

Algebraic Properties of Automata Associated to Petri Nets and Applications to Computation in Biological Systems

Attila Egri-Nagy and Chrystopher L. Nehaniv

*The Royal Society/Wolfson Foundation BioComputation Laboratory & Algorithms Research Group
Centre for Computer Science and Informatics Research
University of Hertfordshire
College Lane, Hatfield, Hertfordshire AL10 9AB, United Kingdom
{A.Egri-Nagy, C.L.Nehaniv}@herts.ac.uk*

Abstract

Biochemical and genetic regulatory networks are often modeled by Petri nets. We study the algebraic structure of the computations carried out by Petri nets from the viewpoint of algebraic automata theory. Petri nets comprise a formalized graphical modelling language, often used to describe computation occurring within biochemical and genetic regulatory networks, but the semantics may be interpreted in different ways in the realm of automata. Therefore there are several different ways to turn a Petri net into a state-transition automaton. Here we systematically investigate different conversion methods and describe cases where they may yield radically different algebraic structures. We focus on the existence of group components of the corresponding transformation semigroups, as these reflect symmetries of the computation occurring within the biological system under study. Results are illustrated by applications to the Petri net modelling of intermediary metabolism. Petri nets with inhibition are shown to be computationally rich, regardless of the particular interpretation method. Along these lines we provide a mathematical argument suggesting a reason for the apparent all-pervasiveness of inhibitory connections in living systems.

1. Introduction

Petri nets, of various types, are an increasingly popular formalism for describing the operation of biochemical or genetic regulatory networks [1,2]. In order to better understand the computation occurring in biological networks so described, algebraic techniques can be used to detect hidden structure and symmetries [3–5]. The algebraic hierarchical decomposition theory of finite state automata (known as Krohn-Rhodes theory [6]) provides coordinate systems, i.e. calculational frameworks for understanding any phenomenon modelled as an automaton. The decomposition identifies reversible and irreversible components of the system and also their relationships described in a hierarchical frame. ‘Hierarchical’ is emphasized here due to the fact

that scientific models and cognition seem to utilize this kind of one-way dependency structure without feedback very often [7,8].

Our ultimate goal is to usefully apply the decomposition method to biological problems, where one usually has huge amount of experimental data related to biochemical or genetic regulatory networks, but a global understanding of the mechanisms involved is lacking. Petri nets are often used to express what is known about particular biochemical and genetic regulatory networks, or to model other systems [1], but apparently have not previously been investigated from the viewpoint of algebraic automata theory.

Before studying the coordinate systems associated to Petri nets by examining examples in detail, we have to study the relationship between au-

tomata and Petri nets. We have to check whether the conversion transfers the computational characteristics of Petri nets faithfully, and to address finiteness of state-transition automata required in basic Krohn-Rhodes theory. To accomplish this we focus on the existence of counters (algebraically these indicate the presence of nontrivial finite group components). In Krohn-Rhodes theory the complexity of the decomposition is defined by the number of alternations between strictly counting components (groups) and counter-free components in the decomposition, therefore it is important to see what properties of Petri nets lead to the existence of counters.

2. Petri Nets

Petri nets comprise a formal, graphical technique for specifying and analysing concurrent, discrete-event based dynamical systems. Petri nets are also executable models, whose syntax and semantics have been defined in many variants and with various extensions [9], so that the standardization of the Petri net technique is a work in progress.

Though there is not much variation in defining the basic Petri net, we still have to fix our notation. The well-informed reader would not find anything surprising in the definition, however the discussion of derived attributes may reveal the slightly different terminology that is needed for the forthcoming algebraic proofs.

2.1. Basic Attributes and Dynamics

Informally speaking a Petri net is a bipartite directed graph with two kinds of nodes, *places* P and *transitions* T , where there are no arcs between nodes of the same kind. More formally,

Definition 1 A Petri net \mathcal{N} is a sextuple (P, T, I, O, W, \bar{I}) , with

- P , the set of places. A place is a container of a non-negative integer number of tokens. $\#p$ (or simply p) denotes the number of tokens at place p . (graphically: a circular node)
- T , the set of transitions. A transition consumes tokens from its input places and produces tokens that are put into its output places. (graphically: a rectangular node)
- I , the set of input arcs, $I \subseteq P \times T$. These arcs define the input places of each transition.
- O , the set of output arcs, $O \subseteq T \times P$. These arcs define the output places of each transition.

- W , weights of arcs, $W : IOO \rightarrow \mathbb{N} = \{0, 1, 2, \dots\}$. These weight values determine how many tokens are consumed/produced when firing a transition involves a given input/output arc, respectively.
- \bar{I} , the set of inhibitory arcs, $\bar{I} \subseteq P \times T$. Places with nonzero tokens can inhibit transitions.

As a slight abuse of the notation we will use some attributes as functions, e.g. $I(t)$ gives the set of input arcs of transition t , $W(a)$ yields the weight of arc a , etc.

A Petri net can be embedded into a bigger system, an *environment*. For instance, thinking in chemical terms, there might be some influx, and removal of some substrates. Environmental input/output can be represented by a transition with no input/output arcs (i.e. in the case of input from the environmental, it just produces tokens without any input from places in the net, or, in the case of output to the environment, just consumes its input but produces no tokens), respectively. For an environmental input transition t , $I(t) = \emptyset$, while $O(t) = \emptyset$ for an environmental output transition.

These attributes define the syntax of the basic Petri net. With this information one can draw the corresponding diagram, representing places and transitions graphically. However, the actual dynamics, the semantics, is not yet defined.

The dynamic behaviour of the Petri net is determined by the execution of transitions. A transition t is *enabled* if $\forall i \in I(t), W(i) \leq \#(\text{source}(i))$, i.e. there are enough tokens in the input places. The transition t is *inhibited* if $\exists i \in \bar{I}(t), \#(\text{source}(i)) > 0$, i.e. there is at least one token at the source of some inhibitory arc to t . A transition is *fireable* if it is enabled and not inhibited. The intuition is that, if a transition “fires”, the input arcs of the transition instantaneously consume tokens from their sources according to the respective weights of these input arcs, while output arcs of the transition instantaneously produce tokens at their sources according to the respective weights of these arcs.

2.2. Derived Attributes

For the ease of reasoning it is useful to associate certain higher level attributes to Petri nets.

The *local state* of a place p is the number $\#p$ of tokens it contains. The *global state* of the net is a vector $m = (\#p_1, \dots, \#p_n)$ containing all local states (in an arbitrary but fixed order). The global state is usually called the *marking* of the net. A lo-

cal state transition is a change Δt_p of the number of tokens in a given place p , therefore we can associate to each transition $t \in T$ its *canonical vector of changes*, or simply (*canonical*) *change vector*, denoted by $\Delta t = (\Delta t_{p_1}, \dots, \Delta t_{p_n})$ for transition t . Each transition changes the global state in a specific way, by the overall effect of the local state transitions. It is mathematically convenient to define the weight of an arc to be zero if the arc does not occur in the I or O . Then the components of the canonical change vector of the transition t are given for each place p as

$$\Delta t_p = -W(p, t) + W(t, p).$$

The basic semantics of Petri nets are that any fireable transition may “fire” and thus change the global state according to its canonical change vector. However, in certain cases the *actual change vector* may differ from the canonical change vector, e.g. if a transition is inhibited then its actual change vector is the zero vector.

2.3. Concurrency

Petri nets and automata are both “concurrent” in the sense that any transition that is possible from a given state may occur. However, when considering Petri nets, sometimes another type of concurrency is permitted. State-transition automata are sequential, at a moment only one transformation (perhaps chosen from many possibilities) can be executed, the input symbols of an input word are fed in order into the automaton one at a time. However, in many treatments, Petri net transitions can fire simultaneously. This parallel execution in Petri nets can be recaptured in automata by using a simple powerset construction and, for Petri nets without inhibitory arcs, is inessential (see Appendix A).¹

For the sake of simplicity, here we assume that distinct transitions may not fire simultaneously.

3. Finite State Automata and Transformation Semigroups

By a finite state *automaton*, we mean a triple $\mathcal{A} = (A, X, \delta)$ where A is the finite *state set*, X is the (finite) *input alphabet* and $\delta : A \times X \rightarrow A$ is the

¹ In the case of inhibitory arcs, this type of concurrency can lead to additional computational power as shown by the example in Appendix A.

transition function. We do not explicitly consider the output of the automaton as it can be recovered from the state and the input symbol. We tacitly use the state as the output. We write X^+ for finite nonempty sequences of input letters from X , and call any such sequence w a *word*. We extend δ to give the action of words on states of A inductively defining $\delta(a, xw) = \delta(\delta(a, x), w)$ whenever w is a word and x is an input letter. One shows easily by an inductive argument that $\delta(a, uv) = \delta(\delta(a, u), v)$ holds for any words $u, v \in X^+$. We write $a \cdot w$ for $\delta(a, w)$ when $a \in A, w \in X^+$. Thus any input word w acts as a well-defined mapping from A to A , i.e. $a \mapsto a \cdot w$ for all $a \in A$. For a mapping $f : A \rightarrow A$, we say the sequence of inputs (or word) w *realizes the mapping f as an operator on states* if $\forall a \in A, f(a) = a \cdot w$.

A function $f : A \rightarrow A$ on the set A is called a *permutation (reversible)* if it is one-to-one and onto; otherwise, it must collapse elements (some $a \in A$ is an image of more than one element, i.e. the image of A under f is a proper subset of A), and therefore is *irreversible* as an operator on A . A *permutation group* is a set G of invertible mappings closed under composition, together with the state set A on which the mappings act. For more on elementary group theory see for instance [10]. A *transformation semigroup (ts)* (A, S) has a similar structure to a permutation group, but S consists of general transformations, not necessarily just permutations.

The connection between automata and ts’s is very close. The input symbols correspond to a basic set of mappings of the state set and using the usual function composition (concatenating input symbols in the automata realm) we have a ts defined on the state set. Thus the ts associated to $\mathcal{A} = (A, X, \delta)$ is (A, S) where S is the set of mappings on A that are realized by some input sequence $w \in X^+$. A *state-transition automaton* is defined just like a finite state automaton, but without the assumption of finiteness.

3.1. Counting and Non-counting

For algebraic automata theory we make a distinction between two different types of computations, between the reversible and the irreversible. However, there is a very subtle issue about how to understand reversibility. For an operator $f : A \rightarrow A$ to have an inverse, it is not enough that we can return from any given $f(a)$ to $a \in A$. There must be an inverse operator $f^{-1} : A \rightarrow A$ that undoes f which is itself a

one-to-one onto correspondence.

In the case of finite automata, if input word $w \in X^+$ realizes f as a reversible operator on states, there must be an input word $\bar{w} \in X^+$ such that $(a \cdot w) \cdot \bar{w} = a$ for all $a \in A$; i.e. \bar{w} undoes w independent of the starting state a . The existence of an inverse to a transformation f is equivalent to the property that some power of f is the identity operator. This follows from a more general fact shown in Appendix B. Thus, in this case, we may return to a previous state by going in the same direction, i.e. by repeating the same operator, but not by “magic” reset operators.²

We can illustrate this with the three automata on Fig. 1.

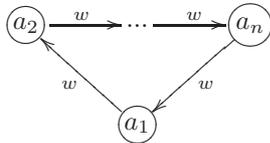
\mathcal{A}_1 is automaton with a cyclic group of order 3, $y = x^3$ is the identity. The operation x counts modulo 3.

\mathcal{A}_2 has the very same underlying graph, but we use different labelling. We have two more input symbols. It may not be immediately obvious from the diagram, but for instance the word yxz generates a cyclic group of order 2 acting on states $\{a, b\}$. So, \mathcal{A}_2 can count modulo 2.

\mathcal{A}_3 appears to be completely reversible (we can go from any state to any other state), yet it has no counter: there is no sequence of input symbols which when repeated cycles through distinct states.

It is very difficult to see from the state-transition diagram whether a given automaton is capable of counting or not, since counting may appear only when using long sequences of input symbols.

Definition 2 *An automaton \mathcal{A} is counting if there exist an input word w and a set of pairwise distinct states a_1, \dots, a_n , $n \geq 2$, such that $a_k \cdot w = a_{k+1}$ for $1 \leq k < n$ and $a_n \cdot w = a_1$. If \mathcal{A} is not counting, it is called non-counting (or counter-free or aperiodic).*



The structure $(a_1, \dots, a_n, \langle w \rangle)$ is called a *counter* or an *algebraic cycle* with generating word w and is just the cyclic permutation group on n points, where $\langle w \rangle$

² A *reset* is a constant operator $r : A \rightarrow A$ with $r(a) = a_0$ for some fixed state a_0 and every state $a \in A$. That is, r has only one state in its image. For $|A| \geq 2$, no reset is an invertible operator.

is the group generated by the transformation w , i.e. the distinct powers of w acting on the a_i 's.

The following simple argument shows that the definition captures the existence of finite group components.

Proposition 3 *An automaton \mathcal{A} is counting if and only if there is nontrivial finite permutation group (B, G) which is realized in the ts of \mathcal{A} .*

Proof: If \mathcal{A} is counting then by definition there exists at least one nontrivial finite permutation group (B, G) , given by an algebraic cycle in \mathcal{A} . Conversely, let (B, G) be a nontrivial finite permutation subgroup of a ts and let $g \in G$ be a nontrivial permutation (i.e. not the identity). Let $a \in B$ be a state which is not fixed by g , thus $a \neq a \cdot g$, so we leave the state a . Now it is possible to return to state a by repeating g by Fact 16 (Appendix B), completing an algebraic cycle, so \mathcal{A} is counting. \square

More detailed arguments (e.g. involving primitive words) connected to this concept can be found in [11].

The basic building blocks of the hierarchical decompositions of finite automata are the strict counting components in which every non-identity transformation is the generator of an algebraic cycle (permutation automata) and the flip-flop automaton, which is non-counting. Krohn-Rhodes theory shows that an automaton can be built from the direct and cascaded products of flip-flops if and only if it is non-counting – no group components appear in the decomposition [6].

4. Automata Associated to Petri Nets

There are many different ways how to convert a Petri net into an automaton, and from there canonically into a transformation semigroup. They are based on the variety of semantics that can be applied to the same net. They basically reflect different interpretations of Petri nets, therefore they do depend on what is modelled and on the modeller’s “meta-physical” standpoint. Predictions made from these various standpoints may agree or disagree in different ways with the observations of modelled phenomena, so it seems well worth knowing what the alternative viewpoints are.

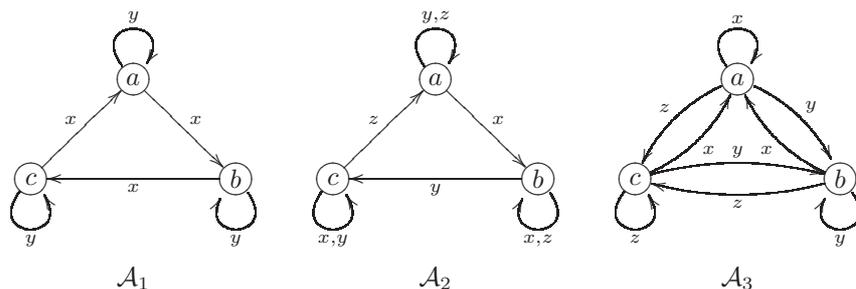


Fig. 1. Counting or non-counting? See text for details.

4.1. Representing States

Representing the states of the Petri net is fairly straightforward. Since the state set of the ts is the set of global states (i.e. markings) of the net. For practical computational applications even when considering finite capacities this can be the biggest hurdle, since the size of the state set can easily be large. One can consider some subset of this set (e.g. the set of reachable states from an initial marking) to alleviate this problem, but this is not crucial to our discussion here.

4.2. Representing Transformations

In order to construct a state transformation corresponding to a transition of the Petri net, we need to know for each state what is the resulting state when that particular transition fires. There exists a very simple algorithm to do this: we just enumerate the global states and apply the transitions for each marking and record the result of the transition. However, the transition may not be enabled/fireable in all possible global states, and this is the point when different interpretations pop up. The definition of automata forces us to have a valid resulting state for any state when applying a transformation to it.³ However, when a transition is not fireable we do not have a “real” resulting state. For the solution of this dilemma we have two options for constructing the associated automaton:

Identity Construction $\mathcal{T}_I(\mathcal{N})$. When the conditions are not satisfied nothing will happen by exe-

cuting the corresponding transformation; thus we use the identity map for that particular global state, i.e. the actual change vector is zero and the global state remains the same. This interpretation is quite natural, though – as we will see later – algebraically it has far reaching consequences.

Dead State Construction $\mathcal{T}_D(\mathcal{N})$. A transformation from a state which does not satisfy its enabling condition is not permitted, and if it used it takes the system to one particular sink state. That state can be interpreted as an ‘impossible state’ or as a ‘the experiment went wrong’ state. With this method we basically encode a partial transformation in which the undefined values are represented as a particular state, the so-called *dead state*, with no outgoing edges in the state-transition diagram.⁴ However, a transition which is enabled but inhibited results in no change of state.

We also use notations like $\mathcal{T}(\mathcal{N})$, which means that for the conversion we do not specify the actual policy.

4.3. Conversions using a Dead State \mathcal{T}_D

Theorem 4 *Let \mathcal{N} be a Petri net without inhibitory connections and $\mathcal{A} = \mathcal{T}_D(\mathcal{N})$ be the automaton resulting from the conversion using the dead state for non-enabled transitions. Then \mathcal{A} is non-counting (i.e. aperiodic).*

Proof: The dead state is a sink; whenever it is the resulting state the system remains there. Clearly, this state cannot be a member of a nontrivial counting cycle. Therefore we have to consider only the enabled transitions.

³ Note that Eilenberg [12] considers partial automata and develops Krohn-Rhodes theory for them. Use of such a framework could allow us to deal with partial automata naturally associated to Petri nets by allowing non-fireable transitions to act in an undefined way, and would be worth pursuing in future applications of Krohn-Rhodes theory to Petri net analysis.

⁴ The relation of this encoding to the nature of partial actions is subtle, since, in an emulation, different undefined computations can be emulated by variously defined and/or undefined computations. See [12].

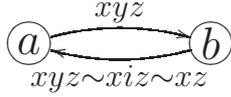


Fig. 2. Schematic example of an algebraic cycle generated by a word consisting of transitions x, y, z . The word xyz takes the global state a of the Petri net to a different global state b . The same change vector could not restore the marking a , but transition y is not fireable in the changed context, and so acts like the identity i , thus, in starting context b , the word is effectively (in terms of its actual change vector) the same as xz , which is able to restore a .

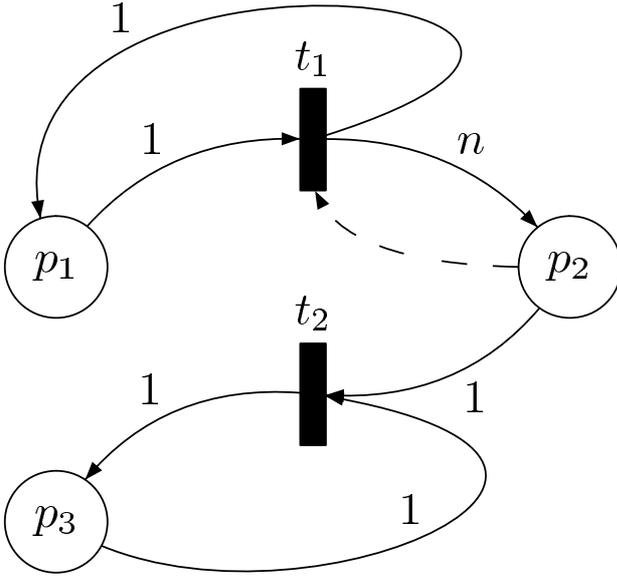


Fig. 3. An example Petri net whose associated automaton is not aperiodic. The transformation semigroup has \mathbb{Z}_n , the cyclic group on n points, as an embedded permutation group. The group is generated by the word $t_1 t_2$, where $(t_1 t_2)^{n+1}$ is the identity element of the group. Starting from the global state $(1, 0, 1)$ transition t_1 fills up the place p_2 with n tokens (arriving at $(1, n, 1)$). As p_2 is not empty anymore, t_1 is inhibited and acts as an identity (note that its input token is always available by a feedback connection back to p_1). The canonical change vector Δt_2 is $(0, -1, 0)$, which consumes the tokens of p_2 until we are back to the initial global state. The role of p_3 is rather technical; it ensures that the consumed token from p_2 is not accumulated anywhere else. As a variant, transition t_2 could have an environmental output rather than an output to place p_3 .

To prove aperiodicity, we have to show that counting cycles do not exist. According to its definition, for counting we need a starting state and a word whose powers carry it to some distinct states and eventually return it back to the starting state. A word w of the input alphabet of \mathcal{A} corresponds to a chain of transitions in the Petri net. The effect of these transitions can also be summarized as a cumu-

lative canonical change vector Δw which is the sum of the canonical change vectors of its constituent transitions. Since we may assume all transitions in the word w are fireable (see above), successive applications of the word w correspond to the repetitive additions of the same cumulative change vector. Moreover, there is no interaction between the values of the different positions of the change vector when adding it repeatedly. Therefore, if there is a nonzero value in Δw then the absolute value of that position eventually keeps growing whenever we apply w , thus we cannot get back to the original state, since we would need the zero change vector for that. On the other hand, if Δw is a zero vector we stay in the starting state, thus we have only a trivial cycle. \square

Remark. The preceding theorem may seem surprising. Myriad finite automata exist which are not aperiodic. Isn't it possible to regard any finite automaton as a Petri net without inhibitory connections? As the theorem shows, the answer is a resounding “no”. In a finite automaton, generally a single input letter will be the label of many transitions between distinct states. But this seems hard to arrange in the case of a Petri net. In a Petri net each transition corresponds to a unique label, and thus the label will generally only appear in a much more constrained way as labelling arcs in the associated automaton between *distinct* states corresponding to markings of the Petri net.

The proof gives us the intuition that in order to have counting components we need words, series of transitions that have different effects (actual change vectors different from canonical change vectors) for just some global states, i.e. in different marking contexts. We can return to the starting state (using the notation of Theorem 4) only by using different change vector at some point. But that “different change vector” can arise from the word w itself, when not all input symbols of the word can be applied (as they may correspond to transitions that are not fireable), therefore only a (scattered) subword acts “genuinely” on the state. This is illustrated in Fig. 2.

Observation 5 *There exist Petri nets with inhibition whose associated automata are counting, even under the dead-state interpretation.*

This is shown in Figure 3 which realizes a modulo n counter (for both the dead state or identity constructions). Inhibitory connections can provide a way of achieving this since in certain cases they yield the

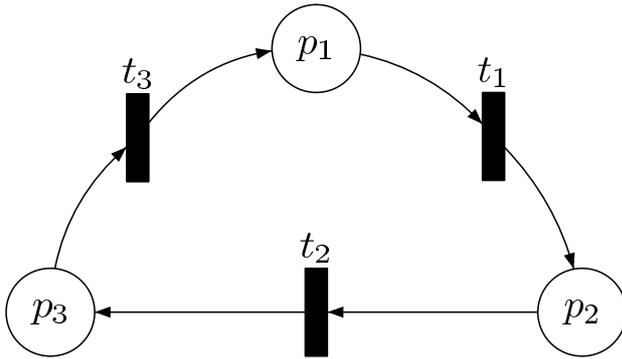


Fig. 4. A Petri net with a modulo 2 counter component in the algebraic decomposition of its associated automaton using the identity policy for non-enabled transitions. All input arcs have weight 1. The generator of the counter is $t_1 t_3 t_2$. Starting from the marking $(1, 0, 0)$ (i.e. there is only one token in p_1) by applying the generator we get the canonical change vector of $t_1 i t_2$ (where i is the identity) since t_3 is not enabled for $(0, 1, 0)$. Then again, applying the generator to the result $(0, 0, 1)$ we get the canonical change vector of $i t_3 i$ which yields $(1, 0, 0)$.

zero vector as the actual change vector for a transition, as in Fig. 3 where t_1 acts as the identity when place p_2 is nonempty.

From this example and the the previous theorem we can establish the following guiding principle:

Principle 6 For $\mathcal{T}(\mathcal{N})$ to be counting, it is necessary that at least one transition yields an actual change vector different from its canonical change vector for at least one global state in \mathcal{N} .

4.4. Conversions using Identities \mathcal{T}_I

According to Principle 6, to avoid aperiodicity we need some transitions that sometimes have actual changes different from their canonical change vector. But this is readily supplied by the interpretation of using identities for non-enabled transitions. Indeed, even without inhibitory connections, the policy of using the identity when a non-enabled transition is applied also can result in counting (see Fig. 4). Thus, **Observation 7** Under the identity construction, there are Petri nets without inhibition such that their associated automaton is counting.

It is debatable how “genuine” these group components are. Do they reflect real symmetries of the original Petri net, or they are only some by-product of a tricky interpretation?

5. Finite Capacities

In most treatments of Petri nets, the number of tokens is not limited. However, this unbounded capacity is not used in practice, and in order to apply the techniques of algebraic (finite) automata theory, we shall assume that the capacity of all places in the net is finite. This is just a technical assumption, but we have to investigate its effect on the algebraic structure. We augment the definition of the Petri net with the following entry.

– C , capacity of places, $P \mapsto \mathbb{N} \cup \{\infty\}$. This value determines how many tokens each place can accommodate.

The number of local states for a place is its capacity plus 1, since places can be empty. Using these values the number of global states of a Petri net \mathcal{N} is

$$\prod_{p \in P} (C(p) + 1).$$

Clearly, this can be a very big number even for Petri nets of small size.

By definition a transition is not fireable if there are not enough input tokens. But we also have to consider the “other end” of transitions as well – namely, the finite capacities of places. What should happen if there is not enough space for the output tokens? We have two choices:

Strict Policy. If the output places of transition cannot accommodate the resulting tokens, the transition is not enabled. This mirrors the condition of sufficient supply of input tokens.

Max Policy. The transition may be enabled regardless the content of the output places, but in case of token overflow the next local state will be the maximum capacity of the place. The extra tokens are lost. The “excess material” leaves the system, yet another way to represent the environmental output.

The choice of output condition policy can have its effect only in those global states when some of the places contain ‘enough’ tokens. If the capacities are large compared to weights of the output arcs, the difference between the effects of the policies may be negligible for markings with small numbers of tokens. However in small capacity examples, like in Fig. 5, it can result in counting.

Nevertheless, for the dead state interpretation \mathcal{T}_D , it is not hard to see that Theorem 4, i.e. aperiodicity if there are no inhibitory arcs, holds for even when some capacities are finite whether the strict or max output policy is followed.

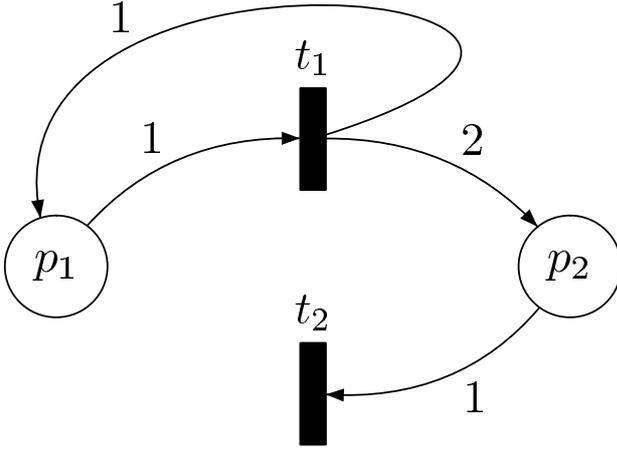


Fig. 5. An example Petri net whose associated automaton is not aperiodic when using conversion \mathcal{T}_I with strict output policy. Let the capacity of place p_2 be 2 tokens. The decomposition has \mathbb{Z}_2 , the cyclic group on 2 points (the group consists of one transposition and the identity mapping), as one of its components. The group is generated by the word $t_1 t_2$. The canonical change vector for t_1 is $(0, 2)$ and for t_2 is $(0, -1)$, but when starting from marking $(1, 0)$ the effective change vector of applying $t_1 t_2$ twice is $(0, 0) = (0, 2) + (0, -1) + (0, 0) + (0, -1)$, which equals the canonical change vector of $t_1 t_2 i t_2$. The second application of t_1 fails as there is not enough space to accommodate two more tokens and we use the strict policy. Thus $(1, 0) \cdot t_1 t_2 = (1, 1)$ and $(1, 1) \cdot t_1 t_2 = (1, 0)$. Note that, in contrast, the conversion \mathcal{T}_I done according to the max policy yields an aperiodic automaton.

Corollary 8 *Let \mathcal{N} be a Petri net with finite capacities but no inhibitory arcs. Let the $\mathcal{T}_D^f(\mathcal{N})$ the automaton derived using the strict or max policy using the dead-state construction. Then $\mathcal{T}_D^f(\mathcal{N})$ is non-counting.*

Proof: The automaton in question is a subautomaton of the automaton $\mathcal{T}_D(\mathcal{N})$ for the Petri net without finite capacity limits, except it has some modified transition arcs targeting the dead state rather than their original targets. Obviously any algebraic cycle in the resulting automaton would have to appear also in $\mathcal{T}_D(\mathcal{N})$ contradicting Theorem 4. \square

6. Completeness

From the mathematical viewpoint it is important to see what algebraic structures can arise from interpretations of Petri nets. We showed several examples of how finite cyclic groups can be constructed by Petri nets. The question is whether all finite groups can be constructed this way or not. For answering this question we need to have examples of a Petri

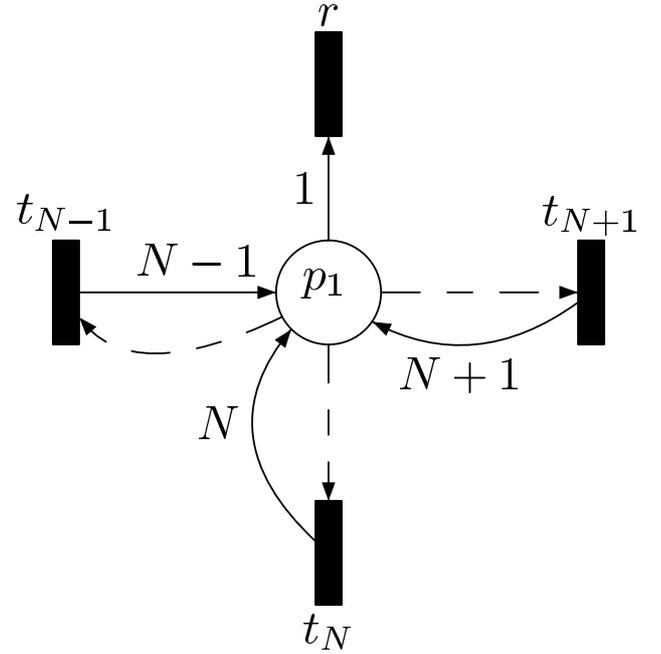


Fig. 6. The Petri net $\Gamma(N)$ with a transformation semigroup realizing all possible mappings of N points, i.e. the full transformation semigroup which contains the symmetric group as a subsemigroup.

net that yields a symmetric group, as the symmetric group permuting n points S_n contains all the groups acting by permutations on n points. We can produce such a general example using inhibitory connections (see Fig. 6 and Prop. 10(e)).

Let us consider Petri nets with one place. Let r be a transition consuming exactly one token. Let t_k be inhibited if a token is present, otherwise t_k produces k tokens.

Let $N \geq 2$. Let $\Gamma(N)$ be the Petri net with one place and with transitions r, t_1, \dots, t_{2N-1} .

Proposition 9 *Let $f : \{1, 2, \dots, N\} \rightarrow \{1, 2, \dots, N\}$ be any function. Then f is realized by the transformation semigroup of $\Gamma(N)$.*

Proof: Consider the sequence of transitions

$$\prod_{i=1}^N r t_{f(i)+(N-i)} = r t_{f(1)+(N-1)} \cdot r t_{f(2)+(N-2)} \cdots r t_{f(N-1)+1} \cdot r t_{f(N)}$$

If there are i tokens ($1 \leq i \leq N$) present then in the course of this sequence there will be i firings of r before the state with zero tokens is reached and a transition of the form t_k can fire. But then $k = f(i) + (N - i) > N - i$, so there will be $f(i) + (N - i)$ tokens after this t_k fires. After this, there will be $N - i$ more firings of r during which the remaining t_k 's (of

which there are exactly $N - i$) will remain inhibited. Clearly, the transition sequence maps each i to $f(i)$ for each $i \in \{1, \dots, N\}$.

It follows that the full transformation semigroup, and hence the symmetric group on N letters is in the semigroup of the Petri net. \square

Remark: For 0 tokens, the transition sequence above maps 0 to $f(1)$ (for the identity construction) or to the dead state (for the dead state construction); for more than N tokens it removes N tokens.

Remark: Examining the above, we can work with $\{0, 1, \dots, N - 1\}$ and f mapping that set to itself by shifting the first r -transition to the end of the sequence and increasing the number of tokens produced by a t -transition by 1. That is, the transition sequence

$$\begin{aligned} & t_{f(0)+1+(N-1)} r t_{f(1)+1+(N-2)} r \cdots \\ & \cdots t_{f(N-2)+1+1} r t_{f(N-1)+1} r \\ & = \prod_{i=0}^{N-1} t_{f(i)+N-i} r \end{aligned}$$

realizes $f : \{0, \dots, N - 1\} \rightarrow \{0, \dots, N - 1\}$.

Proposition 10 *As a mapping on states $\{1, \dots, N\}$,*

(a) *the transposition (1 2) is representable as $rt_{N+1}rt_{N-1}(rt_N)^{N-2}$.*

(b) *the elementary collapsing sending 2 tokens to 1 token is realized by $rt_Nrt_{N-1}(rt_N)^{N-2}$.*

(c) *rt_N represents the cycle $(N N - 1 \cdots 1)$.*

(d) *The full transformation semigroup on $\{1, \dots, N\}$ is generated by r, t_{N-1}, t_N, t_{N+1} .*

(e) *The Petri net $\Gamma'(N)$ with one place, with markings of 0 to $N + 1$ tokens, and the four transitions listed in (d) realizes the full transformation semigroup.*

Proof: (a) By the construction of Proposition 9, the transposition is given by

$$\begin{aligned} & rt_{2+(N-1)} r t_{1+(N-2)} r t_{3+(N-3)} \cdots \\ & \cdots r t_{i+(N-i)} \cdots r t_{N+(N-N)}, \end{aligned}$$

which simplifies to the form above. (b) is similiar. (c) is easily verified by direct calculation. (d) follows from (a), (b), and (c), and the well-known fact that an n -cycle, transposition of wo of its adjacent elements, and an elementary collapsing generate the full transformation semigroup (see, e.g., [13, Prop. 1.5]). Note $\Gamma'(N)$ is a sub-Petri net of $\Gamma(N)$. Then (e) follows from (d) and the observation that the generators map the set $\{0, \dots, N + 1\}$ to itself in the

case of the identity construction or, alternatively, for the dead-state construction, the generators map the set $\{0, \dots, N + 1\} \cup \{\text{the dead state}\}$ to itself. \square

Remark: By the second remark above, on $\{0, 1, \dots, N - 1\}$, we can represent the transposition (0 1) by $t_{N+1}rt_{N-1}r(t_Nr)^{N-2}$. The elementary collapsing taking 0 to $N - 1$ is given by t_{N-1} . Also, we observe that t_Nr represents the cycle $(N - 1 N - 2 \cdots 1 0)$. So these same generators as before, r, t_{N+1}, t_N, t_{N-1} , also yield the full transformation semigroup on $\{0, 1, \dots, N - 1\}$.

From the above propositions, remarks, and the example $\Gamma'(N)$ shown in Figure 6, we conclude that using sequences of transitions for the example (whose markings reachable from the marking with 0 tokens have 0, 1, ..., $N + 1$ tokens) that each of the subsets $\{0, 1, \dots, N - 1\}$ and $\{1, 2, \dots, N\}$ can be transformed into themselves by any function on these sets.

Theorem 11 *The symmetric group on N points and the full transformation semigroup on N points can be realized as substructures of the semigroup of the Petri net with inhibition.*

This can be seen as a mathematical indication of the all-pervasiveness of inhibitory connections in biological networks: they give the possibility for building computationally rich structures, such as the full transformation semigroups and full permutation groups. (See Discussion section below).

7. A Biological Example

Let us consider a well-known example from biochemistry, the Krebs, or citric acid cycle. Glucose is oxidized through several steps to carbon dioxide and water, while energy is stored in ATP. Here we create a Petri net model \mathcal{K} and concentrate on the role of two coenzymes NAD and CoASH (this restriction makes the model simpler). We restrict the state set to all markings of the Petri net reachable by starting from the initial marking with a single token at the place for pyruvic acid and with all other places empty. This simple technical step is computationally important, since it significantly reduces the set of states of the derived automaton. In this case the max and strict interpretations will only impact the places for coenzymes CoASH and NAD and will have no effect on the presence or absence of symme-

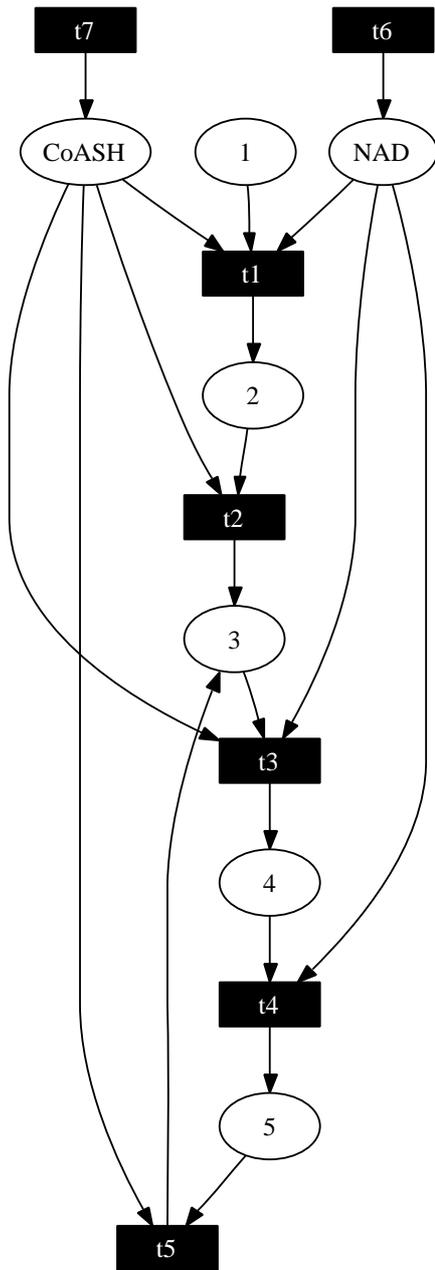


Fig. 7. A Petri net model \mathcal{K} of the Krebs cycle in respect to the coenzymes NAD and CoASH catalysing the reactions represented as transitions. The places with numbers correspond to the substrates available for the reactions. This is an extremely simplified model (we let the reactions go to completion), so one state can stand for many substrates: 1={Pyruvic acid}, 2={Acetyl CoA}, 3={Citric acid, Isocitric acid, Oxalosuccinic acid, α -Ketoglutaric acid}, 4={Succinyl CoA, Succinic acid, Fumaric acid, L-Malic acid}, 5={Oxaloacetic acid}. The fact that coenzymes are not consumed in the reactions is represented by the environmental inputs, which have the effect of keeping them available.

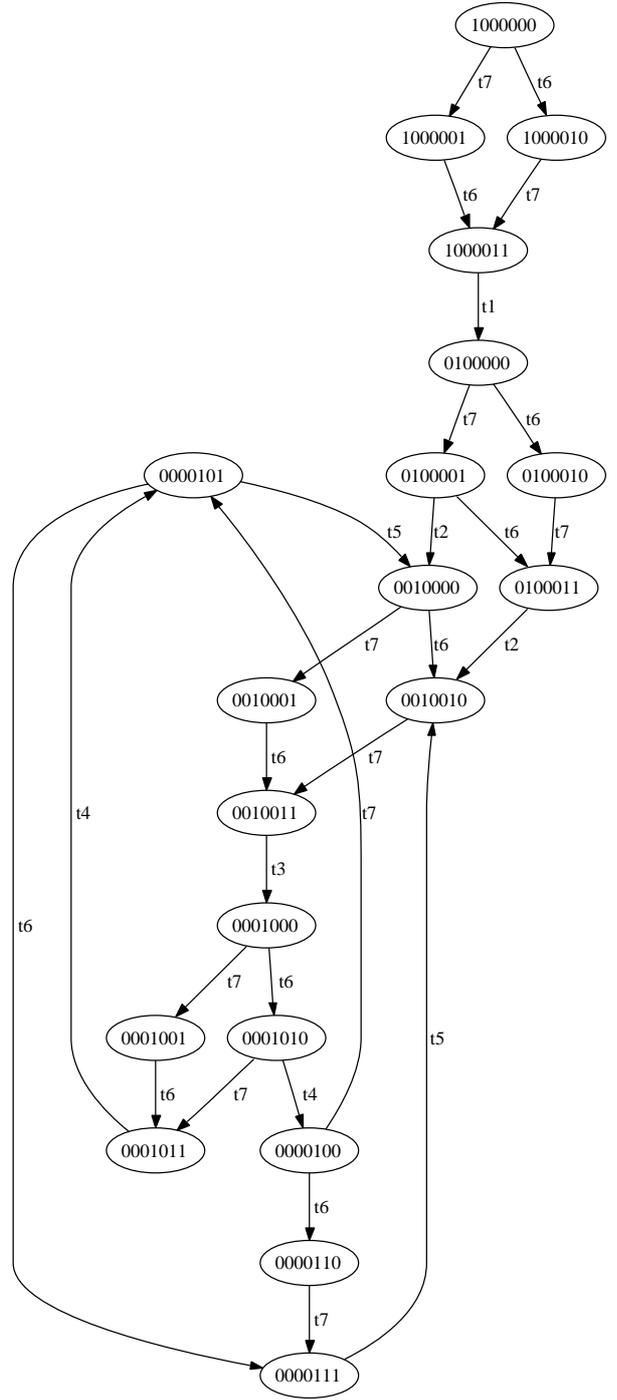


Fig. 8. The automaton derived from the Petri net model \mathcal{K} of the Krebs cycle. The states (global markings) are represented as bitstrings with the digit at position i indicating the existence (1) or absence (0) of a token at the corresponding place i in the Petri net (place 6 is NAD and place 7 is CoASH).

try groups, since amongst reachable markings exactly one substrate place will contain a token, so we shall treat the places corresponding to them as having capacity 1.

The Petri net \mathcal{K} has no inhibitory connections. The automaton derived from considering just the fireable transitions of this Petri net is shown in Figure 8. For the identity interpretation $\mathcal{T}_I(\mathcal{K})$ any transition arcs not shown act as the identity, preserving the current state; whereas for the dead state interpretation \mathcal{T}_D following such an arc leads to a sink state. By Theorem 4 proved above, under the dead state interpretation of the Petri Net model of the Krebs cycle presented here, computation is necessarily counter-free. This fact has also been verified by explicit computation using `jSgpDec` [14].

Using the identity interpretation, the effect of a non-fireable transition quite naturally represents an interaction with enzymes without the enabling cofactor(s), hence no chemical transition in the substrate is achieved. Under this interpretation, symmetry groups underlying the computation of the Krebs cycle in intermediary metabolism are revealed. Using the Petri net \mathcal{K} in Fig. 7 we found cyclic groups of order 2 and 3 among its prime components⁵, which agrees with our knowledge about the Krebs cycles from the results of a manual calculation by John Rhodes [7] using a similar automaton model of the Krebs cycle.

8. Discussion and Conclusions

Petri nets are popular in biological applications (biochemical and genetic regulatory networks), therefore it is important to explore the consequences of the actual modelling decisions. We showed that the actual interpretations, the semantics of the Petri net, can have a decisive effect on the algebraic invariants of the net. From aperiodicity we can go to counting automata by changing the interpretation of nonfireable transitions. This is a very important warning for the users of automata derived from Petri nets: the possible answers for the seemingly metaphysical question ‘What happens when a non-fireable transition tries to fire?’ have far reaching

⁵ These symmetry groups occur in multiple ways in the Krebs cycle. For instance, the global marking 0000111 and word $t_6t_7t_5t_4t_3t_6$ determine an algebraic cycle, counting cyclically through three states (0000111, 0010010, 0001010); while the word $t_4t_6t_5t_3t_6t_7$ transposes the states 0001011 and 0010011, yielding a modulo 2 counter.

consequences. Moreover, it may be rather surprising that in order to have counting components we needed transitions that in some cases behave as an identity transformation: *without effectless transitions we do not have group components.*

	Dead State Construction	Identity Construction
No Inhibition	Non-counting	Figs. 4 (and 5) (\mathbb{Z}_2)
Inhibition	Fig. 3 (\mathbb{Z}_n) and Fig. 6 (S_N)	

Table 1

Summary table for example Petri nets with counting components. Using a dead state construction for a Petri net always results in an a non-counting structure (Theorem 4). The identity construction for Petri nets without inhibition can give raise to counters regardless of policy (**strict**, **max**, or no bounds on number of tokens; see text) as in the example of Fig. 4, but the Petri net of Fig. 5 is counting only in the case of the strict policy. The Petri net with inhibition $\Gamma(N)$ in Fig. 6 generates the symmetric group S_N (and even the full transformations semigroup) regardless of interpretation (dead state construction \mathcal{T}_D or identity construction \mathcal{T}_I) and policy (**strict**, **max**, or no bounds on number of tokens). Similarly, the one in Fig. 3 with inhibition always yields a modulo n counter \mathbb{Z}_n regardless of interpretation and policy.

It remains an open question to determine whether Petri nets without inhibition could realize the full symmetric group (under the identity construction interpretation), although our examples do show the possibility of nontrivial permutation groups in this case. In contrast, from our results we can immediately extract the following conclusion for Petri nets with inhibition:

Observation 12 *With inhibition Petri nets can realize any finite permutation group or transformation semigroup. This strong result for the class of Petri nets with inhibition is completely independent of the methods of interpretation considered and of the policies for treatment of finite vs. unbounded capacities.* *Proof:* The first assertion is just the content of Theorem 11, which follows from Proposition 10(e) using the example of Petri net of Figure 6. The second part follows by observing that the steps in the proof hold for every choice of interpretation (dead state or identity construction) and policy (strict or max, using capacities of at least $N + 1$ when realizing the full transformation semigroup or the symmetric group on N points, or allowing unbounded numbers of tokens). \square

Table 1 summarizes the main results and examples constructed in the course of considering of different interpretations and policies for Petri nets.

It is notable that independent of these considerations one can find computationally rich structures in cases where inhibition plays an active role. The all-pervasive role of inhibition and negative feedback (in addition to positive activation) in yielding complex self-organized structures in biological systems is well-recognized (e.g. in stigmergy [15], pattern formation and morphogenesis [16], evolution [17], as well as genetic [18] and neural [19] regulatory control). The mathematical construction of computationally rich structures (the full transformation semigroups and full permutation groups) in Petri nets with inhibition demonstrated here could also be taken as indicative of the underlying reasons for this pervasiveness of inhibition and negative feedback in the computation of biological systems.

References

- [1] M. Peleg, D. Rubin, R. B. Altman, Using Petri Net Tools to Study Properties and Dynamics of Biological Systems, *Journal of the American Medical Informatics Association* 12 (2) (2005) 181–199.
URL <http://www.jamia.org/cgi/content/abstract/12/2/181>
- [2] W. Marwan, A. Sujatha, C. Starostzik, Reconstructing the regulatory network controlling commitment and sporulation in *Physarum polycephalum* based on hierarchical Petri Net modelling and simulation, *Journal of Theoretical Biology* 236 (2005) 349–365.
- [3] C. L. Nehaniv, J. L. Rhodes, The evolution and understanding of hierarchical complexity in biology from an algebraic perspective, *Artificial Life* 6 (1) (2000) 45–67.
- [4] C. L. Nehaniv, J. L. Rhodes, On the manner in which biological complexity may grow, in: *Mathematical and Computational Biology, Lectures on Mathematics in the Life Sciences*, Vol. 26, American Mathematical Society, 1999, pp. 93–102.
- [5] A. Egri-Nagy, C. L. Nehaniv, Hierarchical coordinate systems for understanding complexity and its evolution with applications to genetic regulatory networks, *Artificial Life* 14 (3), (Special Issue on the Evolution of Complexity), in press.
- [6] K. Krohn, J. L. Rhodes, B. R. Tilson, The prime decomposition theorem of the algebraic theory of machines, in: M. A. Arbib (Ed.), *Algebraic Theory of Machines, Languages, and Semigroups*, Academic Press, 1968, Ch. 5, pp. 81–125.
- [7] J. L. Rhodes, *Applications of Automata Theory and Algebra via the Mathematical Theory of Complexity to Biology, Physics, Psychology, Philosophy, and Games*, World Scientific Press, in press, foreword by Morris W. Hirsch, edited by Chrystopher L. Nehaniv (Original version: University of California at Berkeley, Mathematics Library, 1971).
- [8] C. L. Nehaniv, Algebra and formal models of understanding, in: M. Ito (Ed.), *Semigroups, Formal Languages and Computer Systems*, Vol. 960, Kyoto Research Institute for Mathematics Sciences, RIMS Kokyuroku, 1996, pp. 145–154.
- [9] J. Desel, G. Juhás, “What Is a Petri Net?”, in: *Unifying Petri Nets: Advances in Petri Nets*, Springer-Verlag, London, UK, 2001, pp. 1–25.
- [10] M. Hall, *The Theory of Groups*, The Macmillan Company, New York, 1959.
- [11] A. Egri-Nagy, C. L. Nehaniv, Cycle structure in automata and the homonomy decomposition, *Acta Cybernetica* 17 (2005) 199–211, [ISSN: 0324-721X].
- [12] S. Eilenberg, *Automata, Languages and Machines*, Vol. B, Academic Press, 1976.
- [13] P. Dömösi, C. L. Nehaniv, *Algebraic Theory of Finite Automata Networks: An Introduction*, SIAM Series on Discrete Mathematics and Applications, 2005.
- [14] A. Egri-Nagy, C. L. Nehaniv, SgpDec, jSgpDec – computational tools for semigroup decompositions, (<http://sgpdec.sf.net>), (formerly jGrasp) (2003).
- [15] E. Bonabeau, M. Dorigo, G. Theraulaz, *Swarm Intelligence : From Natural to Artificial Systems*, Santa Fe Institute Studies on the Sciences of Complexity, Oxford University Press, 1999.
- [16] H. Meinhardt, *Modelling Biological Pattern Formation*, Academic Press, 1982.
- [17] C. Darwin, A. Wallace, On the tendency of species to form varieties; and on the perpetuation of varieties and species by natural means of selection, *Journal of the Proceedings of the Linnean Society, Zoology* 3 (20 August 1858) 45–62.
- [18] E. H. Davidson, *The Regulatory Genome: Gene Regulatory Networks in Development and Evolution*, Academic Press, 2006.
- [19] P. Jonas, G. Buzsaki, Neural inhibition, *Scholarpedia* (2007) 21952 [last accessed 9 November 2007].
URL http://scholarpedia.org/article/Neural_Inhibition

Appendix A: On Concurrency

We discuss in this appendix what happens to the computational power of Petri nets in case simultaneous firing of multiple transitions at the same time is permitted. The appendix can be read after section 4 on automata associated to Petri nets.

Definition 13 *A concurrent transition is a non-empty set of transitions, $Q \subseteq T$, which are executed simultaneously. The transition Q is enabled if there are enough tokens for all its individual transitions (the sum of the required tokens are available):*

$$\forall p \in P, \#p \geq \sum_{t \in Q} W(p, t).$$

This definition implies that the members of Q could be executed sequentially in any order to yield the same effect as Q . There may be other definitions of concurrent transitions, but then only some particu-

lar order of sequential execution may yield the same result if any.

Definition 14 Let \mathcal{N} be a Petri net. Then $\mathcal{N}^\#$ is the Petri net made from \mathcal{N} by replacing T with $2^T \setminus \{\emptyset\}$, i.e. the set of all possible concurrent transitions, but with the restriction that only one transition may fire at a given moment.

Using this construction, we can limit the discussion to nets in which a single transition fires at a given moment. The important question is, how is \mathcal{N} , when restricted to singleton transitions, related to $\mathcal{N}^\#$? Or, asking in a different way: how much more capable is $\mathcal{N}^\#$ compared to the sequential version of \mathcal{N} ? Or, asking in a different way: can the concurrent transition be emulated by a sequence of transitions?

Theorem 15 Let \mathcal{N} be a Petri net without inhibitory connections. Then $\mathcal{T}(\mathcal{N}) = \mathcal{T}(\mathcal{N}^\#)$.

Proof: The definition of concurrent transition ensures that the execution of the individual transitions within are completely independent (the sum of the required tokens are available even if transitions share input places). A concurrent transition corresponds to a series of transformations in which the order of the execution does not matter (locally commutative). Since the transformation semigroup is built from all sequences of input symbols that yield distinct transformations, the ts contains the concurrent actions. \square

Remark: For the arguments presented in this paper it is not vital to consider the difference between \mathcal{N} and $\mathcal{N}^\#$ when there are no inhibitory arcs. But we have a different situation if the Petri net contains inhibitory connections. The simple example in Figure 9 shows that $\mathcal{T}(\mathcal{N}^\#)$ contains more transformations than $\mathcal{T}(\mathcal{N})$, i.e. $\mathcal{T}(\mathcal{N}^\#)$ may be strictly larger than $\mathcal{T}(\mathcal{N})$. Thus, with inhibition the transformation semigroup of a Petri net may in some cases become computationally more powerful by allowing simultaneous firing of sets of transitions.

Appendix B: Powers of Permutations

Here we prove a simple fact used in relating counting to the existence of permutations.

Fact 16 If $\mathcal{A} = (A, X, \delta)$ is a state-transition automaton, B is a finite subset of the state set A , and $w \in X^+$ a word of positive length in the input letters such that the mapping induced on A by w maps B to B by a nontrivial permutation. Then for some positive $n > 1$, w^n acts as the identity operator on B (so

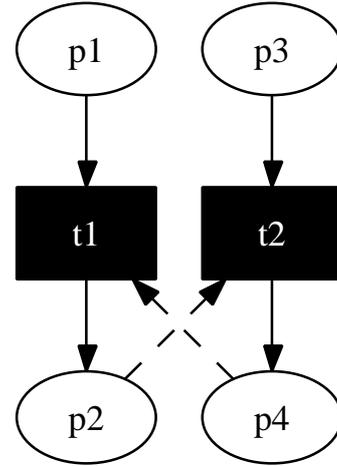


Fig. 9. An example where allowing concurrent transitions yield a bigger automaton. Starting from the marking $(1, 0, 1, 0)$ both t_1 and t_2 are fireable, even concurrently. But any sequential emulation of the concurrent transition will fail because the second transition (the order does not matter due to the mutual inhibition) will be inhibited, and thus not able to fire.

in particular w^{n-1} acts as the inverse operator to w when restricted to B . Moreover, for all $b \in B$, the state sequence $(b, b \cdot w, b \cdot w^2, b \cdot w^3, \dots)$ eventually returns to b and cycles forever.

Proof: Since B is finite there are only finitely many different mappings amongst $\langle w \rangle = \{w, w^2, w^3, \dots\}$, i.e. $w^k = w^{k+n}$ for some $k, n \geq 1$. Since w^k is a permutation on B , for each $b \in B$, there is some $b' \in B$ with $b' \cdot w^k = b$, whence $b \cdot w^n = (b' \cdot w^k) \cdot w^n = b' \cdot w^{k+n} = b' \cdot w^k = b$. That is, w^n is the identity map on B . Since w is a nontrivial permutation of B , we have $n > 1$. \square