - any node) are the same as the tree originating in node $A$. Level $m = 4$ partition is labeled by 16 binary strings, coded by dotted (0) and full (1) links read down the tree, starting from $A$. See also figure 11.14.

Example 14.7 A 7-state transition graph. Consider a state space partitioned into 7 regions

$$\{M_{00}, M_{011}, M_{010}, M_{110}, M_{111}, M_{101}, M_{100}\}. \quad (14.11)$$

Let the evolution in time map the regions into each other by acting on the labels as shift (12.11): $M_{011} \rightarrow \{M_{110}, M_{111}\}$, $M_{00} \rightarrow \{M_{00}, M_{011}, M_{010}\}$, with nonvanishing $L_{110,011}$, $L_{011,00}$. This is compactly summarized by the transition graph of figure 14.3.

(continued as example 15.6)

14.3 Transition graphs: stroll from link to link

What do finite graphs have to do with infinitely long trajectories? To understand the main idea, let us construct a graph that enumerates all possible itineraries for the case of complete binary symbolic dynamics. In this construction the nodes will be unlabeled, links labeled, signifying different kinds of transitions.

Example 14.8 Complete binary topological dynamics. Mark a dot '.' on a piece of paper. Draw two short lines out of the dot, end each with a dot. The full line will signify that the first symbol in an itinerary is '1', and the dotted line will signify '0.' Repeat the procedure for each of the two new dots, and then for the four dots, and so on. The result is the binary tree of figure 14.4. Starting at the top node, the tree enumerates exhaustively all distinct finite itineraries of lengths $n = 1, 2, 3, \ldots$

$$\{0, 1\} \{00, 01, 10, 11\}$$
$$\{000, 001, 010, 011, 100, 101, 111, 110\} \ldots.$$

The $n = 4$ nodes in figure 14.4 correspond to the 16 distinct binary strings of length 4, and so on. By habit we have drawn the tree as the alternating binary tree of figure 11.14, but that has no significance as far as enumeration of itineraries is concerned.
- a binary tree with labels in the natural order, as increasing binary ‘decimals’ would serve just as well.

The trouble with an infinite tree is that it does not fit on a piece of paper. On the other hand, we are not doing much - at each node we are turning either left or right. Hence all nodes are equivalent. In other words, the tree is self-similar; the trees originating in nodes $B$ and $C$ are themselves copies of the entire tree. The result of identifying $B = A$, $C = A$ is a single node, 2-link transition graph with adjacency matrix figure 14.2

$$A = (2) = \begin{pmatrix} A & B & C \end{pmatrix}. \quad (14.12)$$

An itinerary generated by the binary tree figure 14.4, no matter how long, corresponds to a walk on this graph.

This is the most compact encoding of the complete binary symbolic dynamics. Any number of more complicated transition graphs such as the 2-node (14.6) and the 4-node (14.10) graphs generate all itineraries as well, and might be sometimes preferable.

We turn next to the simplest example of pruned symbolic dynamics, the finite subshift obtained by prohibition of repeats of one of the symbols, let us say _00_.

**Example 14.9 ‘Golden mean’ pruning.** (a link-to-link version of example 14.5) Now the admissible itineraries are enumerated by the pruned binary tree of figure 14.5. Identification of nodes $A = C = E$ leads to the finite 2-node, 3-links transition graph

$$T = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} B & A=C=E \end{pmatrix}. \quad (14.13)$$

As 0 is always followed by 1, the walks on this graph generate only the admissible itineraries. This is the same graph as the 2-node graph (14.9). (continued in example 15.4)
14.3.1 Converting pruning blocks into transition graphs

Suppose now that, by hook or crook, you have been so lucky fishing for pruning rules that you now know the grammar (11.23) in terms of a finite set of pruning blocks \( G = \{ b_1, b_2, \ldots, b_k \} \), of lengths \( \leq m \). Our task is to generate all admissible itineraries. What to do?

We have already seen the main ingredients of a general algorithm: (1) transition graph encodes self-similarities of the tree of all itineraries, and (2) if we have a pruning block of length \( m \), we need to descend \( m \) levels before we can start identifying the self-similar sub-trees.

Finite grammar transition graph algorithm.

1. Starting with the root of the tree, delineate all branches that correspond to all pruning blocks; implement the pruning by removing the last node in each
pruning block (marked ‘x’ in figure 14.6 (a)).

2. Label all nodes internal to pruning blocks by the itinerary connecting the root point to the internal node, figure 14.6 (b). Why? So far we have pruned forbidden branches by looking $m_b$ steps into future for a given pruning block, let’s say $b = 10110$. However, the blocks with a right combination of past and future [1.0110], [10.110], [101.10] and [1011.0] are also pruned. In other words, any node whose near past coincides with the beginning of a pruning block is potentially dangerous - a branch further down the tree might get pruned.

3. Add to each internal node all remaining branches allowed by the alphabet, and label them, figure 14.6 (c). Why? Each one of them is the beginning point of an infinite tree, a tree that should be similar to another one originating closer to the root of the whole tree.

4. Pick one of the free external nodes closest to the root of the entire tree, forget the most distant symbol in its past. Does the truncated itinerary correspond to an internal node? If yes, identify the two nodes. If not, forget the next symbol in the past, repeat. If no such truncated past corresponds to any internal node, identify with the root of the tree.

This is a little bit abstract, so let's say the free external node in question is [1010.]. Three time steps back the past is [010.]. That is not dangerous, as no pruning block in this example starts with 0. Now forget the third step in the past: [10.] is dangerous, as that is the start of the pruning block [10.110]. Hence the free external node [1010.] should be identified with the internal node [10.].

5. Repeat until all free nodes have been tied back into the internal nodes.

6. Clean up: check whether every node can be reached from every other node. Remove the transient nodes, i.e., the nodes to which dynamics never returns.

7. The result is a transition graph. There is no guarantee that this is the smartest, most compact transition graph possible for given pruning (if you have a better algorithm, teach us), but walks around it do generate all admissible itineraries, and nothing else.

---

Example 14.10 Heavy pruning.

We complete this training by examples by implementing the pruning of figure 12.11 (b). The pruning blocks are

$$\{100.10, [10.1], [010.01], [011.01], [11.1], [101.10]\}. \quad (14.14)$$

Blocks 01101, 10110 contain the forbidden block 101, so they are redundant as pruning rules. Draw the pruning tree as a section of a binary tree with 0 and 1 branches and label each internal node by the sequence of 0’s and 1’s connecting it to the root of the tree (figure 14.6 (a)). These nodes are the potentially dangerous nodes - beginnings of blocks that might end up pruned. Add the side branches to those nodes (figure 14.6 (b)).
As we continue down such branches we have to check whether the pruning imposes constraints on the sequences so generated: we do this by knocking off the leading bits and checking whether the shortened strings coincide with any of the internal pruning tree nodes: \[00 \rightarrow 0; \quad 110 \rightarrow 10; \quad 011 \rightarrow 11; \quad 0101 \rightarrow 101\ (pruned); \quad 1000 \rightarrow 00 \rightarrow 00; \quad 10011 \rightarrow 0011 \rightarrow 011 \rightarrow 11; \quad 01000 \rightarrow 0.\]

The trees originating in identified nodes are identical, so the tree is “self-similar.”

Now connect the side branches to the corresponding nodes, figure 14.6 (d). Nodes “.” and 1 are transient nodes; no sequence returns to them, and as you are interested here only in infinitely recurrent sequences, delete them. The result is the finite transition graph of figure 14.6 (d); the admissible bi-infinite symbol sequences are generated as all possible walks on this graph.

Résumé

The set of all admissible itineraries is encoded multiplicatively by transition matrices, diagrammatically by transition graphs. Pruning rules for inadmissible sequences are implemented by constructing corresponding transition matrices and/or transition graphs.

Commentary

**Remark 14.1** Transition graphs. We enjoyed studying Lind and Marcus [14.1] introduction to symbolic dynamics and transition graphs. Finite transition graphs or finite automata are discussed in refs. [14.2, 14.3, 14.4]. They belong to the category of regular languages. Transition graphs for unimodal maps are discussed in refs. [14.8, 14.9, 14.10]. (see also remark 11.1)

**Remark 14.2** Inflating transition graphs. In the above examples the symbolic dynamics has been encoded by labeling links in the transition graph. Alternatively one can encode the dynamics by labeling the nodes, as in example 14.6, where the 4 nodes refer to 4 Markov partition regions \(M_{00}, M_{01}, M_{10}, M_{11}\), and the 8 links to the 8 non-zero entries in the 2-step memory transition matrix (14.10).

**Remark 14.3** The unbearable growth of transition graphs. A construction of finite Markov partitions is described in refs. [14.11, 14.12], as well as in the innumerably many other references.

If two regions in a Markov partition are not disjoint but share a boundary, the boundary trajectories require special treatment in order to avoid overcounting, see sect. 21.3.1. If the image of a trial partition region cuts across only a part of another trial region and thus violates the Markov partition condition (11.2), a further refinement of the partition is needed to distinguish distinct trajectories.
The finite transition graph construction sketched above is not necessarily the minimal one; for example, the transition graph of figure 14.6 does not generate only the “fundamental” cycles (see chapter 20), but shadowed cycles as well, such as \( f_{0011} \) in (15.20). For methods of reduction to a minimal graph, consult refs. [14.8, 12.49, 14.9]. Furthermore, when one implements the time reversed dynamics by the same algorithm, one usually gets a graph of a very different topology even though both graphs generate the same admissible sequences, and have the same determinant. The algorithm described here makes some sense for 1-dimensional dynamics, but is unnatural for 2-dimensional maps whose dynamics it treats as 1-dimensional. In practice, generic pruning grows longer and longer, and more plentiful pruning rules. For generic flows the refinements might never stop, and almost always we might have to deal with infinite Markov partitions, such as those that will be discussed in sect. 15.5. Not only do the transition graphs get more and more unwieldy, they have the unpleasant property that every time we add a new rule, the graph has to be constructed from scratch, and it might look very different from the previous one, even though it leads to a minute modification of the topological entropy. The most determined effort to construct such graphs may be the one of ref. [12.14]. Still, this seems to be the best technology available, unless the reader alerts us to something superior.
Exercises

14.1. Time reversibility,∗∗ Hamiltonian flows are time reversible. Does that mean that their transition graphs are symmetric in all node → node links, their transition matrices are adjacency matrices, symmetric and diagonalizable, and that they have only real eigenvalues?

14.2. Alphabet \{0,1\}, prune _100_, _00100_, _01100_. This example is motivated by the pruning front description of the symbolic dynamics for the Hénon-type mapsremark 12.3.

step 1. _100_ prunes all cycles with a _00_ subsequence with the exception of the fixed point 0; hence we factor out \(1 - t_0\) explicitly, and prune _00_ from the rest. This means that \(x_0\) is an isolated fixed point - no cycle stays in its vicinity for more than 2 iterations. In the notation of sect. 14.3.1, the alphabet is \{1, 2, 3; \(\emptyset\)\}, and the remaining pruning rules have to be rewritten in terms of symbols 2=10, 3=100;

step 2. alphabet \{1, 2, 3; \(\emptyset\)\}, prune _33_, _213_, _313_. This means that the 3-cycle 3 = 100 is pruned and no long cycles stay close enough to it for a single _100_ repeat. As in example 1?!. prohibition of _33_ is implemented by dropping the symbol “3” and extending the alphabet by the allowed blocks 13, 23:

step 3. alphabet \{1, 2, 13, 23; \(\emptyset\)\}, prune _213_, _23_, _13_. where 13 = 13, 23 = 23 are now used as single letters. Pruning of the repetitions _13_ (the 4-cycle 13 = 1100 is pruned) yields the

result: alphabet \{1, 2, 13, 113; \(\emptyset\)\}, unrestricted 4-ary dynamics. The other remaining possible blocks _213_, _23_ are forbidden by the rules of step 3.

References


