# ABELIAN NETWORKS: FOUNDATIONS AND EXAMPLES

BENJAMIN BOND AND LIONEL LEVINE

Abstract. In Dhar's model of abelian distributed processors, finite automata occupy the vertices of a graph and communicate via the edges. A local commutativity condition ensures that the final output does not depend on the order in which the automata process their inputs. We consider the halting problem for such networks and the critical group, an invariant that governs the behavior of the network on large inputs. Our main results are 1. A finite abelian network halts on all inputs if and only if all eigenvalues of its production matrix lie in the open unit disk; 2. The critical group of an irreducible abelian network acts freely and transitively on recurrent states of the network; 3. The critical group is a quotient of a free abelian group by a subgroup containing the image of the Laplacian, with equality in the case that the network is rectangular. We also estimate the running time of an abelian network up to a constant additive error.

## Contents



*Date*: September 12, 2013.

2010 *Mathematics Subject Classification.* 68Q10, 37B15, 20M14, 20M35, 05C50,

*Key words and phrases.* abelian distributed processors, asynchronous computation, burning algorithm, chip-firing, commutative monoid action, critical group, finite automata, halting problem, Laplacian lattice, least action principle, rotor walk, sandpile group.

## 1. Introduction

In recent years, it has become clear that certain interacting particle systems studied in combinatorics and statistical physics have a common underlying structure. These systems are characterized by an *abelian property* which says changing the order of certain interactions has no effect on the final state of the system. Up to this point, the tools used to study these systems – least action principle, local-to-global principles, burning algorithm, transition monoids and critical groups – have been developed piecemeal for each particular system. Following Dhar [Dha99a], we aim to identify explicitly what these various systems have in common and exhibit them as special cases of what we call an *abelian network*.

The immediate payoff of this perspective is more general theorems with more conceptual proofs. For example, in [H+08] it was proved that the sandpile group of a graph $G$ has a free and transitive action on the set of spanning trees of $G$. In Theorem 7.9 we generalize this result to irreducible abelian networks, and the proof shows how it is really an example of a general mechanism by which group actions arise from commutative monoid actions (Theorem A.5).

A second goal of this paper is to start a philosophical discussion about non-commutativity. Just as in physics one infers from macroscopic observations the properties of microscopic particles that cannot be observed individually, we would like to be able to infer from large-scale behavior of a cellular automaton something about the local rules that generate that behavior. In particular, are there

certain large-scale features that can only be produced by *noncommutative* local interactions?

Intuition suggests that noncommutativity is a major source of dynamical richness and complexity. Yet abelian networks produce surprisingly rich and intricate large-scale patterns from local rules [Ost03, DSC09, FL12]. Moore and Nilsson [MN99] pioneered the study of computational complexity in the abelian sandpile model, and by now a lot is known in this area; see [MM11] for a recent compilation. The requirement that a distributed network produce the same output regardless of the order in which processors act would seem to place a severe restriction on the kinds of tasks it can perform. Yet abelian networks can perform some highly nontrivial tasks, such as solving certain integer programs (Corollary 4.11). Are there other computational tasks that *require* noncommutativity?

In this paper, by defining abelian networks and exploring their fundamental properties, we hope to take a step toward making these questions precise and eventually answering them. After giving the formal definition of an abelian network in §2, we survey a number of examples in §3. The main results begin in §4, where we prove a least action principle for abelian networks and explore some of its consequences. One consequence is that "local abelianness implies global abelianness" in a sense we shall make precise. In §5 we define the total kernel and production matrix of an abelian network, which are used in §6 to give conditions for a finite abelian network to halt on all inputs. Such a network has a natural invariant attached to it, the *critical group*, a finite abelian group whose structure we investigate in §7. Finally, in §8 we generalize Dhar's burning algorithm to abelian networks. Some background on monoid actions is reviewed in an appendix §A.

## 2. Definition of an abelian network

This section begins with the formal definition of an abelian network, which is based on Deepak Dhar's model of *abelian distributed processors* [Dha99a, Dha99b, Dha06]. The term "abelian network" is convenient when one wants to refer to a collection of communicating processors as a single entity. Some readers may wish to look at the examples in §3 before reading this section in detail.

Let $G = (V, E)$ be a directed graph, which may have self-loops and multiple edges. Associated to each vertex $v \in V$ is a *processor* $\mathcal{P}_v$, which is an automaton with a single input feed and multiple output feeds, one for each edge $(v, u) \in E$. Each processor reads the letters in its input feed in first-in-first-out order.

The processor $\mathcal{P}_v$ has an input alphabet $A_v$ and state space $Q_v$. Its behavior is governed by a *transition function* $T_v$ and *message passing functions* $T_{(v,u)}$ associated to each edge $(v, u) \in E$. Formally, these are maps

$$T_v : A_v \times Q_v \to Q_v \qquad \text{(new internal state)}$$
$$T_{(v,u)} : A_v \times Q_v \to A_u^* \qquad \text{(letters sent from } v \text{ to } u\text{)}$$

Here $A_u^*$ denotes the free monoid on the alphabet $A_u$. We interpret these functions as follows. If the processor $\mathcal{P}_v$ is in state $q$ and processes input $a$, then two things happen:

(1) Processor $\mathcal{P}_v$ transitions to state $T_v(a, q)$; and
(2) For each edge $(v, u) \in E$, processor $\mathcal{P}_u$ receives input $T_{(v,u)}(a, q)$.

If more than one $\mathcal{P}_v$ has inputs to process, then changing the order in which processors act may change the order of messages arriving at other processors. Concerning this issue, Dhar writes that

> "In many applications, especially in computer science, one considers such networks where the speed of the individual processors is unknown, and where the final state and outputs generated should not depend on these speeds. Then it is essential to construct protocols for processing such that the final result does not depend on the order at which messages arrive at a processor." [Dha06]

Therefore we ask that the following aspects of the computation *do not depend on the order in which individual processors act*:

(a) The **halting status** (i.e., whether or not processing eventually stops).
(b) The **final output** (final states of the processors).
(c) The **run time** (total number of letters processed by all $\mathcal{P}_v$).
(d) The **local run times** (number of letters processed by a given $\mathcal{P}_v$).
(e) The **specific local run times** (number of times a given $\mathcal{P}_v$ processes a given letter $a \in A_v$).

A priori it is not obvious that these goals are actually achievable by any nontrivial network. In §4 we will see, however, that a simple local commutativity condition ensures all five goals are achieved. To state this condition, we extend the domain of $T_v$ and $T_{(v,u)}$ to $A_v^* \times Q_v$: if $w = aw'$ is a word in alphabet $A_v$ beginning with $a$, then set $T_v(w, q) = T_v(w', T_v(a, q))$ and $T_{(v,u)}(w, q) = T_{(v,u)}(a, q)T_{(v,u)}(w', T_v(a, q))$, where the product denotes concatenation of words.

Let $\mathbb{N}^A$ be the free commutative monoid generated by $A$, and write $w \mapsto |w|$ for the natural map $A^* \to \mathbb{N}^A$, so that $|w|_a$ for $a \in A$ denotes the number of letters $a$ in the word $w$.

**Definition 2.1.** (Abelian Processor) The processor $\mathcal{P}_v$ is called *abelian* if for any words $w, w' \in A_v^*$ such that $|w| = |w'|$, we have for all $q \in Q_v$ and all edges $(v, u) \in E$

$$T_v(w, q) = T_v(w', q) \qquad \text{and} \qquad |T_{(v,u)}(w, q)| = |T_{(v,u)}(w', q)|.$$

That is, permuting the letters input to $\mathcal{P}_v$ does not change the resulting state of the processor $\mathcal{P}_v$, and may change each output word sent to $\mathcal{P}_u$ only by permuting its letters.

**Definition 2.2.** (Abelian Network) An *abelian network* on a directed graph $G = (V, E)$ is a collection of automata $\mathcal{N} = (\mathcal{P}_v)_{v \in V}$ indexed by the vertices of $G$, such that each $\mathcal{P}_v$ is abelian.

We make a few remarks about the definition:

1. The definition is *local* in the sense that it involves checking a condition on each processor individually. As we will see, these local conditions imply a "global" abelian property (Lemma 4.9).

2. A processor $\mathcal{P}_v$ is called *unary* if its alphabet $A_v$ has cardinality 1. A unary processor is trivially abelian, and any network of unary processors is an abelian network. Most of the examples of abelian networks studied so far are actually unary networks (an exception is the "block-renormalized sandpile" defined in [Dha99a]). Non-unary networks represent an interesting realm for future study. The "oil and water model" defined in §3.9 is an example of an abelian network that is not a block-renormalized unary network.

3. By enlarging the edge set $E$ with parallel edges, we could assume that at most one letter at a time is sent along any edge. By enlarging the state space $Q_v$ to remember the letter just processed by $\mathcal{P}_v$, we could remove the dependence of $T_{(u,v)}$ on $A_v$. So it is no less general to restrict to message passing functions of the form

$$T_{(u,v)} : Q_v \to A_u \cup \{\epsilon\}$$

where $\epsilon$ is the empty word.

2.1. **Comparison with cellular automata.** Cellular automata are traditionally studied on the grid $\mathbb{Z}^d$ or on other lattices, but they may be defined on any directed graph $G$. Indeed, we would like to suggest that the study of cellular automata on $G$ could be a fruitful means of revealing interesting graph-theoretic properties of $G$.

Abelian networks may be viewed as cellular automata enjoying the following two properties.

1. **Abelian networks can update asynchronously.** Traditional cellular automata update in parallel: at each time step, all cells *simultaneously* update their states based on the states of their neighbors. Since perfect simultaneity is hard to achieve in practice, the physical significance of parallel updating cellular automata is open to debate. Abelian networks do not require the kind of central control over timing needed to enforce simultaneous parallel updates, because they reach the same final state no matter in what order the updates occur.

2. **Abelian networks do not rely on shared memory.** Implicit in the update rule of cellular automata is an unspecified mechanism by which each cell is kept informed of the states of its neighbors. The lower-level interactions needed to facilitate this exchange of information in a physical implementation are absent from the model. Abelian networks include these interactions by operating in a "message passing" framework instead of the "shared memory" framework of cellular automata: An individual processor in an abelian network cannot access the states of neighboring processors, it can only read the messages they send.

## 2.2. **Summary of notation.**

| | |
|---|---|
| $G = (V, E)$ | directed graph |
| $d_v$ | outdegree of vertex $v$ |
| $\mathcal{P}_v$ | processor at vertex $v$ |
| $A_v$ | input alphabet of $\mathcal{P}_v$ |
| $Q_v$ | state space of $\mathcal{P}_v$ |
| $\mathcal{N} = (\mathcal{P}_v)_{v \in V}$ | abelian network |
| $A = \sqcup_{v \in V} A_v$ | total alphabet |
| $Q = \prod_{v \in V} Q_v$ | total state space |
| $A^*$ | the free monoid on alphabet $A$ |
| $\mathbb{N}^A$ | the commutative monoid $\prod_{a \in A} \mathbb{N}$ |
| $\mathbb{Z}^A$ | the abelian group $\prod_{a \in A} \mathbb{Z}$ |
| $\mathbb{Q}^A$ | the vector space $\prod_{a \in A} \mathbb{Q}$ |
| $1_a$ | basis elements of $\mathbb{N}^A, \mathbb{Z}^A, \mathbb{Q}^A$ |
| $\lvert w \rvert_a$ | number of letters $a$ in word $w$ |
| $T_v$ | transition function of vertex $v$ |
| $T_{(v,u)}$ | message passing function of edge $(v, u)$ |
| $t_a$ | for $a \in A_v$, the map $Q_v \to Q_v$ sending $q \mapsto T_v(a, q)$ |
| $t_v(\mathbf{x})$ | for $\mathbf{x} \in \mathbb{N}^{A_v}$, the composition $\prod_{a \in A_v} t_a^{\mathbf{x}_a}$ |
| $\mathbf{x}_v$ | for $\mathbf{x} \in \mathbb{N}^A$, the image of $\mathbf{x}$ under the projection $\mathbb{N}^A \to \mathbb{N}^{A_v}$ |
| $t(\mathbf{x})$ | for $\mathbf{x} \in \mathbb{N}^A$, the map $Q \to Q$ sending $\mathbf{q} \mapsto (t_v(\mathbf{x}_v)\mathbf{q}_v)_{v \in V}$ |
| $M_v$ | transition monoid of $\mathcal{P}_v$, generated by $\{t_a\}_{a \in A_v}$ |
| $e_v$ | minimal idempotent of $M_v$ |
| $K_v$ | kernel of the action $\mathbb{Z}^{A_v} \times e_v Q_v \to e_v Q_v$ |
| $K = \prod_{v \in V} K_v$ | total kernel (Definition 5.4) |
| $r_a$ | smallest positive integer $k$ such that $k1_a \in K$ |
| $P$ | production matrix (Definition 5.7) |
| $D$ | the diagonal matrix $D_{aa} = r_a$ |
| $L$ | Laplacian, $L = (I - P)D$ |
| $\mathbf{x}.\mathbf{q}$ | for $\mathbf{x} \in \mathbb{N}^A$, $\mathbf{q} \in Q$: processors are in state $\mathbf{q}$, with $\mathbf{x}_a$ letters $a$ present for each $a \in A$. |
| $\mathbf{x} \triangleright \mathbf{y}.\mathbf{q}$ | the result $\mathbf{y}'.\mathbf{q}'$ of processing each letter in $\mathbf{x}$ once, starting from $(\mathbf{x} + \mathbf{y}).\mathbf{q}$. |
| $\mathcal{N}_\mathbf{q}$ | local component of $\mathcal{N}$ containing state $\mathbf{q}$ |
| $\mathbf{x} \triangleright\triangleright \mathbf{q}$ | the result $\mathbf{q}'$ of processing all letters in $\mathbf{x}.\mathbf{q}$ until halting |
| $[\mathbf{x}.\mathbf{q}]$ | the odometer: $[\mathbf{x}.\mathbf{q}]_a$ is the number of letters $a$ processed. |
| $\tau(\mathbf{x})$ | for $\mathbf{x} \in \mathbb{N}^A$, the map $\mathbf{q} \mapsto \mathbf{x} \triangleright\triangleright \mathbf{q}$ |
| $M$ | transition monoid of $\mathcal{N}$, generated by $\{\tau(1_a)\}_{a \in A}$ |
| $e$ | minimal idempotent of $M$ |
| $\operatorname{Crit} \mathcal{N}$ | critical group ($= eM$) |
| $\operatorname{Rec} \mathcal{N}$ | the set of recurrent states ($= eQ$) |
| $\texttt{Sand}(G, s)$ | sandpile network with sink vertex $s$ |
| $\texttt{Topp}(L)$ | toppling network with Laplacian matrix $L$ |
| $\texttt{Rotor}(G, s)$ | rotor network with sink vertex $s$ |

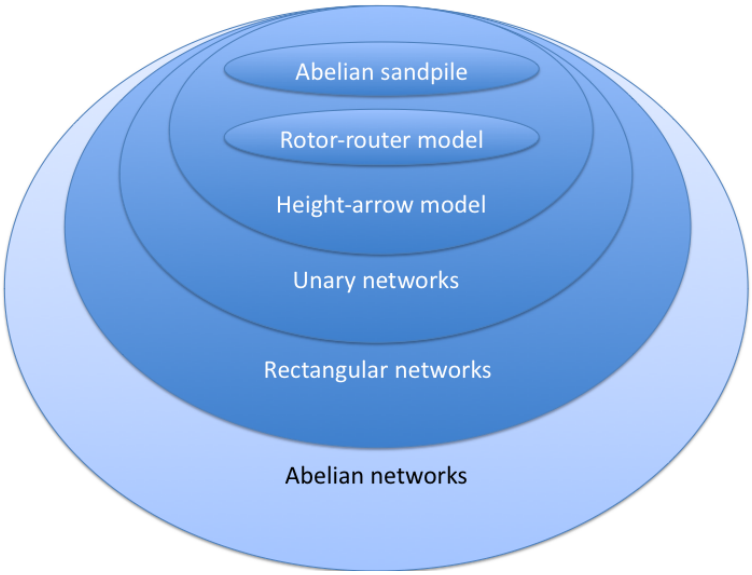


FIGURE 1. Venn diagram illustrating several classes of abelian networks.

## 3. EXAMPLES

3.1. **Sandpile networks.** Figure 1 shows increasingly general classes of abelian networks. The oldest and most studied is the *abelian sandpile model* [BTW87, Dha90], also called *chip-firing* [BLS91, Big99]. Given a directed graph $G = (V, E)$, the processor at each vertex $v \in V$ has input alphabet $A_v = \{v\}$ and state space $Q_v = \{0, 1, \ldots, r_v - 1\}$, where $r_v$ is the outdegree of $v$. The transition function is

$$T_v(q) = q + 1 \pmod{r_v}.$$

(Formally we should write $T_v(v, q)$, but when $\#A_v = 1$ we omit the redundant first argument.) The message passing functions are

$$T_{(v,u)}(q) = \begin{cases} \epsilon, & q < r_v - 1 \\ u, & q = r_v - 1 \end{cases}$$

for each edge $(v, u) \in E$. Here $\epsilon \in A^*$ denotes the empty word. Thus each time the processor at vertex $v$ transitions from state $r_v - 1$ to state 0, it sends one letter to each of its out-neighbors (Figure 2). When this happens we say that vertex $v$ *topples* (or "fires").

3.2. **Toppling networks.** These have the same transition and message passing functions as the sandpile networks above, but we allow the number of states $r_v$ (called the *threshold* of vertex $v$) to be different from the outdegree of $v$. These networks can be concretely realized in terms of "chips": If a vertex in state $q$ has $k$ letters in its input feed, then we say that there are $q + k$ chips at that vertex. When $v$ has at least $r_v$ chips, it can *topple*, losing $r_v$ chips and sending one chip along each outgoing edge. In a sandpile network the total number of chips is

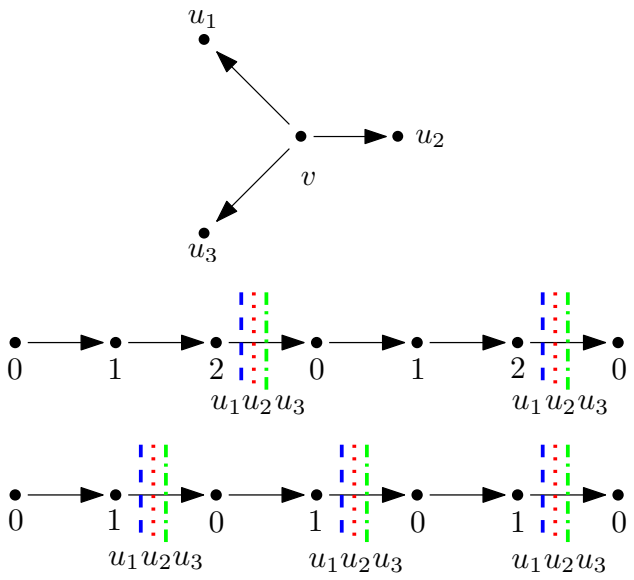


FIGURE 2. Top: portion of graph $G$ showing a vertex $v$ and its outneighbors $u_1, u_2, u_3$. Middle: State diagram for $v$ in a sandpile network. Dots represent states, arrows represent transitions when a letter is processed, and dashed vertical lines indicate when letters are sent to the neighbors. Bottom: State diagram for $v$ in a toppling network with $r_v = 2$.

conserved, but in a toppling network, chips may be created (if $r_v$ is less than the outdegree of $v$, as in the last diagram of Figure 2) or destroyed (if $r_v$ is larger than the outdegree of $v$).

Note that some chips are "latent" in the sense that they are encoded by the internal states of the processors. For example if a vertex $v$ with $r_v = 2$ is in state 0, receives one chip and processes it, then the letter representing that chip is gone, but the internal state increases to 1 representing a latent chip at $v$. If $v$ receives another chip and processes it, then its state returns to 0 and it topples by sending one letter to each out-neighbor.

It is convenient to specify a toppling network by its *Laplacian*, which is the $V \times V$ matrix $L$ with diagonal entries $L_{vv} = r_v - d_{vv}$ and off-diagonal entries $L_{uv} = -d_{uv}$. Here $d_{uv}$ is the number of edges from $v$ to $u$ in the graph $G$. The toppling network with no loops ($d_{vv} = 0$) and Laplacian $L$ will be denoted $\texttt{Topp}(L)$.

Sometimes it is useful to consider toppling networks where the number of chips at a vertex may become negative. We can model this by enlarging the state space of each processor to include $-\mathbb{N}$; these additional states have transition function $T_v(q) = q+1$ and send no messages. In §4 we will see that these enlarged toppling networks solve certain integer programs.

3.3. **Sinks and counters.** It is common to consider the sandpile network $\texttt{Sand}(G, s)$ with a *sink* $s$, a vertex whose processor has only one state and never sends any

messages. If every vertex of $G$ has a directed path to the sink, then any finite input to $\texttt{Sand}(G, s)$ will produce only finitely many topplings. A natural question is, when does the same finiteness hold in a general toppling network – which may have a mix of creative and destructive vertices, but perhaps no sink? This is a case of the halting problem for abelian networks, treated in §6.

The *graph Laplacian* $\Delta$ of $G$ has diagonal entries $\Delta_{vv} = d_v - d_{vv}$ and off-diagonal entries $\Delta_{uv} = -d_{uv}$. We have $\texttt{Sand}(G, s) = \texttt{Topp}(\Delta^s)$, where the *damped Laplacian* $\Delta^s$ is obtained from $\Delta$ by setting $\Delta_{ss} = 1$ and $\Delta_{vs} = 0$ for $v \neq s$.

The set of *recurrent states* (defined in §7) of a sandpile network with sink is in bijection with objects of interest in combinatorics such as oriented spanning trees and $G$-parking functions [PS04].

A *counter* is a unary processor with state space $\mathbb{N}$ and transition $T(q) = q + 1$, which never sends any messages. It behaves like a sink, but keeps track of how many letters it has received.

3.4. **Bootstrap percolation.** In this simple model of crack formation, each vertex $v$ has a threshold $b_v$. Vertex $v$ becomes "infected" as soon as at least $b_v$ of its in-neighbors are infected. Infected vertices remain infected forever. A question that has received a lot of attention [Ent87, Hol03] due to its subtle scaling behavior is: What is the probability the entire graph becomes infected, if each vertex independently starts infected with probability $p$? To realize bootstrap percolation as an abelian network, we take $A_v = \{v\}$ and $Q_v = \{0, 1, \ldots, b_v\}$, with $T_v(q) = \min(q + 1, b_v)$ and

$$T_{(v,u)}(q) = \begin{cases} u, & q = b_v - 1 \\ \epsilon, & q \neq b_v - 1. \end{cases}$$

The internal state $q$ of the processor $\mathcal{P}_v$ keeps track of how many in-neighbors of $v$ are infected. When this count reaches the threshold, $\mathcal{P}_v$ sends a letter to each out-neighbor of $v$ informing them that $v$ is now infected.

3.5. **Rotor networks.** A *rotor* is a unary processor $\mathcal{P}_v$ that outputs exactly one letter for each letter input. That is, for all $q \in Q_v$

$$\sum_{(v,u)\in E} \sum_{a \in A_u} |T_{(v,u)}(q)|_a = 1. \tag{1}$$

By inputting a single letter at vertex $v_0$ into a network of rotors with underlying graph $G$, we obtain in a natural way an infinite walk $v_0, v_1, \ldots$ in $G$. Vertex $v_n$ is the location of the single letter present after $n$ processings. This *rotor walk* has been studied under various names: In computer science it was introduced as a model of autonomous agents exploring a territory ("ant walk," [WLB96]) and later studied as a means of broadcasting information through a network [DFS08]. In statistical physics it was proposed as a model of self-organized criticality ("Eulerian walkers," [PDDK96]). Propp proposed rotor walk as a way of derandomizing certain features of random walk [Pro03, CS06, HP10, Pro10].

Most commonly studied are the *simple* rotor networks on a directed graph $G$, in which the out-neighbors of vertex $v$ are served repeatedly in a fixed order

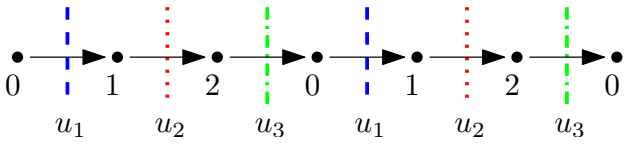


FIGURE 3. State diagram for a vertex $v$ in a simple rotor network. The out-neighbors $u_1, u_2, u_3$ of $v$ are served repeatedly in a fixed order.

$u_1, \dots, u_{d_v}$ (Figure 3). Formally, we set $Q_v = \{0, 1, \dots, d_v - 1\}$, with transition function $T_v(q) = q + 1 \pmod{d_v}$ and message passing functions

$$T_{(v,u_j)}(q) = \begin{cases} u_j, & q \equiv j - 1 \pmod{d_v} \\ \epsilon, & q \not\equiv j - 1 \pmod{d_v}. \end{cases}$$

**Prospective convention.** Here we should comment on an implicit choice in our definition of abelian network: In interpreting the message passing functions, we have chosen the *prospective* convention: the state of a vertex indicates what it will do next. In the alternative *retrospective* convention, one replaces the state $q$ in item (2) of §2 with $T_v(a, q)$, so that the state of a vertex indicates what it has just done.

A state of a simple rotor network $\texttt{Rotor}(G, s)$ with sink vertex $s$ corresponds to a choice of one outgoing edge from each vertex in $V - \{s\}$ (indicating where the next letter will be sent, if we are using the prospective convention, or where the last letter was sent if we are using the retrospective convention). It turns out that in the retrospective convention, the recurrent states correspond to choices in which these edges form a spanning tree of $G$ oriented toward the sink. In particular, the recurrent rotor states are equinumerous with the recurrent sandpile states on the same graph. This curious fact has inspired several different bijections between recurrent sandpile states and spanning trees [BW97, CL03, CP05]. Recurrent sandpile states carry a natural group structure (about which we say more in §7) whereas spanning trees do not. However, interpreting a spanning tree as a state of a rotor network allows one to construct a free transitive action of the sandpile group on spanning trees [H+08], which "explains" why these sets are equinumerous. In Theorem 7.9 we show that this group action is a general phenomenon for irreducible abelian networks.

The temptation to identify the recurrent states of $\texttt{Rotor}(G, s)$ with oriented spanning trees has been the main reason for using the retrospective convention in past works such as [H+08]. Even for rotor networks, however, there is a sense in which the prospective convention is more natural. For an edge $e = (v, u_i)$, write $e^+ = (v, u_{i+1 \bmod d_v})$. A *progressed spanning tree* is a spanning subgraph of $G$ with edge set $\{e^+ : e \in T\}$ where $T$ is the edge set of a spanning tree oriented toward $s$. In the prospective convention, the recurrent states of $\texttt{Rotor}(G, s)$ are the progressed spanning trees. This serves to remind us that there are many simple

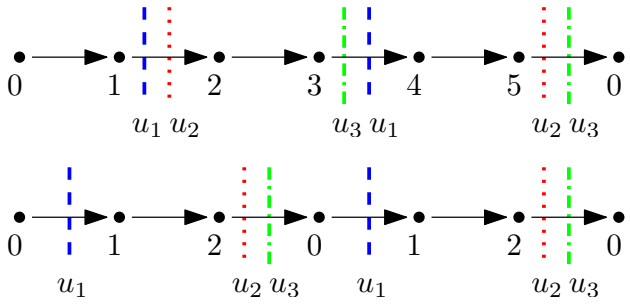


FIGURE 4. Example state diagrams for a vertex $v$ in the Dartois-Rossin height arrow model with $d_v = 3$ and $\tau_v = 2$ (top) and Eriksson's periodically mutating game (bottom).

rotor networks: $\texttt{Rotor}(G, s)$ depends not only on $G$ and $s$ but also on the choice of orderings $u_1, \ldots, u_{d_v}$.

**Rotor-router aggregation.** Enlarge each state space $Q_v$ of a simple rotor network to include a transient state $-1$, which transitions to state 0 but passes no message. Starting with all processors in state $-1$, the effect is that each vertex "absorbs" the first letter it receives, and behaves like a rotor thereafter. If we input $n$ letters to one vertex $v_0$, then each letter performs a rotor walk starting from $v_0$ until reaching a site that has not yet been visited by any previous walk, where it gets absorbed. Propp [Pro03] proposed this model as a way of derandomizing a certain random growth process (internal DLA). When the underlying graph is $\mathbb{Z}^2$, the resulting set of $n$ visited sites is very close to circular [LP09], and the final states of the processors display intricate patterns that are still not well understood [FL12].

3.6. **Height arrow model.** Dartois and Rossin [DR04] proposed a common generalization of rotor and sandpile networks called the *height-arrow model*. An example state diagram is shown in Figure 4. For each vertex $v$ we set $A_v = \{v\}$ and choose a threshold $1 \leq \tau_v \leq d_v$, where $d_v$ is the outdegree of $v$. We set $Q_v = (\mathbb{Z}/d_v\mathbb{Z}) \times (\mathbb{Z}/\tau_v\mathbb{Z})$ with transition function

$$T_v(q, c) = \begin{cases} (q, c+1) & c < \tau_v - 1 \\ (q + \tau_v \pmod{d_v}, 0), & c = \tau_v - 1. \end{cases}$$

Fix an ordering $u_0, \ldots, u_{d_v-1}$ of the out-neighbors of $v$. The message passing function for the edge $(v, u_j)$ is given by

$$T_{(v,u_j)}(q, c) = \begin{cases} u_j & \text{if } c = \tau_v - 1 \text{ and } j \in \{q + j \pmod{d_v}\}_{j=0}^{\tau_v - 1} \\ \epsilon & \text{else.} \end{cases}$$

We think of the state $(q, c)$ as representing an arrow pointing to the neighbor $u_q$ and a number of chips $c$. When vertex $v$ collects $\tau_v$ chips, it increments the arrow $\tau_v$ times and distributes one chip to each neighbor $u_{q+j \pmod{d_v}}$ for $j = 0, 1, \ldots, \tau_v - 1$.

3.7. **Unary networks.** Diaconis and Fulton [DF91] and Eriksson [Eri96] studied generalizations of chip-firing in which each vertex has a stack of instructions. When a vertex accumulates enough chips to follow the top instruction in its stack, it pops that instruction off the stack and follows it. These and all preceding examples are *unary networks*, that is, abelian networks in which each alphabet $A_v$ has cardinality 1. Informally, a unary network on a graph $G$ is a system of local rules by which *indistinguishable* chips move around on the vertices of $G$.

Next we discuss two non-unary examples.

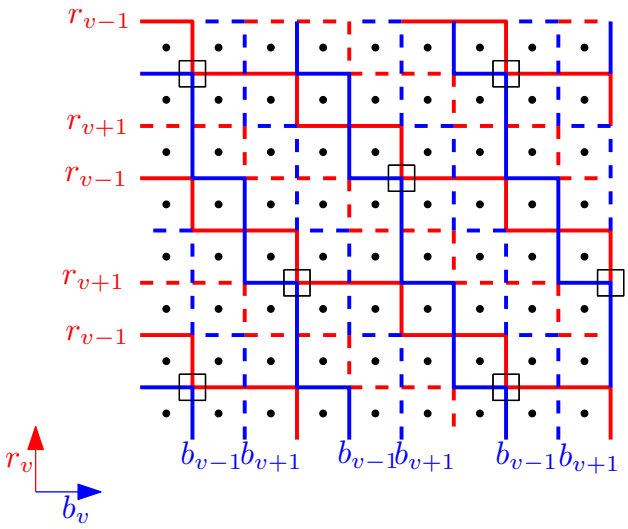


FIGURE 5. Abelian mobile agents: Example state diagram for a processor $\mathcal{P}_v$ in a network whose underlying graph is a path or a cycle. When an agent arrives at vertex $v$, she transitions the state of $v$ (upward if her own state is red, rightward if blue); updates her own state to red or blue according to the color of the line crossed; and moves to $v-1$ or $v+1$ according to whether the line is solid or dashed. In this example $\#Q_v = 12$, and the boxes indicate periodicity in the diagram.

3.8. **Abelian mobile agents.** In the spirit of [WLB96], one could replace the messages in our definition of abelian networks by mobile agents each of which is an automaton. As a function of its own internal state $a$ and the state $q$ of the vertex $v$ it currently occupies, an agent acts by doing three things:

(1) it changes its own state to $S_v(a, q)$; and
(2) it changes the state of $v$ to $T_v(a, q)$; and
(3) it moves to a neighboring vertex $U_v(a, q)$.

Two or more agents may occupy the same vertex, in which case we require that the outcome of their actions is the same regardless of the order in which they act.

For purposes of deciding whether two outcomes are the same, we regard agents with the same internal state and location as indistinguishable.

This model may appear to lie outside our framework of abelian networks, because the computation is located in the moving agents (who carry their internal states with them) instead of in the static processors. However, a moment's thought shows that it has identical behavior to the abelian network with transition function $T_v$ and message passing function

$$T_{(v,u)}(a,q) = \begin{cases} S_v(a,q) & \text{if } u = U_v(a,q) \\ \epsilon & \text{else.} \end{cases}$$

Abelian mobile agents generalize the rotor networks (§3.5) by dropping the requirement that processors be unary. The defining property of abelian mobile agents is that each processor sends exactly one letter for each letter received. In Figure 5 this property is apparent from the fact that each segment of the square grid lies on exactly one message line.

3.9. **Oil and water model.** This is a non-unary generalization of sandpiles, inspired by Paul Tseng's asynchronous algorithm for solving certain linear programs [Tse90]. Each edge of $G$ is marked either as an oil edge or a water edge. When a vertex topples, it sends out one oil chip along each outgoing oil edge and also one water chip along each outgoing water edge. The interaction between oil and water is that a vertex is permitted to topple if and only if sufficiently many chips of *both* types are present at that vertex.

Unlike most of the preceding examples, oil and water can not be realized with a finite state space $Q_v$, because an arbitrary number of oil chips could accumulate at $v$ and be unable to topple if no water chips are present. We set $Q_v = \mathbb{N} \times \mathbb{N}$ and $A_v = \{o_v, w_v\}$, with transition function

$$T_v(o_v, q) = q + (0,1), \qquad T_v(w_v, q) = q + (1,0).$$

The internal state of the processor at $v$ is a vector $q = (q_{oil}, q_{water})$ keeping track of the total number chips of each type it has received (Figure 6).

3.10. **Stochastic abelian networks.** In a stochastic abelian network, we allow the transition functions to depend on a probability space $\Omega$:

$$\begin{aligned} T_v &: A_v \times Q_v \times \Omega \to Q_v && \text{(new internal state)} \\ T_{(v,u)} &: A_v \times Q_v \to A_u^* && \text{(letters sent from } v \text{ to } u\text{)} \end{aligned}$$

A variety of models in statistical mechanics — including classical Markov chains and branching processes, branching random walk, certain directed edge-reinforced walks, internal DLA [DF91], the Oslo model [Fre93], the abelian Manna model [Dha99c], excited walk [BW03], the Kesten-Sidoravicius infection model [KS05, KS08], two-component sandpiles and related models derived from abelian algebras [AR08, APR09], activated random walkers [DRS10], stochastic sandpiles [RS11], and low-discrepancy random stack [FL12] — can all be realized as stochastic abelian networks. In at least one case [RS11] the abelian nature of the

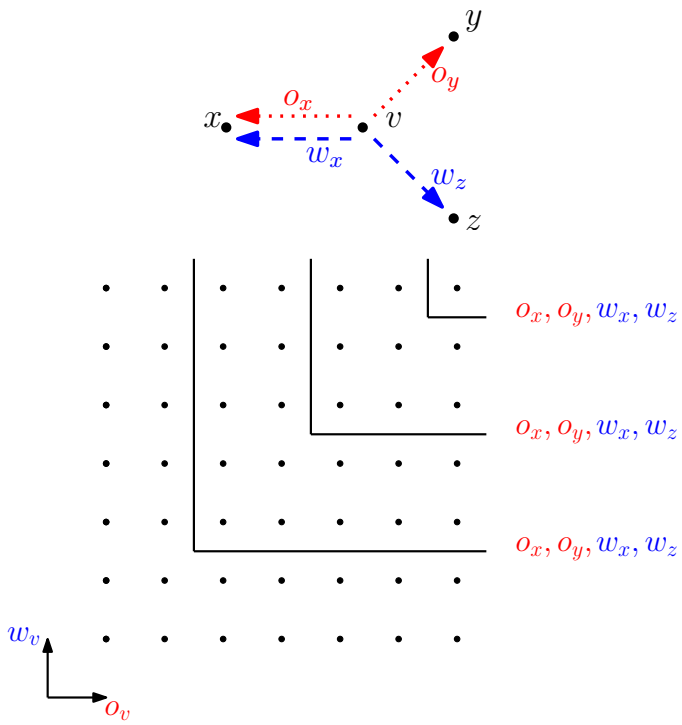


FIGURE 6. Example state diagram for the oil and water model. Top: Vertex $v$ has outgoing oil edges to $x$ and $y$, and water edges to $x$ and $z$. Bottom: each dot represents a state in $Q_v = \mathbb{N} \times \mathbb{N}$, with the origin at lower left. A toppling occurs each time the state transition crosses one of the bent lines (for example, by processing an oil $o_v$ in state $(1, 2)$, resulting in transition to state $(2, 2)$).

model enabled a major breakthrough in proving the existence of a phase transition. Stochastic abelian networks are beyond the scope of the present paper and will be treated in a sequel.

## 4. Least action principle

Our first aim is to prove a least action principle for abelian networks, Lemma 4.5. This principle says — in a sense to be made precise — that each processor in an abelian network performs the minimum amount of work possible to remove all messages from the network. Various special cases of the least action principle to particular abelian networks have enabled a flurry of recent progress: bounds on the growth rate of sandpiles [FLP10], exact shape theorems for rotor aggregation [KL10, HS11], proof of a phase transition for activated random walkers [RS11], and a fast simulation algorithm for growth models [FL12].

The proof of the least action principle follows Diaconis and Fulton [DF91, Theorem 4.1]. Our observation is that their proof actually shows something more general: it applies to any abelian network. Moreover, as noted in [FLP10, RS11],

the proof applies even to executions that are complete but not legal. To explain the last point requires a few definitions.

Let $\mathcal{N}$ be an abelian network with underlying graph $G = (V, E)$, total state space $Q = \prod Q_v$ and total alphabet $A = \sqcup A_v$. In this section we do not place any finiteness restrictions on $\mathcal{N}$: the underlying graph may be finite or infinite, and the state space $Q_v$ and alphabet $A_v$ of each processor may be finite or infinite.

For each $v \in V$ and $a \in A_v$, denote by $t_a : Q_v \to Q_v$ the map $t_a(q) = T_v(a, q)$.

**Definition 4.1.** The *local monoid* at vertex $v$ is the submonoid $M_v \subset \mathrm{End}\,(Q_v)$ generated by $\{t_a\}_{a \in A_v}$, where $\mathrm{End}\,(Q_v)$ denotes the monoid of all set maps $Q_v \to Q_v$ with the operation of composition.

By the definition of an abelian processor, $t_a t_b = t_b t_a$ for all $a, b \in A_v$, so $M_v$ is a commutative monoid. The direct product of local monoids $\prod_{v \in V} M_v$ acts on $Q$ coordinatewise: given $\mathbf{m} = (m_v)_{v \in V}$ and $\mathbf{q} = (q_v)_{v \in V}$ we define $\mathbf{mq} = (m_v q_v)_{v \in V}$. Note that this action depends only on the transition functions $T_v$, and not on the message passing functions $T_e$.

To involve the message passing functions, it is convenient to view the entire network $\mathcal{N}$ as a single automaton with alphabet $A$ and state space $\mathbb{Z}^A \times Q$. For its states we will use the notation $\mathbf{x}.\mathbf{q}$, where $\mathbf{x} \in \mathbb{Z}^A$ and $\mathbf{q} \in Q$. If $\mathbf{x} \in \mathbb{N}^A$ the state $\mathbf{x}.\mathbf{q}$ corresponds to the configuration of the network $\mathcal{N}$ such that

- For each $a \in A$, there are $\mathbf{x}_a$ letters of type $a$ present; and
- For each $v \in V$, the processor at vertex $v$ is in state $q_v$.

Allowing $\mathbf{x}$ to have negative coordinates is a useful device that enables the least action principle (Lemma 4.5 below). Formally, $\mathbf{x}.\mathbf{q}$ is just an alternative notation for the ordered pair $(\mathbf{x}, \mathbf{q})$. The decimal point in $\mathbf{x}.\mathbf{q}$ is intended to evoke the intuition that the internal states $\mathbf{q}$ of the processors represent latent "fractional" messages.

Note that $\mathbf{x}.\mathbf{q}$ indicates only the states of the processors and the *number* of letters present of each type. It gives no information about the order in which letters are to be processed. Indeed, one of our goals is to show that the order does not matter (Theorem 4.8).

For any $v \in V$ and $a \in A_v$ we define the state transition $\pi_a : \mathbb{Z}^A \times Q \to \mathbb{Z}^A \times Q$ by

$$\pi_a(\mathbf{x}.\mathbf{q}) = \mathbf{x}'.\mathbf{q}'$$

where $\mathbf{q}' \in Q$ and $\mathbf{x}' \in \mathbb{Z}^A$ are obtained as follows. Let $\mathbf{q}' = t_a(\mathbf{q})$, where we have extended the transition map $t_a : Q_v \to Q_v$ to a map $Q \to Q$ by setting $q'_v = t_a(q_v)$ and $q'_u = q_u$ for all $u \neq v$. Let

$$\mathbf{x}' = \mathbf{x} - 1_a + \mathbf{N}(a, q_v)$$

where $\mathbf{N}(a, q_v)_b$ is the number of $b$'s produced when processor $\mathcal{P}_v$ in state $q_v$ processes the letter $a$. In other words,

$$\mathbf{N}(a, q_v) = \sum_e |T_e(a, q_v)|$$

where the sum is over all outgoing edges $e$ from $v$ (both sides are vectors in $\mathbb{Z}^A$).

An *execution* is a word $w = w_1 \cdots w_r \in A^*$. It prescribes an order in which letters in the network are to be processed. For simplicity, we consider only finite executions ($r < \infty$) in the present paper, but we remark that infinite executions (and non-sequential execution procedures) are also of interest [FMR09]. For an execution $w = w_1 \cdots w_r \in A^*$, define $\pi_w$ to be the composition $\pi_{w_r} \circ \cdots \circ \pi_{w_1}$.

Fix an initial state $\mathbf{x}.\mathbf{q}$ and an execution $w$. Set $\mathbf{x}^0 = \mathbf{x}$, $\mathbf{q}^0 = \mathbf{q}$ and

$$\mathbf{x}^i.\mathbf{q}^i = \pi_{w_1 \cdots w_i}(\mathbf{x}.\mathbf{q}), \qquad i = 1, \ldots, r.$$

The *result* of executing $w$ is $\pi_w(\mathbf{x}.\mathbf{q}) = \mathbf{x}^r.\mathbf{q}^r$. Our goal is to compare the results of different executions.

For each $i = 1, \ldots, r$ we have $w_i \in A_{v(i)}$ for some vertex $v(i)$. Let

$$\mathbf{N}(w, \mathbf{q}) := \sum_{i=1}^{r} \mathbf{N}(w_i, q_{v(i)}^{i-1})$$

so that $\mathbf{N}(w, \mathbf{q})_b$ is the total number of $b$'s produced by executing $w$ starting from state $\mathbf{q}$.

Recall that $|w| \in \mathbb{N}^A$ and $|w|_a$ is the number of occurrences of letter $a$ in the word $w$. From the definition of $\pi_a$ we have by induction on $r$

$$\pi_w(\mathbf{x}.\mathbf{q}) = (\mathbf{x} - |w| + \mathbf{N}(w, \mathbf{q})).t_w(\mathbf{q}) \tag{2}$$

where $t_w = t_{w_r} \circ \cdots \circ t_{w_1}$.

In the next lemma and throughout this paper, inequalities on vectors are coordinatewise.

**Lemma 4.2.** (Monotonicity) *For $w, w' \in A^*$ and $\mathbf{q} \in Q$, if $|w| \leq |w'|$, then $\mathbf{N}(w, \mathbf{q}) \leq \mathbf{N}(w', \mathbf{q})$.*

*Proof.* For a vertex $v \in V$ let $p_v : A^* \to A_v^*$ be the projection defined by $p_v(a) = a$ for $a \in A_v$ and $p_v(a) = \epsilon$ (the empty word) for $a \notin A_v$. Since

$$\mathbf{N}(w, \mathbf{q}) = \sum_{v \in V} \mathbf{N}(p_v(w), \mathbf{q})$$

it suffices to prove the lemma for $w, w' \in A_v^*$ for each $v \in V$.

Fix $v \in V$ and $w, w' \in A_v^*$ with $|w| \leq |w'|$. Then there is a word $w''$ such that $|ww''| = |w'|$. Given a letter $a \in A_u$, if $(v, u) \notin E$ then $\mathbf{N}(w, \mathbf{q})_a = \mathbf{N}(w', \mathbf{q})_a = 0$. If $(v, u) \in E$, then since $\mathcal{P}_v$ is an abelian processor,

$$\begin{aligned} \mathbf{N}(w', \mathbf{q})_a = |T_{(v,u)}(w', q_v)|_a &= |T_{(v,u)}(ww'', q_v)|_a \\ &= |T_{(v,u)}(w, q_v)|_a + |T_{(v,u)}(w'', T_v(w, q_v))|_a. \end{aligned}$$

The first term on the right side equals $\mathbf{N}(w, \mathbf{q})_a$, and the remaining term is nonnegative, completing the proof. □

**Lemma 4.3.** *For $w, w' \in A^*$, if $|w| = |w'|$, then $\pi_w = \pi_{w'}$.*

*Proof.* Suppose $|w| = |w'|$. Then for any $\mathbf{q} \in Q$ we have $\mathbf{N}(w, \mathbf{q}) = \mathbf{N}(w', \mathbf{q})$ by Lemma 4.2. Since $t_a$ and $t_b$ commute for all $a, b \in A$, we have $t_w(\mathbf{q}) = t_{w'}(\mathbf{q})$. Hence the right side of (2) is unchanged by substituting $w'$ for $w$. □

Given $\mathbf{y} \in \mathbb{N}^A$, write $\pi_{\mathbf{y}}$ to mean $\pi_w$ for any word $w$ such that $|w| = \mathbf{y}$. Lemma 4.3 ensures that $\pi_{\mathbf{y}}$ is well-defined. We record here two properties of $\pi_{\mathbf{y}}$ that will be useful later.

**Lemma 4.4.** *For all $\mathbf{y}, \mathbf{z} \in \mathbb{N}^A$ we have*

(i) $\pi_{\mathbf{y}+\mathbf{z}} = \pi_{\mathbf{y}} \circ \pi_{\mathbf{z}}$.
(ii) *For all $\mathbf{x}, \mathbf{w} \in \mathbb{Z}^A$ and all $\mathbf{q} \in Q$, if $\pi_{\mathbf{y}}(\mathbf{x}.\mathbf{q}) = \mathbf{x}'.\mathbf{q}'$, then*

$$\pi_{\mathbf{y}}((\mathbf{x}+\mathbf{w}).\mathbf{q}) = (\mathbf{x}'+\mathbf{w}).\mathbf{q}'.$$

*Proof.* Part (i) is immediate from Lemma 4.3. Part (ii) is immediate from (2). □

The letter $a \in A$ is called a *legal move* from $\mathbf{x}.\mathbf{q}$ if $\mathbf{x}_a \geq 1$. An execution $w_1 \cdots w_r$ is called *legal* for $\mathbf{x}.\mathbf{q}$ if $w_{i+1}$ is a legal move from $\mathbf{x}^i.\mathbf{q}^i$ for each $i = 0, \ldots, r-1$. An execution $w_1 \cdots w_r$ is called *complete* for $\mathbf{x}.\mathbf{q}$ if $\mathbf{x}^r_a \leq 0$ for all $a \in A$.

**Lemma 4.5.** (Least Action Principle) *If $w = w_1 \cdots w_r$ is legal for $\mathbf{x}.\mathbf{q}$ and $w' = w'_1 \cdots w'_s$ is complete for $\mathbf{x}.\mathbf{q}$, then $|w| \leq |w'|$ and $r \leq s$.*

*Proof.* Noting that $r = \sum_{a\in A} |w|_a$ and $s = \sum_{a\in A} |w'|_a$, it suffices to prove $|w| \leq |w'|$. Supposing for a contradiction that $|w| \not\leq |w'|$, let $i$ be the smallest index such that $|w_1 \cdots w_i| \not\leq |w'|$. Let $u = w_1 \cdots w_{i-1}$ and $a = w_i$. Then $|u|_a = |w'|_a$, and $|u|_b \leq |w'|_b$ for all $b \neq a$. Since $a$ is a legal move from $\mathbf{x}^{i-1}.\mathbf{q}^{i-1}$, we have by (2) and Lemma 4.2

$$\begin{aligned} 1 &\leq \mathbf{x}^{i-1}_a \\ &= \mathbf{x}^0_a - |u|_a + \mathbf{N}(u, \mathbf{q}^0)_a \\ &\leq \mathbf{x}^0_a - |w'|_a + \mathbf{N}(w', \mathbf{q}^0)_a. \end{aligned}$$

Since $w'$ is complete, the right side is $\leq 0$ by (2), which yields the required contradiction. □

**Lemma 4.6.** (Halting Dichotomy) *For a given initial state $\mathbf{q}$ and input $\mathbf{x}$ to an abelian network $\mathcal{N}$, either*

(1) *There does not exist a finite complete execution; or*
(2) *Every legal execution is finite, and any two complete legal executions $w, w'$ satisfy $|w| = |w'|$.*

*Proof.* If there exists a finite complete execution, say of length $s$, then every legal execution has length $\leq s$ by Lemma 4.5. If $w$ and $w'$ are complete legal executions, then $|w| \leq |w'| \leq |w|$ by Lemma 4.5. □

Note that in case (1) any finite legal execution $w$ can be extended by a legal move: since $w$ is not complete, there is some letter $a \in A$ such that $wa$ is legal. So in this case there is an infinite word $a_1 a_2 \cdots$ such that $a_1 \cdots a_n$ is a legal execution for all $n \geq 1$. The *halting problem* for abelian networks asks, given $\mathcal{N}$, $\mathbf{x}$ and $\mathbf{q}$, whether (1) or (2) of Lemma 4.6 is the case. In case (2) we say that $\mathcal{N}$ *halts* on input $\mathbf{x}.\mathbf{q}$.

**Definition 4.7.** (Odometer) If $\mathcal{N}$ halts on input $\mathbf{x}.\mathbf{q}$, we denote by $[\mathbf{x}.\mathbf{q}]_a = |w|_a$ the total number of letters $a$ processed during a complete legal execution $w$ of $\mathbf{x}.\mathbf{q}$. The vector $[\mathbf{x}.\mathbf{q}] \in \mathbb{N}^A$ is called the *odometer* of $\mathbf{x}.\mathbf{q}$. By Lemma 4.6, the odometer does not depend on the choice of complete legal execution $w$.

No messages remain at the end of a complete legal execution $w$, so the network ends in state $\pi_w(\mathbf{x}.\mathbf{q}) = \mathbf{0}.t_w(\mathbf{q})$. Hence by (2), the odometer can be written as

$$[\mathbf{x}.\mathbf{q}] = |w| = \mathbf{x} + \mathbf{N}(w, \mathbf{q})$$

which simply says that the total number of letters processed (of each type $a \in A$) is the sum of the letters input and the letters produced by message passing. The coordinates of the odometer are the "specific local run times" from §2. We can summarize our progress so far in the following theorem.

**Theorem 4.8.** *Abelian networks have properties (a)–(e) from §2.*

*Proof.* By Lemma 4.6 the halting status does not depend on the execution, which verifies item (a). Moreover for a given $\mathcal{N}, \mathbf{x}, \mathbf{q}$ any two complete legal executions have the same odometer, which verifies items (c)–(e). The odometer and initial state $\mathbf{q}$ determine the final state $t_w(\mathbf{q})$, which verifies (b). □

The next lemma illustrates a general theme of *local-to-global principles* in abelian networks. Suppose we are given a partition $V = I \sqcup O$ of the vertex set into "interior" and "output" nodes, and that the output nodes never send messages (for example, the processor at each output node could be a counter (§3.3)). If $\mathcal{N}$ halts on all inputs, then we can regard the induced subnetwork $(\mathcal{P}_v)_{v \in I}$ of interior nodes as a single processor $\mathcal{P}_I$ with input alphabet $A_I := \sqcup_{v \in I} A_v$, state space $Q_I := \prod_{v \in I} Q_v$, and an output feed for each edge $(v, u) \in I \times O$.

**Lemma 4.9.** (Local Abelianness Implies Global Abelianness) *If $\mathcal{N}$ halts on all inputs and $\mathcal{P}_v$ is an abelian processor for each $v \in I$, then $\mathcal{P}_I$ is an abelian processor.*

*Proof.* Given an input $\iota \in A_I^*$ and an initial state $\mathbf{q} \in Q_I$, we can process one letter at a time to obtain a complete legal execution for $|\iota|.\mathbf{q}$. Now suppose we are given inputs $\iota, \iota'$ such that $|\iota| = |\iota'|$. By Lemma 4.6, any two complete legal executions $w, w'$ for $|\iota|.\mathbf{q}$ satisfy $|w| = |w'|$. In particular, $t_w(\mathbf{q}) = t_{w'}(\mathbf{q})$, so the final state of $\mathcal{P}_I$ does not depend on the order of input. Moreover, for each edge $(v, u) \in I \times O$, since $\mathcal{P}_v$ is an abelian processor,

$$|T_{(v,u)}(w_v, q_v)| = |T_{(v,u)}(w'_v, q_v)|$$

so for each $a \in A_u$ the number of letters $a$ sent along $(v, u)$ does not depend on the order of input. □

For another example of a local-to-global principle, see Lemma 7.6. Further local-to-global principles in the case of rotor networks are explored in [GLPZ12].

**Definition 4.10.** An abelian network $\mathcal{N}$ is *locally finite* if each processor $\mathcal{P}_v$ is a finite-state machine, i.e., each state space $Q_v$ and alphabet $A_v$ is finite.

$\mathcal{N}$ is *spatially finite* if its underlying graph $G$ is finite.

$\mathcal{N}$ is *finite* if it is both locally and spatially finite.

To illustrate an application of the least action principle, consider the case of a finite toppling network, enlarged to allow negative chip counts as described in §3.1. Given a vector $\mathbf{q} \in \mathbb{Z}^A$, write $\mathbf{q}^+ = \max(\mathbf{q}, \mathbf{0})$ and $\mathbf{q}^- = \min(\mathbf{q}, \mathbf{0})$.

**Theorem 4.11.** (Toppling Networks Solve Certain Integer Programs) *Let $\mathcal{N}$ be a spatially finite toppling network with Laplacian $L$ and threshold vector $\mathbf{r} = (r_v)_{v \in V}$. Fix vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^V$ with $\mathbf{a}_v > 0$ for all $v \in V$. Let $\mathbf{q} = \mathbf{b} + \mathbf{r} - \mathbf{1}$. If $\mathcal{N}$ halts on input $\mathbf{q}^+.\mathbf{q}^-$, then its odometer $[\mathbf{q}^+.\mathbf{q}^-]$ is the unique solution $\mathbf{x} \in \mathbb{Z}^V$ to the integer program*

$$\begin{array}{lll} \text{Minimize} & \mathbf{a}^T\mathbf{x} & \\ \text{subject to} & \mathbf{x} \geq \mathbf{0} \quad \text{and} \quad L\mathbf{x} \geq \mathbf{b}. & \end{array} \tag{3}$$

*If $\mathcal{N}$ does not halt on input $\mathbf{q}^+.\mathbf{q}^-$, then (3) has no solution $\mathbf{x} \in \mathbb{Z}^V$.*

*Proof.* Toppling vertex $v$ decreases the number of chips at $v$ by $r_v$ and increases the number of chips at each $u \in V$ by $d_{uv}$. At the end of a complete execution, the number of chips at each vertex $v$ is at most $r_v - 1$. Hence, if $\mathbf{x}_v \geq 0$ is the number of times vertex $v$ topples in a finite complete execution for $\mathbf{q}^+.\mathbf{q}^-$, then

$$\mathbf{q} - L\mathbf{x} \leq \mathbf{r} - \mathbf{1}$$

so $\mathbf{x}$ satisfies (3). Conversely, if $\mathbf{x} \in \mathbb{Z}^V$ is any solution of (3), then any word $w$ with $|w| = \mathbf{x}$ is a finite complete execution for $\mathbf{q}^+.\mathbf{q}^-$. Hence by Lemma 4.6, $\mathcal{N}$ halts on input $\mathbf{q}^+.\mathbf{q}^-$ if and only if (3) has a solution.

If $\mathcal{N}$ halts on input $\mathbf{q}^+.\mathbf{q}^-$, then by Lemma 4.5, the odometer $\mathbf{x}$ is given by

$$\mathbf{x}_v = \min\{|w|_v \,:\, w \text{ is a complete execution for } \mathbf{q}^+.\mathbf{q}^-\}.$$

Hence for each $v \in V$ the odometer minimizes $\mathbf{a}_v\mathbf{x}_v$ among solutions $\mathbf{x} \in \mathbb{Z}^V$ of (3). In particular, it minimizes $\mathbf{a}^T\mathbf{x}$. The strict inequality $\mathbf{a}_v > 0$ ensures the minimizer is unique. □

*Remark.* An interesting feature of the integer program (3) is that the solution does not depend on the objective vector $\mathbf{a}$, provided that $\mathbf{a}_v > 0$ for all $v \in V$.

If we regard the output of an abelian network as the final states of its processors, then the toppling network in Theorem 4.11 outputs not $\mathbf{x}$ but $\mathbf{q} - L\mathbf{x}$. However, it is easy to design an abelian network that computes $\mathbf{x}$ itself. For each $v \in V$ add a new vertex $v'$ whose processor is a counter (§3.3), and have $v$ send one letter to $v'$ each time $v$ topples. Then the final state of $v'$ is $\mathbf{x}_v$.

## 5. Total kernel and production matrix

In this section we define two basic algebraic objects associated to an abelian network, the *total kernel* $K$ and *production matrix* $P$. The former depends only on the state transitions $T_v$, and the latter on the message passing functions $T_{(v,u)}$. We will use them to give a criterion for halting in §6 and to find generators and relations for the critical group in §7.

5.1. **The local action.** Given $\mathbf{x} \in \mathbb{N}^A$ and $\mathbf{q} \in Q$, define

$$\mathbf{x} \triangleright \mathbf{q} := \pi_{\mathbf{x}}(\mathbf{x}.\mathbf{q}).$$

In words, $\mathbf{x} \triangleright \mathbf{q} \in \mathbb{N}^A \times Q$ is the result of performing the following operation starting from state $\mathbf{q}$: "for each $a \in A$ add $\mathbf{x}_a$ letters of type $a$ and process each letter once." If $\mathbf{x} \triangleright \mathbf{q} = \mathbf{y}.\mathbf{r}$ then message passing produced a total of $\mathbf{y}_a$ letters of type $a$ for each $a \in A$, and the resulting state of the processor at vertex $v$ was $\mathbf{r}_v$ for each $v \in V$.

For any $\mathbf{z} \in \mathbb{Z}^A$ we also write

$$\mathbf{x} \triangleright (\mathbf{z}.\mathbf{q}) := \pi_{\mathbf{x}}((\mathbf{x} + \mathbf{z}).\mathbf{q}).$$

The next lemma shows that $\triangleright$ defines a monoid action of $\mathbb{N}^A$ on $\mathbb{Z}^A \times Q$. We call this the *local action* because each processor $\mathcal{P}_v$ processes only the letters $\mathbf{x}_v$ that were added at $v$. (It does not process any additional letters it may receive due to message passing from its neighbors, nor the letters $\mathbf{z}_v$ that were "already present.")

**Lemma 5.1.** *For any $\mathbf{x}, \mathbf{y} \in \mathbb{N}^A$ and $\mathbf{z} \in \mathbb{Z}^A$ and any $\mathbf{q} \in Q$,*

$$\mathbf{x} \triangleright (\mathbf{y} \triangleright (\mathbf{z}.\mathbf{q})) = (\mathbf{x} + \mathbf{y}) \triangleright (\mathbf{z}.\mathbf{q}).$$

*Proof.* Write $\mathbf{y} \triangleright \mathbf{q} = \mathbf{y}'.\mathbf{q}'$. By Lemma 4.4(i), since $\pi_{\mathbf{x}+\mathbf{y}} = \pi_{\mathbf{x}} \circ \pi_{\mathbf{y}}$, we have

$$\begin{aligned}
(\mathbf{x} + \mathbf{y}) \triangleright (\mathbf{z}.\mathbf{q}) &= \pi_{\mathbf{x}+\mathbf{y}}((\mathbf{x} + \mathbf{y} + \mathbf{z}).\mathbf{q}) \\
&= \pi_{\mathbf{x}}(\pi_{\mathbf{y}}((\mathbf{x} + \mathbf{y} + \mathbf{z}).\mathbf{q})) \\
&= \pi_{\mathbf{x}}((\mathbf{x} + \mathbf{y}' + \mathbf{z}).\mathbf{q}') \\
&= \mathbf{x} \triangleright ((\mathbf{y}' + \mathbf{z}).\mathbf{q}') \qquad (4) \\
&= \mathbf{x} \triangleright (\mathbf{y} \triangleright (\mathbf{z}.\mathbf{q}))
\end{aligned}$$

where in the third and last equalities we have used Lemma 4.4(ii). □

Recall from Definition 4.1 the local monoid $M_v$ generated by the maps $t_a$ for $a \in A_v$ sending $q \mapsto T_v(a, q)$. Write $t_v : \mathbb{N}^{A_v} \to M_v$ for the monoid homomorphism sending $\mathbf{1}_a \mapsto t_a$ for $a \in A_v$. Denote by

$$t : \mathbb{N}^A \to \prod_{v \in V} M_v$$

the Cartesian product of the maps $t_v$. Each $t_v$ is surjective by the definition of $M_v$, so $t$ is surjective. Note that if $\mathbf{y} \triangleright \mathbf{q} = \mathbf{z}.\mathbf{r}$, then $\mathbf{r} = t(\mathbf{y})\mathbf{q}$. Given $\mathbf{m} \in \prod_{v \in V} M_v$, write $\mathbf{m}\mathbf{q} = (\mathbf{m}_v \mathbf{q}_v)_{v \in V}$. In general, knowing $\mathbf{y} \triangleright \mathbf{q} = \mathbf{z}.\mathbf{r}$ does not determine $\mathbf{y} \triangleright (\mathbf{m}\mathbf{q})$, but the next lemma shows that it does in the case $\mathbf{r} = \mathbf{q}$.

**Lemma 5.2.** *If $\mathbf{y} \triangleright \mathbf{q} = \mathbf{z}.\mathbf{q}$, then $\mathbf{y} \triangleright (\mathbf{m}\mathbf{q}) = \mathbf{z}.(\mathbf{m}\mathbf{q})$ for all $\mathbf{m} \in \prod_{v \in V} M_v$.*

*Proof.* Since $t$ is surjective there exists $\mathbf{u} \in \mathbb{N}^A$ such that $t(\mathbf{u}) = \mathbf{m}$. Then $\mathbf{u} \triangleright \mathbf{q} = \mathbf{w}.\mathbf{m}\mathbf{q}$ for some $\mathbf{w} \in \mathbb{N}^A$. By Lemma 5.1,

$$\mathbf{y} \triangleright (\mathbf{w}.\mathbf{m}\mathbf{q}) = \mathbf{y} \triangleright (\mathbf{u} \triangleright \mathbf{q}) = \mathbf{u} \triangleright (\mathbf{y} \triangleright \mathbf{q}) = \mathbf{u} \triangleright (\mathbf{z}.\mathbf{q}) = (\mathbf{w} + \mathbf{z}).\mathbf{m}\mathbf{q}.$$

From Lemma 4.4(ii) it follows that $\mathbf{y} \triangleright \mathbf{m}\mathbf{q} = \mathbf{z}.\mathbf{m}\mathbf{q}$. □

5.2. **Production matrix.** Let $e_v$ be the minimal idempotent of the local monoid $M_v$. The recurrent elements (Lemma A.3) of the monoid action $M_v \times Q_v \to Q_v$ play a special role in this section.

**Definition 5.3.** A state $\mathbf{q} \in Q$ is *locally recurrent* if $\mathbf{q}_v \in e_v Q_v$ for all $v \in V$. (Equivalently, $\mathbf{q}_v = e_v \mathbf{q}_v$ for all $v \in V$.)

By Lemma A.4, every $m \in M_v$ acts invertibly on $e_v Q_v$. Thus for each $a \in A_v$ the map $q \mapsto T_v(a, q)$ is a permutation of $e_v Q_v$, so we have a group action

$$\mathbb{Z}^{A_v} \times e_v Q_v \to e_v Q_v.$$

Let $K_v$ be the set of vectors in $\mathbb{Z}^{A_v}$ that act as the identity on $e_v Q_v$.

**Definition 5.4.** The *total kernel* of $\mathcal{N}$ is the subgroup of $\mathbb{Z}^A$ given by

$$K = \prod_{v \in V} K_v.$$

From here on we assume that $\mathcal{N}$ is a *finite* abelian network (Definition 4.10). The main ingredients that rely on finiteness are the results from §A.

**Lemma 5.5.** *If $\mathcal{N}$ is finite, then $K$ is a subgroup of finite index in $\mathbb{Z}^A$. In particular, $K$ is generated as a group by $K \cap \mathbb{N}^A$.*

*Proof.* Since $\mathcal{N}$ is locally finite, each $e_v Q_v$ is a finite set, so for any $\mathbf{x} \in \mathbb{Z}^{A_v}$ we have $n\mathbf{x} \in K_v$ for some $n \geq 1$. Thus $K_v$ has finite index in $\mathbb{Z}^{A_v}$. Since $\mathcal{N}$ is spatially finite, $V$ is a finite set, so $K = \prod_{v \in V} K_v$ has finite index in $\mathbb{Z}^A$. In particular, $K$ contains a vector with all coordinates strictly positive, which implies that $K$ is generated as a group by $K \cap \mathbb{N}^A$. □

Note that

$$K \cap \mathbb{N}^A = \{\mathbf{x} \in \mathbb{N}^A \mid t(\mathbf{x})\mathbf{q} = \mathbf{q} \text{ for all locally recurrent } \mathbf{q} \in Q\}. \tag{5}$$

Fix a locally recurrent state $\mathbf{q} \in Q$. For any $\mathbf{k} \in K \cap \mathbb{N}^A$ we have

$$\mathbf{k} \triangleright \mathbf{q} = P_{\mathbf{q}}(\mathbf{k}).\mathbf{q} \tag{6}$$

for some vector $P_{\mathbf{q}}(\mathbf{k}) \in \mathbb{N}^A$. Next we show that

$$P_{\mathbf{q}} : K \cap \mathbb{N}^A \to \mathbb{N}^A$$

extends to a group homomorphism.

**Lemma 5.6.** *Let $\mathcal{N}$ be a finite abelian network. Then $P_{\mathbf{q}}$ extends to a group homomorphism $K \to \mathbb{Z}^A$.*

*Proof.* Write $P = P_{\mathbf{q}}$. Let $\mathbf{k}_1, \mathbf{k}_2 \in K \cap \mathbb{N}^A$. By Lemma 5.1,

$$(\mathbf{k}_1 + \mathbf{k}_2) \triangleright \mathbf{q} = \mathbf{k}_1 \triangleright (P(\mathbf{k}_2).\mathbf{q}) = (P(\mathbf{k}_1) + P(\mathbf{k}_2)).\mathbf{q}$$

hence

$$P(\mathbf{k}_1 + \mathbf{k}_2) = P(\mathbf{k}_1) + P(\mathbf{k}_2). \tag{7}$$

By Lemma 5.5, every $\mathbf{k} \in K$ can be written as $\mathbf{k}_1 - \mathbf{k}_2$ for $\mathbf{k}_1, \mathbf{k}_2 \in K \cap \mathbb{N}^A$. Define $P(\mathbf{k}) = P(\mathbf{k}_1) - P(\mathbf{k}_2)$. Equation (7) now implies that this extension is well defined and a group homomorphism. □

By tensoring $P_{\mathbf{q}} : K \to \mathbb{Z}^A$ with $\mathbb{Q}$, we obtain a linear map $P_{\mathbf{q}} : \mathbb{Q}^A \to \mathbb{Q}^A$. To be more explicit, for any $\mathbf{x} \in \mathbb{Q}^A$, by Lemma 5.5 there is an integer $n \geq 1$ such that $n\mathbf{x} \in K$, and we define

$$P_{\mathbf{q}}(\mathbf{x}) := \frac{1}{n} P_{\mathbf{q}}(n\mathbf{x}).$$

So far we have defined $P_{\mathbf{q}}$ only for locally recurrent $\mathbf{q}$. We extend the definition to all states $\mathbf{q} = (q_v)_{v \in V}$ by setting $P_{\mathbf{q}} := P_{\widehat{\mathbf{q}}}$, where $\widehat{\mathbf{q}} = (e_v q_v)_{v \in V}$.

**Definition 5.7.** The *production matrix* of a finite abelian network $\mathcal{N}$ with initial state $\mathbf{q}$ is the matrix of the linear map $P_{\mathbf{q}} : \mathbb{Q}^A \to \mathbb{Q}^A$.

The $(a, b)$ entry $p_{ab}$ of the production matrix says "on average" how many letters $a$ are created by processing the letter $b$: specifically, if $n1_b \in K$, then $p_{ab} = \frac{1}{n} p_{ab}(n)$, where $p_{ab}(n) = \mathbf{N}(b^n, q)_a$ is the number of $a$'s created by executing the word $b^n$. We have chosen the term "production matrix" to evoke [DFR05]. Indeed the succession rules studied in that paper can be modeled by an abelian network whose underlying graph is a single vertex with a loop.

We remark that the production matrix can be also defined for some networks that are not finite by setting

$$p_{ab} := \lim_{n \to \infty} \frac{1}{n} p_{ab}(n)$$

if this limit exists.

5.3. **Local components.** If $\mathcal{P}$ be an abelian finite automaton with state space $Q$ and alphabet $A$. That is, we have transition maps $t_a : Q \to Q$ for $a \in A$, such that $t_a \circ t_b = t_b \circ t_a$ for all $a, b \in A$. The *transition monoid* of $\mathcal{P}$ is the submonoid $M \subset \mathrm{End}\,(Q)$ generated by $\{t_a\}_{a \in A}$, where $\mathrm{End}\,(Q)$ denotes the monoid of all set maps $Q \to Q$ under composition. For example, the local monoid $M_v$ of §4 is the transition monoid of $\mathcal{P}_v$; later, in §7 we will study the "global" transition monoid of an abelian network that halts on all inputs.

We say that $\mathcal{P}$ is *irreducible* if the monoid action $M \times Q \to Q$ is irreducible (§A). The *irreducible components* of $\mathcal{P}$ are the processors with alphabet $A$ and state space $Q_\alpha$, where $Q = \sqcup Q_\alpha$ is the partition of $Q$ into equivalence classes under the relation $q \sim q'$ if there exist $m, m' \in M$ such that $mq = m'q'$.

We say that an abelian network $\mathcal{N} = (\mathcal{P}_v)_{v \in V}$ is *locally irreducible* if each processor $\mathcal{P}_v$ is irreducible. The *local components* of $\mathcal{N}$ are the abelian networks $\mathcal{N}_\alpha = (\mathcal{P}_v^{\alpha_v})_{v \in V}$ where each $\mathcal{P}_v^{\alpha_v}$ is an irreducible component of $\mathcal{P}_v$. Note that the local components have the same underlying graph $G$ and alphabet $A$, with a possibly smaller state space at each vertex.

For $\mathbf{q}, \mathbf{r} \in Q$ we write $\mathbf{q} \sim \mathbf{r}$ if $\mathbf{q}$ and $\mathbf{r}$ belong to the same local component; that is, $\mathbf{q}_v \sim \mathbf{r}_v$ for all $v \in V$). We denote by $\mathcal{N}_{\mathbf{q}}$ the local component of $\mathcal{N}$ containing state $\mathbf{q}$.

*Example.* If $\mathcal{N}$ is a height arrow network (§3.6), a vertex starting in state $(q, c)$ can only access states of the form $(q', c')$ where $q' = q + n\tau_v \pmod{d_v}$ for some $n \in \mathbb{N}$. If $g_v := \gcd(d_v, \tau_v) > 1$, then $Q_v$ is not irreducible. Each processor $\mathcal{P}_v$ has $g_v$ irreducible components, and $\mathcal{N}$ has $\prod_{v \in V} g_v$ local components.

The next lemma should be compared with (5).

**Lemma 5.8.** *If $\mathcal{N}$ is finite and locally irreducible, then for any fixed locally recurrent $\mathbf{q} \in Q$ we have*

$$K \cap \mathbb{N}^A = \left\{ \mathbf{k} \in \mathbb{N}^A \mid t(\mathbf{k})\mathbf{q} = \mathbf{q} \right\}.$$

*Proof.* By Theorem A.5 the action of $e_v M_v$ on $e_v Q_v$ is free. If $\mathbf{k} \in \mathbb{N}^A$ and $t(\mathbf{k})\mathbf{q} = \mathbf{q}$ for one locally recurrent $\mathbf{q}$, then for all $v \in V$

$$t_v(\mathbf{k}_v)e_v\mathbf{q}_v = t_v(\mathbf{k}_v)\mathbf{q}_v = \mathbf{q}_v.$$

By freeness it follows that $t(\mathbf{k}_v)e_v = e_v$, so $t(\mathbf{k})\mathbf{r} = \mathbf{r}$ for all locally recurrent $\mathbf{r}$. Hence $\mathbf{k} \in K$. □

**Lemma 5.9.** *If $\mathbf{q} \sim \mathbf{r}$ then $P_\mathbf{q} = P_\mathbf{r}$. If $\mathcal{N}$ is finite and locally irreducible, then the production matrix $P_\mathbf{q}$ does not depend on the initial state $\mathbf{q}$.*

*Proof.* The first statement follows from the second applied to the local component $\mathcal{N}_\mathbf{q}$. To prove the second statement, suppose that $\mathcal{N}$ is locally irreducible and let $\mathbf{q}_1, \mathbf{q}_2 \in Q$ be locally recurrent. By Theorem A.5 each group action $e_v M_v \times e_v Q_v \to e_v Q_v$ is transitive, so there exists $\mathbf{x} \in \mathbb{N}^A$ such that $t(\mathbf{x})\mathbf{q}_1 = \mathbf{q}_2$. Let $\mathbf{z}$ be such that

$$\mathbf{x} \triangleright \mathbf{q}_1 = \mathbf{z}.\mathbf{q}_2.$$

Fix $\mathbf{k} \in K \cap \mathbb{N}^A$, and let $\mathbf{y}_i = P_{\mathbf{q}_i}(\mathbf{k})$ for $i = 1, 2$. Then

$$\mathbf{x} \triangleright (\mathbf{k} \triangleright \mathbf{q}_1) = \mathbf{x} \triangleright (\mathbf{y}_1.\mathbf{q}_1) = (\mathbf{y}_1 + \mathbf{z}).\mathbf{q}_2$$

while

$$\mathbf{k} \triangleright (\mathbf{x} \triangleright \mathbf{q}_1) = \mathbf{k} \triangleright (\mathbf{z}.\mathbf{q}_2) = (\mathbf{y}_2 + \mathbf{z}).\mathbf{q}_2.$$

By Lemma 5.1,

$$\mathbf{k} \triangleright (\mathbf{x} \triangleright \mathbf{q}) = \mathbf{x} \triangleright (\mathbf{k} \triangleright \mathbf{q})$$

hence $\mathbf{y}_1 + \mathbf{z} = \mathbf{y}_2 + \mathbf{z}$, and hence $\mathbf{y}_1 = \mathbf{y}_2$. □

5.4. **Strong components.** Let $\mathcal{N}$ be a locally irreducible finite abelian network with production matrix $P = (p_{ab})_{a,b \in A}$.

**Definition 5.10.** The *production graph* $\Gamma = \Gamma(\mathcal{N})$ of $\mathcal{N}$ is the directed graph with vertex set $A$ and edge set $\{(a, b) \mid p_{ba} > 0\}$.

Write $a \to b$ if there is a directed path in $\Gamma$ from $a$ to $b$. The *strong components* of $\Gamma$ are the equivalence classes of the relation $\{(a, b) \mid a \to b \text{ and } b \to a\}$.

**Definition 5.11.** The *strong components* of $\mathcal{N}$ are the subnetworks $\mathcal{N}^1, \ldots, \mathcal{N}^s$ with alphabets $A^1, \ldots, A^s$, where $A^1, \ldots, A^s$ are the strong components of $\Gamma$.

Note that the strong components are distinct from the local components of §5.3. The former are defined by restricting the alphabet, the latter by restricting the state space. Furthermore, the strong components of a locally irreducible network need not be locally irreducible (Figure 7). However, Lemma 5.13 below shows that the local components of a strong component do not decompose any further. Moreover, all local components of a given strong component are homotopic.

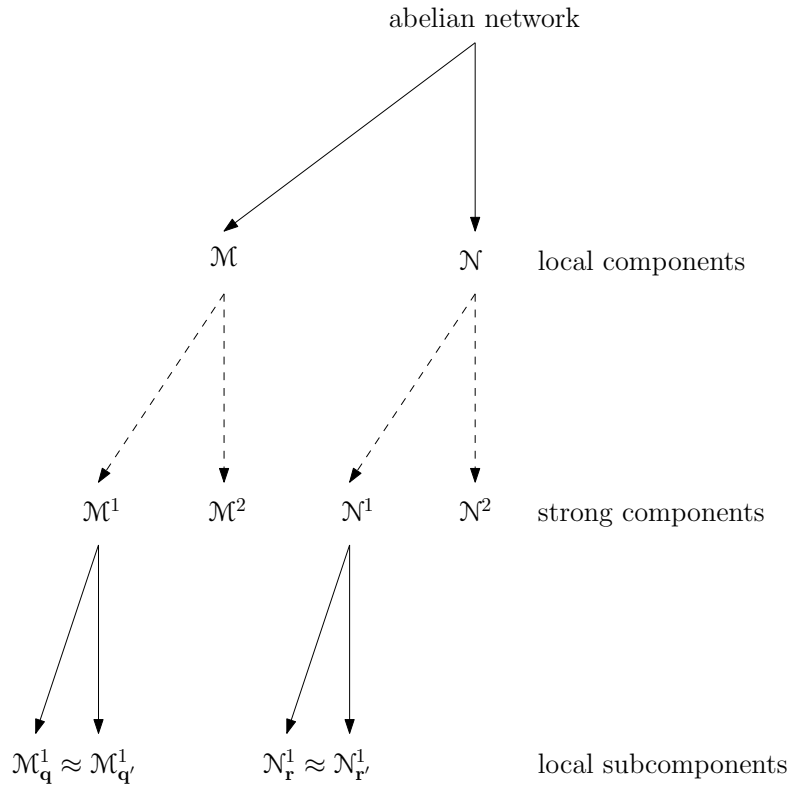


FIGURE 7. The local components of an abelian network may decompose into strong components, which may further decompose further into local subcomponents. Solid arrows represent restrictions of the state space, and dashed arrows represent restrictions of the alphabet.

**Definition 5.12.** (Homotopy) Locally irreducible abelian networks $\mathcal{N}$ and $\mathcal{N}'$ on the same graph with the same total alphabet are called *homotopic*, written $\mathcal{N} \approx \mathcal{N}'$, if they have the same total kernel $K$ and the same production matrix $P$.

Here is the reason for calling this homotopy. If $\mathcal{N}$ and $\mathcal{N}'$ have the same total kernel $K$, then for each vertex $v$ the state diagrams of processors $\mathcal{P}_v$ and $\mathcal{P}'_v$ live on the same discrete torus $\mathbb{Z}^{A_v}/K_v$. Then $\mathcal{N}$ and $\mathcal{N}'$ have the same production matrix if and only if for each $v$ the state diagram of $\mathcal{P}'_v$ can be obtained from the state diagram of $\mathcal{P}_v$ by altering message surfaces without changing the homotopy type of any surface.

**Lemma 5.13.** *Let $\mathcal{N}$ be a locally irreducible finite abelian network with strong components $\mathcal{N}^i$, total kernel $K$ and production matrix $P$. Let $\mathcal{N}^i_{\mathbf{q}}$ be the local component of $\mathcal{N}^i$ containing state $\mathbf{q}$. The following hold for all $\mathbf{q}, \mathbf{q}' \in Q$.*

(i) *The total kernel of $\mathcal{N}^i_{\mathbf{q}}$ is the group generated by $K \cap \mathbb{N}^{A^i}$.*
(ii) *The production matrix of $\mathcal{N}^i_{\mathbf{q}}$ is the $A^i \times A^i$ submatrix $P_{ii}$ of $P$.*

(iii) $\mathcal{N}^i_{\mathbf{q}}$ *has only one strong component.*
(iv) $\mathcal{N}^i_{\mathbf{q}} \approx \mathcal{N}^i_{\mathbf{q}'}$.

*Proof.* Since $\mathcal{N}^i_{\mathbf{q}} = \mathcal{N}^i_{\widehat{\mathbf{q}}}$ where $\widehat{\mathbf{q}} = (e^i_v \mathbf{q}_v)_{v\in V}$, we may assume that $\mathbf{q}$ is a locally recurrent state of $\mathcal{N}^i$.

(i) By Lemma 5.5, the total kernel $K^i_{\mathbf{q}}$ of $\mathcal{N}^i_{\mathbf{q}}$ is generated as a group by $K^i_{\mathbf{q}} \cap \mathbb{N}^{A^i}$. Since both $\mathcal{N}^i_{\mathbf{q}}$ and $\mathcal{N}$ are locally irreducible, by Lemma 5.8 we have for $\mathbf{k} \in \mathbb{N}^{A^i}$

$$\mathbf{k} \in K^i_{\mathbf{q}} \iff t(\mathbf{k})\mathbf{q} = \mathbf{q} \iff \mathbf{k} \in K.$$

Hence

$$K^i_{\mathbf{q}} \cap \mathbb{N}^{A^i} = K \cap \mathbb{N}^{A^i}.$$

(ii) Write $\triangleright$ for the local action of $\mathcal{N}$ and $\triangleright^i$ for the local action of $\mathcal{N}^i$. For $\mathbf{k} \in \mathbb{N}^{A^i}$ the only difference between these actions is that $\mathbf{k} \triangleright \mathbf{q}$ may produce some letters in $A - A^i$ in addition to the letters in $A^i$ produced by $\mathbf{k} \triangleright^i \mathbf{q}$. Hence, writing $P^i_{\mathbf{q}}$ for the production matrix of $\mathcal{N}^i_{\mathbf{q}}$, if $\mathbf{k} \in K \cap \mathbb{N}^{A^i}$ then $\mathbf{k} \triangleright \mathbf{q} = \mathbf{y}.\mathbf{q}$, where $\mathbf{y} = P^i_{\mathbf{q}}\mathbf{k} + \mathbf{z}$ for some $\mathbf{z} \in \mathbb{N}^{A-A^i}$. Letting $\mathbf{r} = (e_v \mathbf{q}_v)_{v\in V}$, we have $\mathbf{k} \triangleright \mathbf{r} = \mathbf{y}.\mathbf{r}$ by Lemma 5.2. Since $\mathbf{r}$ is locally recurrent for $\mathcal{N}$, it follows that $\mathbf{y} = P\mathbf{k}$. Writing $\rho_i$ for the projection $\mathbb{N}^A \to \mathbb{N}^{A^i}$, we conclude that

$$P^i_{\mathbf{q}}\mathbf{k} = \rho_i(P\mathbf{k}) = P_{ii}\mathbf{k}$$

and hence $P^i_{\mathbf{q}} = P_{ii}$.

(iii) By part (ii), the production graph of $\mathcal{N}^i_{\mathbf{q}}$ is the strong component $A^i$ of $\Gamma$, so $\mathcal{N}^i_{\mathbf{q}}$ has only one strong component.

(iv) By parts (i) and (ii), $K^i_{\mathbf{q}}$ and $P^i_{\mathbf{q}}$ do not depend on $\mathbf{q}$, so all local components of $\mathcal{N}^i$ are homotopic. □

The strong components are partially ordered by accessibility. If we label them so that $A^i \to A^j$ implies $i \geq j$, then the production matrix is block triangular

$$P = \begin{pmatrix} P_{11} & P_{12} & \cdots & P_{1s} \\ 0 & P_{22} & \cdots & P_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & P_{ss} \end{pmatrix}$$

and the diagonal block $P_{ii}$ is the production matrix of $\mathcal{N}^i$.

## 6. Halting problem

Let $\mathcal{N}$ be an abelian network with total state space $Q$, and fix a state $\mathbf{q} \in Q$. If case (2) of Lemma 4.6 holds for all inputs $\mathbf{x} \in \mathbb{N}^A$, then we say that $\mathcal{N}$ *halts on all inputs to initial state* $\mathbf{q}$. If this is the case for all $\mathbf{q} \in Q$, then we say that $\mathcal{N}$ *halts on all inputs*. The main result of this section is Theorem 6.6, which gives an efficient way to decide whether a finite abelian network halts on all inputs to a given initial state. The case of a toppling network is due to Gabrielov [Gab94].

If $\mathbf{x} \leq \mathbf{y}$, then any legal execution for $\mathbf{x}.\mathbf{q}$ is a legal execution for $\mathbf{y}.\mathbf{q}$. It follows that halting is a monotone property:

If $\mathcal{N}$ halts on input $\mathbf{y}.\mathbf{q}$, then $\mathcal{N}$ halts on all inputs $\mathbf{x}.\mathbf{q}$ for $\mathbf{x} \leq \mathbf{y}$. (8)

Recall the equivalence relation $\sim$ on $Q$ introduced in §5.3.

**Lemma 6.1.** *If $\mathbf{q}_1 \sim \mathbf{q}_2$, then $\mathcal{N}$ halts on all inputs to initial state $\mathbf{q}_1$ if and only if $\mathcal{N}$ halts on all inputs to initial state $\mathbf{q}_2$.*

*Proof.* If $\mathbf{q}_1 \sim \mathbf{q}_2$ then $t(\mathbf{y})\mathbf{q}_1 = t(\mathbf{y})\mathbf{q}_2$ for some $\mathbf{y} \in \mathbb{N}^A$. Thus, it suffices to show that $\mathcal{N}$ halts on all inputs to initial state $\mathbf{q}$ if and only if $\mathcal{N}$ halts on all inputs to initial state $t(\mathbf{y})\mathbf{q}$. Let $\mathbf{y} \triangleright \mathbf{q} = \mathbf{z}.\mathbf{r}$. Then $\mathbf{r} = t(\mathbf{y})\mathbf{q}$. For any $\mathbf{x} \in \mathbb{N}^A$, we have

$$(\mathbf{x} + \mathbf{y}) \triangleright \mathbf{q} = \mathbf{x} \triangleright (\mathbf{y} \triangleright \mathbf{q}) = \mathbf{x} \triangleright (\mathbf{z}.\mathbf{r})$$

so both $(\mathbf{x} + \mathbf{y}).\mathbf{q}$ and $(\mathbf{x} + \mathbf{z}).\mathbf{r}$ have legal executions resulting in $(\mathbf{x} + \mathbf{y}) \triangleright \mathbf{q}$. Thus $\mathcal{N}$ halts on input $(\mathbf{x} + \mathbf{y}).\mathbf{q}$ if and only if $\mathcal{N}$ halts on input $(\mathbf{x} + \mathbf{z}).\mathbf{r}$. By monotonicity (8), it follows that $\mathcal{N}$ halts on all inputs to initial state $\mathbf{q}$ if and only if $\mathcal{N}$ halts on all inputs to initial state $\mathbf{r}$. □

**Definition 6.2.** A state $\mathbf{x}.\mathbf{q}$ is an *amplifier* if $\mathbf{x} \in \mathbb{N}^A$ and there exists a nonempty legal execution $w$ from $\mathbf{x}.\mathbf{q}$ such that $\pi_w(\mathbf{x}.\mathbf{q}) = \mathbf{y}.\mathbf{q}$ for some $\mathbf{y} \geq \mathbf{x}$.

**Definition 6.3.** A state $\mathbf{x}.\mathbf{q}$ is a *strong amplifier* if $\mathbf{x} \in \mathbb{N}^A - \{\mathbf{0}\}$ and $\mathbf{x} \triangleright \mathbf{q} = \mathbf{y}.\mathbf{q}$ for some $\mathbf{y} \geq \mathbf{x}$.

In words, a strong amplifier is a pair $\mathbf{x}.\mathbf{q}$ with the property that after processing all letters once, the network has returned to the same state $\mathbf{q}$ with at least as many letters of each type as before.

*Example.* In the sandpile network $\texttt{Sand}(G)$ of an undirected graph with no sink, let $\mathbf{x} = (d_v)_{v \in V}$ be the configuration where each vertex has the same number of letters ("chips") as its degree. For any initial state $\mathbf{q}$, processing all letters once causes each vertex to topple once, so that each vertex $v$ receives one letter from each of its $d_v$ neighbors. Hence $\mathbf{x} \triangleright \mathbf{q} = \mathbf{x}.\mathbf{q}$, and $\mathbf{x}.\mathbf{q}$ is a strong amplifier.

**Lemma 6.4.** *The following are equivalent for a finite abelian network $\mathcal{N}$.*

1. *$\mathcal{N}$ has an amplifier.*
2. *$\mathcal{N}$ has a strong amplifier.*
3. *$\mathcal{N}$ fails to halt on some input.*

*Proof.* If $\mathcal{N}$ has an amplifier $\mathbf{x}.\mathbf{q}$, then there is a legal execution $\pi_{\mathbf{u}}(\mathbf{x}.\mathbf{q}) = \mathbf{y}.\mathbf{q}$ for some $\mathbf{y} \geq \mathbf{x}$ and $\mathbf{u} \in \mathbb{N}^A - \{\mathbf{0}\}$. Then $\mathbf{u} \triangleright \mathbf{q} = \pi_{\mathbf{u}}(\mathbf{u}.\mathbf{q}) = (\mathbf{u} + \mathbf{y} - \mathbf{x}).\mathbf{q}$, where the second equality follows from Lemma 4.4(ii). Therefore $\mathbf{u}.\mathbf{q}$ is a strong amplifier, which shows that (1) ⇒ (2).

Next if $\mathcal{N}$ has a strong amplifier $\mathbf{u}.\mathbf{q}$, then there is a legal execution $w$ with $|w| = \mathbf{u} \in \mathbb{N}^A - \{\mathbf{0}\}$ starting with $\mathbf{u}.\mathbf{q}$ and ending with $(\mathbf{u} + \mathbf{v}).\mathbf{q}$ for some $\mathbf{v} \geq \mathbf{0}$. Then for any $n \geq 0$ the same $w$ is a legal execution starting with $(\mathbf{u} + n\mathbf{v}).\mathbf{q}$ and ending with $(\mathbf{u} + (n+1)\mathbf{v}).\mathbf{q}$. Hence $w^n$ is a legal execution starting from $\mathbf{u}.\mathbf{q}$ for all $n \geq 0$. Since there exist arbitrarily long legal executions, we conclude from Lemma 4.6 that $\mathcal{N}$ does not halt on input $\mathbf{u}.\mathbf{q}$, which shows that (2) ⇒ (3).

Lastly, suppose that $\mathcal{N}$ fails to halt on some input $\mathbf{y}.\mathbf{q}$. Then there is an infinite word $w_1 w_2 \cdots$ such that $w_1 \cdots w_n$ is a legal execution for all $n \geq 1$. Let $\mathbf{y}_n.\mathbf{q}_n = \pi_{w_1 \cdots w_n}(\mathbf{y}.\mathbf{q})$. Since the total state space $Q$ is finite, there exists $\mathbf{q} \in Q$ such that $\mathbf{q}_n = \mathbf{q}$ for infinitely many $n$. By Dickson's Lemma B.1, there exist indices $j < k$ such that $\mathbf{q}_j = \mathbf{q}_k = \mathbf{q}$ and $\mathbf{y}_j \leq \mathbf{y}_k$. This $\mathbf{y}_j.\mathbf{q}_j$ is an amplifier, which shows that (3) $\Rightarrow$ (1). □

The next lemma is a variant of Lemma 6.4 with distinguished initial state. Recall from §5.3 the local component $\mathcal{N}_\mathbf{q}$ containing state $\mathbf{q}$.

**Lemma 6.5.** *The following are equivalent for a finite abelian network $\mathcal{N}$ and state $\mathbf{q}$.*

(1) *$\mathcal{N}_\mathbf{q}$ has an amplifier.*
(2) *$\mathcal{N}_\mathbf{q}$ has a strong amplifier.*
(3) *$\mathcal{N}$ fails to halt on some input to initial state $\mathbf{q}$.*

*Proof.* By Lemma 6.4 it suffices to show that (3) is equivalent to the statement that $\mathcal{N}_\mathbf{q}$ fails to halt on some input. Any execution for $\mathbf{x}.\mathbf{q}$ in $\mathcal{N}$ is also an execution for $\mathbf{x}.\mathbf{q}$ in $\mathcal{N}_\mathbf{q}$, so (3) is equivalent to the statement that $\mathcal{N}_\mathbf{q}$ fails to halt on some input to initial state $\mathbf{q}$. By Lemma 6.1, if $\mathcal{N}_\mathbf{q}$ fails to halt on some input, then $\mathcal{N}_\mathbf{q}$ fails to halt on some input to initial state $\mathbf{q}$, which completes the proof. □

**Theorem 6.6.** (Halting Criterion 1) *A finite abelian network $\mathcal{N}$ halts on all inputs to initial state $\mathbf{q}$ if and only if every eigenvalue of the production matrix $P_\mathbf{q}$ has absolute value strictly less than 1.*

*Proof.* Let $P = P_\mathbf{q}$, and let $\mathbf{x}$ and $\lambda$ be the Perron-Frobenius eigenvector and eigenvalue of $P$; if $\lambda = 1$ then we may take $\mathbf{x}$ to have integer entries (Lemma C.1). By Lemma 6.5 it suffices to show that (1) if $\lambda \geq 1$, then $\mathcal{N}_\mathbf{q}$ has an amplifier; and (2) if $\mathcal{N}_\mathbf{q}$ has a strong amplifier, then $\lambda \geq 1$.

(1) If $\lambda \geq 1$, then there is a vector $\mathbf{y} \in \mathbb{Q}^A$ such that $\mathbf{x} \leq \mathbf{y} \leq \lambda\mathbf{x}$. Then $P\mathbf{y} \geq P\mathbf{x} = \lambda\mathbf{x} \geq \mathbf{y}$. Choosing $n \geq 1$ such that $n\mathbf{y} \in K$, we have $n\mathbf{y} \triangleright \mathbf{q} = P(n\mathbf{y}).\mathbf{q}$, so $n\mathbf{y}.\mathbf{q}$ is an amplifier.

(2) If $\mathcal{N}_\mathbf{q}$ has a strong amplifier $\mathbf{y}.\mathbf{q}'$, then by Lemma 5.2, $\mathbf{y}.\mathbf{m}\mathbf{q}'$ is also a strong amplifier for any $\mathbf{m} \in \prod_{v \in V} M_v$. In particular, taking $\mathbf{m} = (e_v)_{v \in V}$ we obtain a strong amplifier $\mathbf{y}.\mathbf{r}$ with $\mathbf{r}$ locally recurrent and $\mathbf{r} \sim \mathbf{q}$. By Lemma 5.9, $P_\mathbf{q}(\mathbf{y}) = P_\mathbf{r}(\mathbf{y}) \geq \mathbf{y}$, which shows that $\lambda \geq 1$. □

*Remark.* In the case that $\mathcal{N}$ is locally irreducible, the production matrix $P_\mathbf{q}$ does not depend on $\mathbf{q}$ by Lemma 5.9. In this case Theorem 6.6 gives a criterion for $\mathcal{N}$ to halt on all inputs regardless of initial state.

If $P_\mathbf{q}$ has Perron-Frobenius eigenvalue $\lambda \geq 1$, then $\mathcal{N}$ may run forever on some inputs to initial state $\mathbf{q}$ and halt on other inputs. In the next section we examine how to tell which is the case.

6.1. **Certifying that a network never halts.** How long must we run an abelian network until we can be sure it will not halt? The next lemma shows that any strong amplifier gives an upper bound.

**Lemma 6.7.** *Let $\mathcal{N}$ be a locally finite, locally irreducible abelian network. Suppose that $\alpha.\mathbf{q}$ is a strong amplifier for $\mathcal{N}$. If $\mathbf{r}$ is locally recurrent and $\mathcal{N}$ halts on input $\mathbf{x}.\mathbf{r}$, then $[\mathbf{x}.\mathbf{r}]_a < \alpha_a$ for some $a \in A$.*

*Proof.* Suppose for a contradiction that $\mathcal{N}$ halts on input $\mathbf{x}.\mathbf{r}$ but $[\mathbf{x}.\mathbf{r}] \geq \alpha$. Write $\mathbf{u} = [\mathbf{x}.\mathbf{r}] = \alpha + \mathbf{v}$ for some $\mathbf{v} \geq \mathbf{0}$. Write $\pi_{\mathbf{v}}(\mathbf{x}.\mathbf{r}) = \mathbf{y}.\mathbf{s}$ where $\mathbf{s} = t(\mathbf{v})\mathbf{r}$. We will show that $\mathbf{y} \leq \mathbf{0}$, so there is a complete execution $w'$ for $\mathbf{x}.\mathbf{r}$ with $|w'| = \mathbf{v}$. However, by the definition of the odometer there is also a legal execution $w$ for $\mathbf{x}.\mathbf{r}$ with $|w| = \mathbf{u}$. This contradicts the least action principle (Lemma 4.5) since $\mathbf{u} - \mathbf{v} = \alpha \in \mathbb{N}^A - \{\mathbf{0}\}$.

Since $\alpha.\mathbf{q}$ is a strong amplifier we have $\alpha \triangleright \mathbf{q} = \beta.\mathbf{q}$ for some $\beta \geq \alpha$. Now by Lemma A.3(1) (which applies to each local action of $M_v$ on $Q_v$, as $\mathcal{N}$ is locally finite and locally irreducible) since $\mathbf{r}$ is locally recurrent we have $\mathbf{r} = \mathbf{m}\mathbf{q}$ for some $\mathbf{m} \in \prod M_v$. So $\mathbf{s} = (t(\mathbf{v})\mathbf{m})\mathbf{q}$, and by Lemma 5.2 it follows that $\alpha \triangleright \mathbf{s} = \beta.\mathbf{s}$. By the definition of the odometer, no messages remain after executing $w$, so $\pi_{\mathbf{u}}(\mathbf{x}.\mathbf{r}) = \mathbf{0}.\mathbf{s}'$ for some state $\mathbf{s}'$. Using $\pi_{\mathbf{u}} = \pi_\alpha \circ \pi_{\mathbf{v}}$, we have

$$\mathbf{0}.\mathbf{s}' = \pi_\alpha(\mathbf{y}.\mathbf{s}) = (\mathbf{y} + \beta - \alpha).\mathbf{s}$$

where the last equality uses Lemma 4.4(ii). Hence $\mathbf{y} = \alpha - \beta \leq \mathbf{0}$ as desired. □

In the special case of the sandpile network $\texttt{Sand}(G)$ on an undirected graph $G$ with no sink, the input $(d_v)_{v \in V}$ is a strong amplifier. Therefore if every vertex topples at least once then the network will not halt, an observation first made by Tardos [Tar88, Lemma 4].

6.2. **Laplacian matrix; Sandpilization.** Let $\mathcal{N}$ be a locally finite and locally irreducible abelian network with total alphabet $A$ and total kernel $K = \prod K_v$. Then $K_v$ has finite index in $\mathbb{Z}^{A_v}$. For each letter $a \in A_v$, let $r_a$ be the smallest positive integer such that $r_a 1_a \in K_v$.

Denote by $D$ the $A \times A$ diagonal matrix with diagonal entries $r_a$, and by $I$ the $A \times A$ idenity matrix. Let $P$ be the production matrix of $\mathcal{N}$, which is well defined by Lemma 5.9.

**Definition 6.8.** The *Laplacian* of $\mathcal{N}$ is the $A \times A$ matrix

$$L = (I - P)D$$

where $I$ is the $A \times A$ identity matrix.

**Lemma 6.9.** *$L$ has integer entries.*

*Proof.* If $\mathbf{x} \in \mathbb{N}^A$ then $D\mathbf{x} \in K$, so $(D\mathbf{x}) \triangleright \mathbf{q} = P(D\mathbf{x}).\mathbf{q}$ by the definition of the production matrix $P$. In particular, $PD\mathbf{x} \in \mathbb{N}^A$. It follows that $L\mathbf{x} \in \mathbb{N}^A$ and hence $L$ has integer entries. □

If $\mathcal{N}$ is any finite abelian network and $\mathbf{q} \in Q$, let $L_{\mathbf{q}} = (I - P_{\mathbf{q}})D_{\mathbf{q}}$ be the Laplacian of the local component $\mathcal{N}_{\mathbf{q}}$.

The class of toppling matrices, defined in §C, is one of several extensions of the notion of "positive definite" to non-symmetric matrices. Using Lemma C.3, we can rephrase Theorem 6.6 in terms of the Laplacian as follows.

**Corollary 6.10.** (Halting Criterion 2) *A finite abelian network* $\mathcal{N}$ *halts on all inputs to initial state* $\mathbf{q}$ *if and only if* $L_{\mathbf{q}}$ *is a toppling matrix.*

*Example.* Let $\mathcal{N}$ be a toppling network with three vertices $a, b, c$ and thresholds $r_a = 3,\ r_b = 4,\ r_c = 5$. For the messages passed we have:

$$\begin{aligned} a \text{ topples } &\rightarrow \quad b \text{ receives 2 chips, } c \text{ receives 2 chips.} \\ b \text{ topples } &\rightarrow \quad a \text{ receives 1 chip, } \ c \text{ receives 2 chips.} \\ c \text{ topples } &\rightarrow \quad a \text{ receives 0 chips, } b \text{ receives 2 chips.} \end{aligned}$$

Note that no vertex is a sink, and vertex $a$ is productive (it "creates" a chip each time it topples). The production matrix and Laplacian of this network are:

$$P = \begin{pmatrix} 0 & \frac{1}{4} & 0 \\ \frac{2}{3} & 0 & \frac{2}{5} \\ \frac{2}{3} & \frac{1}{2} & 0 \end{pmatrix}, \qquad L = \begin{pmatrix} 3 & -1 & 0 \\ -2 & 4 & -2 \\ -2 & -2 & 5 \end{pmatrix}.$$

Since all principal minors of $L$ are positive, $L$ is a toppling matrix. Therefore $\mathcal{N}$ halts on all inputs.

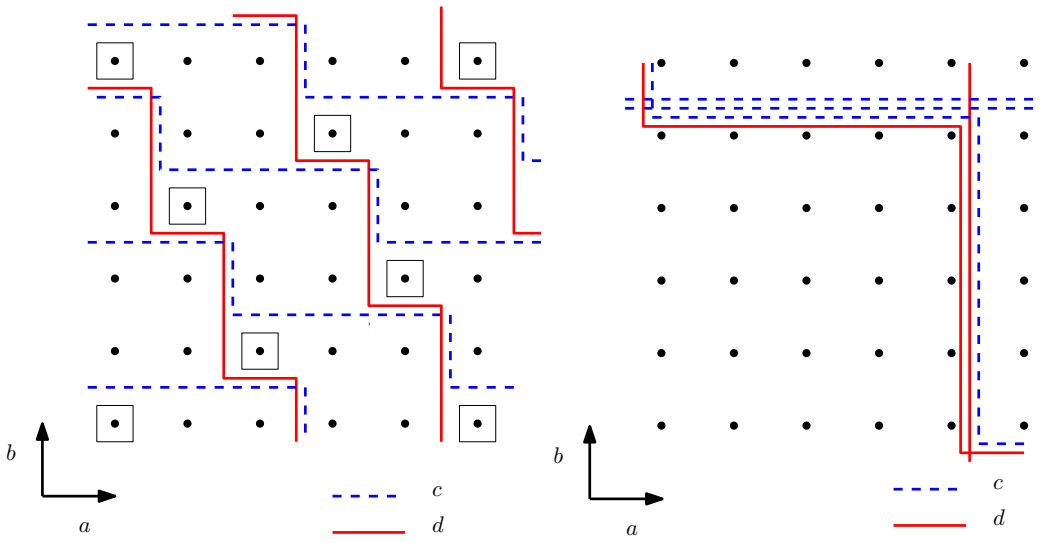


FIGURE 8. Left: State diagram of an abelian processor with 5 states, input alphabet $\{a, b\}$ and output alphabet $\{c, d\}$; the boxed dots all represent the same state. Right: State diagram of its sandpilization, which has 25 states.

We associate to $\mathcal{N}$ a toppling network $\mathcal{S}(\mathcal{N})$ called its *sandpilization*.

**Definition 6.11.** (Sandpileization) Let $\mathcal{N}$ be a locally finite and locally irreducible abelian network $\mathcal{N}$ with Laplacian $L$. The *sandpilization* $\mathcal{S}(\mathcal{N})$ of $\mathcal{N}$ is $\texttt{Topp}(L)$, the locally recurrent toppling network with Laplacian $L$.

The underlying graph of $\mathcal{S}(\mathcal{N})$ is $\Gamma$, the production graph of $\mathcal{N}$ (Definition 5.10), which may be larger than the underlying graph of $\mathcal{N}$.

In $\mathcal{S}(\mathcal{N})$ the threshold of vertex $a$ is $r_a = L_{aa}$, and toppling vertex $a$ sends $-L_{ba}$ letters to each out-neighbor $b$ of $a$. For example, if $\mathcal{N}$ is a simple rotor network $\texttt{Rotor}(G, s)$, then $\mathcal{S}(\mathcal{N})$ is the sandpile network $\texttt{Sand}(G, s)$.

Since $\mathcal{N}$ and $\mathcal{S}(\mathcal{N})$ have the same Laplacian, the following is immediate from Corollary 6.10.

**Corollary 6.12.** *Let $\mathcal{N}$ be a finite locally irreducible abelian network. Then $\mathcal{N}$ halts on all inputs if and only if $\mathcal{S}(\mathcal{N})$ halts on all inputs.*

## 7. Critical group

In this section we define an important algebraic invariant of an irreducible abelian network, its critical group. The critical group governs the "long-term" behavior of the network, i.e., its behavior on sufficiently large inputs. The name "critical group" is due to Biggs [Big99], and the construction goes back to Dhar [Dha90]. These authors studied the group $\operatorname{Crit}\mathcal{N}$ associated to the sandpile network $\mathcal{N} = \texttt{Sand}(G, s)$ with a sink. Motivated by the Neron model in arithmetic geometry, Lorenzini [Lor89, Lor91] studied an isomorphic group which has the advantage that it does not require distinguishing a sink vertex. Priezzhev, Dhar, Dhar and Krishnamurthy [PDDK96] studied the group $\operatorname{Crit}\texttt{Rotor}(G, s)$ associated to a rotor network with a sink, which turns out to be isomorphic to $\operatorname{Crit}\texttt{Sand}(G, s)$.

In this section we will define the group $\operatorname{Crit}\mathcal{N}$ in greater generality: $\mathcal{N}$ can be any finite irreducible abelian network that halts on all inputs. The monoid-theoretic approach to the sandpile group, pioneered by Babai and Toumpakari [BT10] and developed further in [C+13], turns out to be very convenient for this purpose. We will see the isomorphism $\operatorname{Crit}\texttt{Rotor}(G, s) \simeq \operatorname{Crit}\texttt{Sand}(G, s)$ as a case of a more general phenomenon: homotopic networks have isomorphic critical groups (Corollary 7.15).

A remark about terminology. The terms *critical group* and *sandpile group* are used interchangeably in the mathematical literature. In setting of abelian networks, we can make a distinction. If $\mathcal{N}$ is a finite irreducible abelian network that halts on all inputs, then it has an associated *critical group* $\operatorname{Crit}\mathcal{N}$. The *sandpile group* associated to a directed graph $G$ with marked vertex $s$ is the group $\operatorname{Crit}\mathcal{N}$ where $\mathcal{N} = \texttt{Sand}(G, s)$.

### 7.1. The global action.

If $\mathcal{N}$ halts on all inputs, then there is an action of $\mathbb{N}^A$ on $Q$ by the rule "process all letters until halting." We will denote this action by $\triangleright\triangleright$. Formally, given $\mathbf{x} \in \mathbb{N}^A$ and $\mathbf{q} \in Q$, set $\mathbf{x}_0 = \mathbf{x}$, $\mathbf{q}_0 = \mathbf{q}$ and

$$\mathbf{x}_n.\mathbf{q}_n = \mathbf{x}_{n-1} \triangleright \mathbf{q}_{n-1} \tag{9}$$

for $n \geq 1$. Since $\mathcal{N}$ halts on all inputs, there exists $N$ such that $\mathbf{x}_n = \mathbf{0}$ for all $n \geq N$. We define

$$\mathbf{x} \triangleright\triangleright \mathbf{q} := \mathbf{q}_N.$$

We extend the domain of $\triangleright\triangleright$ to $\mathbb{N}^A \times Q$ by defining for $\mathbf{y} \in \mathbb{N}^A$

$$\mathbf{x} \triangleright\triangleright (\mathbf{y}.\mathbf{q}) := (\mathbf{x}+\mathbf{y}) \triangleright\triangleright \mathbf{q}.$$

Recall that the *odometer* $[\mathbf{x}.\mathbf{q}] \in \mathbb{N}^A$ is the vector counting the total number of letters processed of each type. We have

$$[\mathbf{x}.\mathbf{q}] = \sum_{n\geq 0} \mathbf{x}_n.$$

**Lemma 7.1.** *If $w$ is a legal execution from $\mathbf{x}.\mathbf{q}$ to $\mathbf{y}.\mathbf{r}$, then $\mathbf{x} \triangleright\triangleright \mathbf{q} = \mathbf{y} \triangleright\triangleright \mathbf{r}$ and*

$$[\mathbf{x}.\mathbf{q}] = |w| + [\mathbf{y}.\mathbf{r}].$$

*Proof.* Let $w'$ be a complete legal execution for $\mathbf{y}.\mathbf{r}$. Then the concatenation $ww'$ is a complete legal execution for $\mathbf{x}.\mathbf{q}$. □

The next two lemmas are useful special cases of Lemma 7.1.

**Lemma 7.2.** *If $\mathbf{x} \triangleright \mathbf{q} = \mathbf{y}.\mathbf{r}$, then $\mathbf{x} \triangleright\triangleright \mathbf{q} = \mathbf{y} \triangleright\triangleright \mathbf{r}$ and $[\mathbf{x}.\mathbf{q}] = \mathbf{x} + [\mathbf{y}.\mathbf{r}]$.*

*Proof.* If $\mathbf{x} \triangleright \mathbf{q} = \mathbf{y}.\mathbf{r}$ then there is a legal execution $w$ from $\mathbf{x}.\mathbf{q}$ to $\mathbf{y}.\mathbf{r}$ with $|w| = \mathbf{x}$. □

**Lemma 7.3.** *For $\mathbf{x}, \mathbf{y} \in \mathbb{N}^A$ and $\mathbf{q} \in Q$ we have $(\mathbf{x}+\mathbf{y}) \triangleright\triangleright \mathbf{q} = \mathbf{y} \triangleright\triangleright (\mathbf{x} \triangleright\triangleright \mathbf{q})$, and*

$$[(\mathbf{x}+\mathbf{y}).\mathbf{q}] = [\mathbf{x}.\mathbf{q}] + [\mathbf{y}.(\mathbf{x} \triangleright\triangleright \mathbf{q})].$$

*Proof.* If $w$ is a complete legal execution for $\mathbf{x}.\mathbf{q}$, then $w$ is a legal execution from $(\mathbf{x}+\mathbf{y}).\mathbf{q}$ to $\mathbf{y}.(\mathbf{x} \triangleright\triangleright \mathbf{q})$. Since $|w| = [\mathbf{x}.\mathbf{q}]$ the result follows from Lemma 7.1. □

We record a few more identities to be used later.

**Lemma 7.4.** *Given $\mathbf{x} \in \mathbb{N}^A$ and $\mathbf{q} \in Q$, let $\mathbf{k} = [\mathbf{x}.\mathbf{q}]$. Then*

(i) $\pi_{\mathbf{k}}(\mathbf{x}.\mathbf{q}) = \mathbf{0}.(\mathbf{x} \triangleright\triangleright \mathbf{q})$
(ii) $\mathbf{k} \triangleright (\mathbf{x}.\mathbf{q}) = \mathbf{k}.(\mathbf{x} \triangleright\triangleright \mathbf{q})$
(iii) $\mathbf{x} \triangleright\triangleright \mathbf{q} = t(\mathbf{k})\mathbf{q}$
(iv) $\mathbf{x} \triangleright\triangleright (\mathbf{y} \triangleright \mathbf{q}) = (\mathbf{x}+\mathbf{y}) \triangleright\triangleright \mathbf{q}$

*Proof.* Write $s_n = \pi_{\mathbf{k}_n}(\mathbf{x}_n.\mathbf{q}_n)$, where $\mathbf{k}_n = \sum_{j\geq n} \mathbf{x}_j$ and $\mathbf{x}_n.\mathbf{q}_n$ is defined by (9). Then using $\pi_{\mathbf{k}_n} = \pi_{\mathbf{k}_{n+1}} \circ \pi_{\mathbf{x}_n}$ we have for all $n \geq 0$

$$s_n = \pi_{\mathbf{k}_{n+1}}(\mathbf{x}_n \triangleright \mathbf{q}_n) = s_{n+1}.$$

Hence $s_0 = s_N$, which proves part (i).

Part (ii) follows from (i) using Lemma 4.4(ii):

$$\mathbf{k} \triangleright (\mathbf{x}.\mathbf{q}) = \pi_{\mathbf{k}}((\mathbf{x}+\mathbf{k}).\mathbf{q}) = \mathbf{k}.(\mathbf{x} \triangleright\triangleright \mathbf{q}).$$

Part (iii) is immediate from (ii).

To prove part (iv), both states $\mathbf{x} \triangleright\triangleright (\mathbf{y} \triangleright \mathbf{q})$ and $(\mathbf{x}+\mathbf{y}) \triangleright\triangleright \mathbf{q}$ are the result of performing a complete legal exection for $(\mathbf{x}+\mathbf{y}).\mathbf{q}$. By Lemma 4.6 any two complete legal executions for $(\mathbf{x}+\mathbf{y}).\mathbf{q}$ result in the same final state. □

7.2. **Locally irreducible implies globally irreducible.** If an abelian network $\mathcal{N}$ halts on all inputs, then we can view the entire network as a single abelian processor with input alphabet $A = \sqcup A_v$ and state space $Q = \prod Q_v$. Write $\mathrm{End}\,(Q)$ for the monoid of all set maps $Q \to Q$ with the operation of composition. Analogous to the local monoids defined in §4 is a "global" object, the transition monoid of $\mathcal{N}$.

**Definition 7.5.** The *transition monoid* $M$ is the submonoid of $\mathrm{End}\,(Q)$ generated by the maps

$$\tau_a(\mathbf{q}) := 1_a \triangleright\triangleright \mathbf{q}, \qquad a \in A.$$

We say that $\mathcal{N}$ is *irreducible* if the monoid action $M \times Q \to Q$ is irreducible (§A).

For $\mathbf{x} \in \mathbb{N}^A$ we write $\tau(\mathbf{x})$ for the map $\mathbf{x} \mapsto \mathbf{x} \triangleright\triangleright \mathbf{q}$. Then

$$\tau : \mathbb{N}^A \to M \tag{10}$$

is a surjective homomorphism of monoids by Lemma 7.3. In particular, $M$ is commutative.

Next we prove a local-to-global principle for irreducibility.

**Lemma 7.6.** *Let $\mathcal{N} = \{\mathcal{P}_v\}_{v \in V}$ be an abelian network that halts on all inputs. If each processor $\mathcal{P}_v$ is irreducible, then $\mathcal{N}$ is irreducible.*

*Proof.* Let $\mathbf{q}, \mathbf{q}' \in Q$. For each $v \in V$, since $\mathcal{P}_v$ is irreducible, by Lemma A.2 there exist $m_v, m_v' \in M_v$ such that $m_v \mathbf{q}_v = m_v' \mathbf{q}_v'$. Choose, $\mathbf{x}, \mathbf{x}' \in \mathbb{N}^A$ with $t(\mathbf{x}) = m$ and $t(\mathbf{x}') = m'$. Write

$$\mathbf{x} \triangleright \mathbf{q} = \mathbf{y}.\mathbf{r}, \qquad \mathbf{x}' \triangleright \mathbf{q}' = \mathbf{y}'.\mathbf{r}'$$

where $\mathbf{r} = \mathbf{r}'$ since $\mathbf{r}_v = m_v \mathbf{q}_v = m_v' \mathbf{q}_v' = \mathbf{r}_v'$ for all $v \in V$. Then by Lemmas 7.3 and 7.4(iv),

$$\begin{aligned}
(\mathbf{x} + \mathbf{y}') \triangleright\triangleright \mathbf{q} &= \mathbf{y}' \triangleright\triangleright (\mathbf{x} \triangleright \mathbf{q}) \\
&= \mathbf{y}' \triangleright\triangleright (\mathbf{y}.\mathbf{r}) \\
&= (\mathbf{y} + \mathbf{y}') \triangleright\triangleright \mathbf{r} \\
&= \mathbf{y} \triangleright\triangleright (\mathbf{y}'.\mathbf{r}) \\
&= \mathbf{y} \triangleright\triangleright (\mathbf{x}' \triangleright \mathbf{q}') \\
&= (\mathbf{x}' + \mathbf{y}) \triangleright\triangleright \mathbf{q}'.
\end{aligned}$$

Hence $\tau(\mathbf{x}+\mathbf{y}')\mathbf{q} = \tau(\mathbf{x}'+\mathbf{y})\mathbf{q}'$, so the monoid action $M \times Q \to Q$ is irreducible. □

In light of Lemma 7.6 we will drop "locally" from "locally irreducible" when referring to a network that halts on all inputs.

7.3. **Recurrent states.** Throughout this section, we take $\mathcal{N}$ to be a finite irreducible abelian network that halts on all inputs. Since the transition monoid $M$ is a finite and commutative it has a minimal idempotent $e$, and $eM$ is a finite abelian group with identity element $e$ (see §A).

**Definition 7.7.** The *critical group* $\mathrm{Crit}\,\mathcal{N}$ is the group $eM$.

**Definition 7.8.** A state $x \in Q$ is *recurrent* if it satisfies the equivalent conditions of Lemma A.3 for the defining action $M \times Q \to Q$.

Denote by $\operatorname{Rec}\mathcal{N}$ the set of recurrent states of $\mathcal{N}$.

For example, if $\mathcal{N} = \texttt{Rotor}(G, s)$ is a simple rotor network on a directed graph $G$ with sink vertex $s$, then $\operatorname{Rec}\mathcal{N}$ can be identified with spanning trees of $G$ oriented toward $s$ [H+08, Lemma 3.16]. The critical group of $\mathcal{N}$ is isomorphic to the sandpile group $\operatorname{Crit}\texttt{Sand}(G, s)$, as observed in [PDDK96, LL09]. We will deduce this isomorphism as a special case of Theorem 7.14, below.

The following theorem generalizes [H+08, Lemmas 3.13 and 3.17], where it was shown that $\operatorname{Crit}\texttt{Rotor}(G, s)$ acts freely and transitively on the set of spanning trees of $G$ oriented toward $s$.

**Theorem 7.9.** *Let $\mathcal{N}$ be a finite irreducible abelian network that halts on all inputs. The action of the critical group on recurrent states*

$$\operatorname{Crit}\mathcal{N} \times \operatorname{Rec}\mathcal{N} \to \operatorname{Rec}\mathcal{N}$$

*is free and transitive. In particular, $\#\operatorname{Crit}\mathcal{N} = \#\operatorname{Rec}\mathcal{N}$.*

*Proof.* Let $M$ be the transition monoid of $\mathcal{N}$, and let $e$ be the minimal idempotent of $M$. Then $\operatorname{Crit}\mathcal{N} = eM$ and $\operatorname{Rec}\mathcal{N} = eQ$. The monoid action $M \times Q \to Q$ is faithful by definition and irreducible by Lemma 7.6. Hence the group action $eM \times eQ \to eQ$ is free and transitive by Theorem A.5. □

Next we formalize one way in which $\operatorname{Crit}\mathcal{N}$ and its action on $\operatorname{Rec}\mathcal{N}$ govern the "long term behavior" of $\mathcal{N}$. Let $\mu$ be a probability distribution on the total alphabet $A$. Consider the Markov chain $(\mathbf{q}_n)_{n\geq 0}$ on states of $\mathcal{N}$ where the initial state $\mathbf{q}_0$ can be arbitrary, and subsequent states are defined by

$$\mathbf{q}_{n+1} = a_n \triangleright\!\triangleright \mathbf{q}_n, \qquad n \geq 0$$

where the inputs $a_n \in A$ for $n \geq 0$ are drawn independently at random with distribution $\mu$. The next two results generalize [Dha90], where they are proved for sandpile networks.

**Corollary 7.10.** *For any $\mu$, the uniform distribution on $\operatorname{Rec}\mathcal{N}$ is stationary for the Markov chain $(\mathbf{q}_n)_{n\geq 0}$.*

*Proof.* Fix $\mathbf{q}_0 \in \operatorname{Rec}\mathcal{N}$, and let $g$ be uniform random element of $\operatorname{Crit}\mathcal{N}$. By Theorem 7.9, $g \triangleright\!\triangleright \mathbf{q}_0$ is a uniform random element of $\operatorname{Rec}\mathcal{N}$. If $a \in A$ is independent of $g$, then $\tau_a g$ is also a uniform random element of $\operatorname{Crit}\mathcal{N}$; hence if $\mathbf{q}_n$ is uniform on $\operatorname{Rec}\mathcal{N}$, then $\mathbf{q}_{n+1}$ is again uniform on $\operatorname{Rec}\mathcal{N}$. □

Since we assume $\mathcal{N}$ irreducible, its production matrix $P$ does not depend on the choice of initial state; and since we assume $\mathcal{N}$ halts on all inputs, $I - P$ is a toppling matrix by Lemma C.3 and Theorem 6.6. According to Lemma C.2, the inverse of a toppling matrix has nonnegative entries.

Our next result gives an interpretation for the entry $(I-P)^{-1}_{ab}$: it is the expected number of letters $a$ processed before the network halts, when one letter $b$ is input to a uniform recurrent state.

**Theorem 7.11.** (Expected Time To Halt) *Let* $\mathbf{q}$ *be a uniform random element of* $\operatorname{Rec}\mathcal{N}$*. Then for all* $\mathbf{x} \in \mathbb{N}^A$ *we have*

$$\mathbb{E}[\mathbf{x}.\mathbf{q}] = (I - P)^{-1}\mathbf{x}.$$

The main ingredient in the proof is the following observation.

**Lemma 7.12.** *Fix* $\mathbf{x} \in \mathbb{N}^A$ *and a locally recurrent state* $\mathbf{q} \in Q$*. Let* $\mathbf{k} = [\mathbf{x}.\mathbf{q}]$*. If* $\mathbf{x} \triangleright\triangleright \mathbf{q} = \mathbf{q}$*, then* $\mathbf{k} \in K$ *and* $\mathbf{x} = (I - P)\mathbf{k}$*.*

*Proof.* Since $\mathbf{x} \triangleright\triangleright \mathbf{q} = \mathbf{q}$, we have $t(\mathbf{k})\mathbf{q} = \mathbf{q}$ by Lemma 7.4(iii). By Lemma 5.8, since $\mathbf{q}$ is locally recurrent it follows that $\mathbf{k} \in K$. Now by Lemma 7.4(ii) and the definition (6) of the production matrix,

$$\mathbf{k}.\mathbf{q} = \mathbf{k} \triangleright (\mathbf{x}.\mathbf{q}) = (P(\mathbf{k}) + \mathbf{x}).\mathbf{q}$$

Hence $\mathbf{k} = P(\mathbf{k}) + \mathbf{x}$. □

Now consider the group homomorphism

$$\phi : \mathbb{Z}^A \to \operatorname{Crit}\mathcal{N}$$

defined on generators by $a \mapsto e\tau_a$ for each $a \in A$. For $x \in \mathbb{N}^A$ and $\mathbf{q} \in Q$ we have

$$\phi(\mathbf{x})\mathbf{q} = \left(\prod_{a\in A} (e\tau_a)^{\mathbf{x}_a}\right)\mathbf{q} = \left(\prod_{a\in A} \tau_a^{\mathbf{x}_a}\right) e\mathbf{q} = \mathbf{x} \triangleright\triangleright e\mathbf{q}. \tag{11}$$

Since $\operatorname{Crit}\mathcal{N}$ is a finite group, for any $\mathbf{x} \in \mathbb{Z}^A$ there is a positive integer $n$ such that $n\mathbf{x} \in \ker\phi$. Next we observe that input vectors in the kernel of $\phi$ act trivially on recurrent states.

**Lemma 7.13.** *If* $\mathbf{x} \in (\ker\phi) \cap \mathbb{N}^A$ *and* $\mathbf{q} \in \operatorname{Rec}\mathcal{N}$*, then* $\mathbf{x} \triangleright\triangleright \mathbf{q} = \mathbf{q}$*.*

*Proof.* For $\mathbf{q} \in \operatorname{Rec}\mathcal{N}$ we have $\mathbf{q} = e\mathbf{q}$ by Lemma A.3(4). Now by (11),

$$\mathbf{x} \triangleright\triangleright \mathbf{q} = \mathbf{x} \triangleright\triangleright e\mathbf{q} = \phi(\mathbf{x})\mathbf{q} = e\mathbf{q} = \mathbf{q}. \quad \square$$

□

*Proof of Theorem 7.11.* Given $\mathbf{x} \in \mathbb{N}^A$, choose a positive integer $n$ such that $n\mathbf{x} \in \ker\phi$, and let $\mathbf{k} = [n\mathbf{x}.\mathbf{q}]$. Then $n\mathbf{x} \triangleright\triangleright \mathbf{q} = \mathbf{q}$ by Lemma 7.13. By Lemma 7.12 it follows that $n\mathbf{x} = (I - P)\mathbf{k}$. In particular, $\mathbf{k}$ does not depend on $\mathbf{q}$.

Now by Lemma 7.3,

$$\mathbf{k} = \sum_{j=0}^{n-1} [\mathbf{x}.(j\mathbf{x} \triangleright\triangleright \mathbf{q})].$$

By Corollary 7.10, if $\mathbf{q}$ is uniform on $\operatorname{Rec}\mathcal{N}$ then $j\mathbf{x} \triangleright\triangleright \mathbf{q}$ is uniform on $\operatorname{Rec}\mathcal{N}$ for all $j \in \mathbb{N}$. Taking expectations, we obtain

$$\mathbf{k} = \mathbb{E}\mathbf{k} = n\mathbb{E}[\mathbf{x}.\mathbf{q}].$$

Dividing by $n$ yields the result. □

7.4. **Generators and relations.** Next we turn to the problem of describing the critical group by generators and relations. The group homomorphism $\phi : \mathbb{Z}^A \to \operatorname{Crit}\mathcal{N}$ sending $a \mapsto e\tau_a$ is surjective, since $\operatorname{Crit}\mathcal{N} = eM$ and $M$ is generated by $\{\tau_a\}_{a\in A}$. To find the kernel of $\phi$, recall the *production map*

$$P : K \to \mathbb{Z}^A$$

defined in Lemma 5.6, where $K$ is the total kernel (Definition 5.4). Write $I$ for the inclusion $K \hookrightarrow \mathbb{Z}^A$.

**Theorem 7.14.** *The natural map $\phi : \mathbb{Z}^A \to \operatorname{Crit}\mathcal{N}$ induces an isomorphism of abelian groups*

$$\operatorname{Crit}\mathcal{N} \simeq \mathbb{Z}^A/(I-P)K.$$

*Proof.* We must show that $\ker\phi = (I-P)K$. Fix $\mathbf{q} \in \operatorname{Rec}\mathcal{N}$. For any $\mathbf{k} \in K \cap \mathbb{N}^A$ we have

$$\mathbf{k} \triangleright \mathbf{q} = P(\mathbf{k}).\mathbf{q}$$

so $\mathbf{k} \triangleright\triangleright \mathbf{q} = P(\mathbf{k}) \triangleright\triangleright \mathbf{q}$ by Lemma 7.2. Hence $\phi(\mathbf{k})\mathbf{q} = \phi(P(\mathbf{k}))\mathbf{q}$ by (11). By Theorem 7.9 the action of $\operatorname{Crit}\mathcal{N}$ on $\operatorname{Rec}\mathcal{N}$ is free, so we conclude $\phi(\mathbf{k}) = \phi(P(\mathbf{k}))$. This shows that $(I-P)\mathbf{k} \in \ker\phi$ for all $\mathbf{k} \in K \cap \mathbb{N}^A$. By Lemma 5.5, $K$ is generated as a group by $K \cap \mathbb{N}^A$, so $(I-P)K \subset \ker\phi$.

To show the reverse inclusion, given $\mathbf{x} \in (\ker\phi) \cap \mathbb{N}^A$ we have $\phi(\mathbf{x}) = e$ and hence $\mathbf{x} \triangleright\triangleright e\mathbf{q} = e\mathbf{q}$ for all $\mathbf{q} \in Q$ by (11). By Lemma 7.12, $\mathbf{x} = (I-P)\mathbf{k}$ where $\mathbf{k} = [\mathbf{x}.e\mathbf{q}] \in K$. It follows that $(\ker\phi) \cap \mathbb{N}^A \subset (I-P)K$. Since $\operatorname{Crit}\mathcal{N}$ is finite, $\ker\phi$ is a full rank subgroup of $\mathbb{Z}^A$, so it is generated as a group by $(\ker\phi) \cap \mathbb{N}^A$, which shows that $\ker\phi \subset (I-P)K$. □

By Theorem 7.14 the critical group depends only on the total kernel $K$ and production matrix $P$. It follows that homotopic networks (Definition 5.12) have isomorphic critical groups.

**Corollary 7.15.** *If $\mathcal{N} \approx \mathcal{N}'$, then $\operatorname{Crit}\mathcal{N} \simeq \operatorname{Crit}\mathcal{N}'$.*

For instance, $\texttt{Rotor}(G,s) \approx \texttt{Sand}(G,s)$ for a strongly connected graph $G$. In both cases $K = \prod_v (d_v\mathbb{Z})$ and $P_{uv} = d_{uv}/d_u$ where $d_{uv}$ is the number of directed edges from $v$ to $u$ in $G$. Thus Corollary 7.15 generalizes the isomorphism $\operatorname{Crit}\texttt{Rotor}(G,s) \simeq \operatorname{Crit}\texttt{Sand}(G,s)$ between the rotor and sandpile groups of a graph.

Next we relate the critical group to the cokernel of the Laplacian $L = (I-P)D$. Recall $D$ is the $A \times A$ diagonal matrix with diagonal entries

$$r_a = \min\{m \geq 1 \,:\, m1_a \in K\}.$$

**Definition 7.16.** An abelian network $\mathcal{N}$ is *rectangular* if its total kernel is $K = \prod_{a\in A}(r_a\mathbb{Z})$ (that is, $K$ is a rectangular sublattice of $\mathbb{Z}^A$).

**Corollary 7.17.** *The natural map $\mathbb{Z}^A \to \operatorname{Crit}\mathcal{N}$ induces a surjective group homomorphism*

$$\phi : \mathbb{Z}^A/L\mathbb{Z}^A \twoheadrightarrow \operatorname{Crit}\mathcal{N}.$$

*If $\mathcal{N}$ is rectangular, then $\phi$ is an isomorphism.*

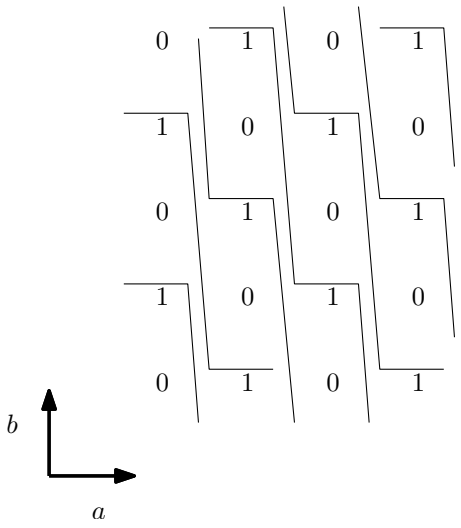


FIGURE 9. State diagram of an abelian processor with two states $0, 1$ and two inputs $a, b$. Its kernel is $\{(m, n) \in \mathbb{Z}^2 : m + n \equiv 0 \pmod 2\}$, so it is not rectangular. Each line crossed results in output of one letter $c$.

*Proof.* By definition, $D\mathbb{Z}^A \subset K$ with equality if $\mathcal{N}$ is rectangular. Since $L = (I - P)D$, we have $L\mathbb{Z}^A \subset (I - P)K$ with equality if $\mathcal{N}$ is rectangular. □

Note that any unary network (and in particular any toppling network) is rectangular.

**Corollary 7.18.** $\operatorname{Crit} \mathcal{S}(\mathcal{N}) \simeq \mathbb{Z}^A/L\mathbb{Z}^A$ *and* $\operatorname{Crit} \mathcal{S}(\mathcal{N}) \twoheadrightarrow \operatorname{Crit} \mathcal{N}$.

*Example.* To see that $\operatorname{Crit} \mathcal{S}(\mathcal{N}) \twoheadrightarrow \operatorname{Crit} \mathcal{N}$ need not be an isomorphism, consider the following non-rectangular network with vertices $i, j$ where $j$ is a sink. Let $Q_i = \{0, 1\}$, $A_i = \{a, b\}$, $A_j = \{c\}$,

$$T_i(q, a) = T_i(q, b) = q + 1 \pmod 2$$

and

$$\begin{aligned} T_{(i,j)}(0, a) &= c \\ T_{(i,j)}(1, a) &= cc \\ T_{(i,j)}(0, b) &= \epsilon \\ T_{(i,j)}(1, b) &= c. \end{aligned}$$

Figure 9 shows the state diagram of $\mathcal{P}_i$.

The production matrix and Laplacian of this network, with rows and columns indexed by $a, b, c$ in that order, are:

$$P = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{3}{2} & \frac{1}{2} & 0 \end{pmatrix} \qquad L = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ -3 & -1 & 1 \end{pmatrix}$$

So we find that $\operatorname{Crit} \mathcal{S}(\mathcal{N}) = \mathbb{Z}^3/L\mathbb{Z}^3 = \mathbb{Z}_2 \times \mathbb{Z}_2$. On the other hand, $\tau_a = \tau_b$ in $\operatorname{Crit} \mathcal{N}$ (both send $0 \mapsto 1 \mapsto 0$) so $\operatorname{Crit} \mathcal{N} = \mathbb{Z}_2$.

Processor $\mathcal{P}_i$ in this example has another curious feature: The input words $aa, ab, bb$ to state 0 all result in the same sequence of states $0, 1, 0$ yet they produce different outputs (*ccc*,*cc*,*c* respectively).

Now we turn to the problem of counting recurrent states, or equivalently (by Theorem 7.9) finding the order of the critical group. Let

$$\iota = [K : D\mathbb{Z}^A]$$

be the index of $D\mathbb{Z}^A$ as a subgroup of $K$. Recalling that $K = \prod_{v \in V} K_v$, we can write $\iota = \prod_{v \in V} \iota_v$ as a product of local indices

$$\iota_v = [K_v : D_v \mathbb{Z}^{A_v}]$$

where $D_v \mathbb{Z}^{A_v} := \prod_{a \in A_v} (r_a \mathbb{Z})$. Note $\iota = 1$ if and only if $\mathcal{N}$ is rectangular.

**Theorem 7.19.**

$$\#\operatorname{Rec} \mathcal{N} = \#\operatorname{Crit} \mathcal{N} = \frac{\det L}{\iota}.$$

*Proof.* The first equality follows from Theorem 7.9. For the second, we have by Theorem 7.14

$$\begin{aligned} \#\operatorname{Crit} \mathcal{N} = [\mathbb{Z}^A : (I-P)K] &= \frac{[\mathbb{Z}^A : L\mathbb{Z}^A]}{[(I-P)K : L\mathbb{Z}^A]} \\ &= \frac{|\det L|}{[(I-P)K : (I-P)D\mathbb{Z}^A]}. \end{aligned}$$

Since $\mathcal{N}$ halts on all inputs, $\det L > 0$ by Corollary 6.10. Likewise, $I - P$ has full rank by Theorem 6.6, so the denominator equals $\iota$. □

*Example.* Let $\mathcal{N}$ be a height arrow network (§3.6) with sink $s$. This is a unary network, so it is rectangular. Each local component has the same production matrix $p_{uv} = d_{uv}/d_v$, and $D$ has diagonal entries $r_s = 1$ and $r_v = \operatorname{lcm}(\tau_v, d_v)$ for $v \neq s$, so the Laplacian is $L = (I-P)D = \Delta^s \Lambda$, where $\Delta^s$ is the damped graph Laplacian and $\Lambda$ is the diagonal matrix with diagonal entries $\lambda_s = 1$ and $\lambda_v = r_v/d_v$ for $v \neq s$. By Corollary 7.17, each local component $\mathcal{N}_{\mathbf{q}}$ has $\operatorname{Crit} \mathcal{N}_{\mathbf{q}} \simeq \mathbb{Z}^V/L\mathbb{Z}^V$, and the number of recurrent states in each local component is $\det L = (\prod \lambda_v) \det \Delta^s$. The number of local components is $\prod g_v$, where $g_v = \gcd(\tau_v, d_v)$. Noting that $\lambda_v = \tau_v/g_v$, the total number of recurrent states is $(\prod \tau_v) \det \Delta^s$. This formula agrees with [DR04, Proposition 3.9]; note that the last line there has a misprint: $d_i$ should be $\tau_i$.

**Theorem 7.20.** *Let $\mathcal{N}$ be a finite irreducible abelian network that halts on all inputs. Let $P$ and $\Gamma$ be the production matrix and production graph of $\mathcal{N}$. The following are equivalent.*

1. *Every locally recurrent state of $\mathcal{N}$ is recurrent.*
2. *Every state of $\mathcal{S}(\mathcal{N})$ is recurrent.*
3. *$\Gamma$ has no directed cycles.*

(4) *$P$ is nilpotent.*

*Proof.* Recall from §5.2 the action of $\mathbb{Z}^{A_v}$ on $e_vQ_v$ for each vertex $v$. Since $\mathcal{N}$ is irreducible this action is transitive, and its kernel is $K_v$. So the number of locally recurrent states of $\mathcal{N}$ is

$$\prod_{v\in V}\#(e_vQ_v)=\prod_{v\in V}\#(\mathbb{Z}^{A_v}/K_v)=[\mathbb{Z}^A:K].$$

By Theorem 7.19, the number of recurrent states of $\mathcal{N}$ is

$$\frac{\det L}{[K:D\mathbb{Z}^A]}=\frac{\det L}{\det D}[\mathbb{Z}^A:K]$$

Thus (1) holds if and only if $\det L=\det D$.

All states of the sandpilization $\mathcal{S}(\mathcal{N})$ are locally recurrent, and $\mathcal{S}(\mathcal{N})$ has the same matrices $L$, $D$ as $\mathcal{N}$, so (1) $\Rightarrow$ (2).

If $\Gamma$ has a directed cycle $a_1\to\cdots\to a_m\to a_1$, then any state of $\mathcal{S}(\mathcal{N})$ with 0 chips at each of $a_1,\dots,a_m$ cannot be recurrent: in the burning algorithm [Dha90], if $a_i$ was the last vertex on the cycle to topple, then $a_{i+1}$ has at least one chip. This shows (2) $\Rightarrow$ (3).

Let $\gamma_k=\mathbf{1}^TP^k\mathbf{1}$ be the number of directed paths of $k$ edges in $\Gamma$. If $\Gamma$ has no directed cycles, then any such path has distinct vertices, so $\gamma_k=0$ for $k=\#A$ (recall the vertex set of $\Gamma$ is $A$), whence $P^k=\mathbf{0}$. This shows (3) $\Rightarrow$ (4).

If $\lambda_1,\dots,\lambda_n$ are the eigenvalues of $P$ with multiplicity, then

$$\frac{\det L}{\det D}=\det(I-P)=\prod_{i=1}^n(1-\lambda_i).$$

If $P$ is nilpotent then $\lambda_1=\cdots=\lambda_n=0$, so $\det L=\det D$. Hence (4) $\Rightarrow$ (1). □

## 8. Burning test

Dhar's burning test [Dha90] is an efficient algorithm for determining whether a state of $\mathtt{Sand}(G,s)$ recurrent. In its simplest form it applies to Eulerian directed graphs $G$ (strongly connected, indegree$(v)$ = outdegree$(v)$ for $v\in V$); see [H$^+$08, §4]. Speer [Spe93] treated general directed graphs. A variant of Speer's algorithm can be found in [PPW11]. In Theorem 8.7 we give a test for recurrence that applies to any finite irreducible abelian network $\mathcal{N}$ that halts on all inputs.

**Definition 8.1.** A *burning element* for $\mathcal{N}$ is a vector $\beta\in\mathbb{N}^A$ such that for $\mathbf{q}\in Q$

$$\mathbf{q}\in\operatorname{Rec}\mathcal{N}\quad\Leftrightarrow\quad\beta\triangleright\!\triangleright\mathbf{q}=\mathbf{q}.$$

To see that burning elements exist, we can use Lemma A.3(4): $\mathbf{q}\in\operatorname{Rec}\mathcal{N}$ if and only if $\mathbf{q}=e\mathbf{q}$, where $e$ is the minimal idempotent of the transition monoid $M$. Recalling the map $\tau$ of (10), any $\beta\in\mathbb{N}^A$ such that $\tau(\beta)=e$ is a burning element. However, such elements are typically large. The power of the burning test derives from the fact that one can often identify a small burning element $\beta$ to reduce the running time $[\beta.\mathbf{q}]$.

In the classical setting of $\mathcal{N}=\mathtt{Sand}(G,s)$, there is a pointwise minimal burning element $\beta$. In the case that $G$ is Eulerian, the usual formula for the burning

element is $\beta_v = d_{vs}$, the number of edges into $v$ from the sink $s$. The special role of the sink is undesirable for us (since in general, an abelian network need not have a sink in order to halt on all inputs). To remove it, notice that

$$\beta = L\mathbf{1}$$

where $L$ is the Laplacian of Definition 6.8. (To make the connection to the graph Laplacian $\Delta_G$, we have $L = \Delta_{G'}$ where $G'$ is the graph obtained from $G$ by removing all outgoing edges from $s$. Since $G$ is Eulerian we have $\Delta_G\mathbf{1} = \mathbf{0}$, so $L\mathbf{1} = -\Delta_G\mathbf{1}_s = \beta$.)

In the case of a general directed graph, $L\mathbf{1}$ may have negative entries. Speer [Spe93] observed that there is a pointwise smallest vector $\mathbf{y} \geq \mathbf{1}$ such that $L\mathbf{y} \geq \mathbf{0}$, and showed that $\beta = L\mathbf{y}$ is a burning element.

8.1. **Time to halt.** In this section we show that the production matrix of $\mathcal{N}$ determines its running time on any input up to an additive constant (Theorem 8.3). In the special case of rotor networks, additive error bounds of this type appear in the work of Cooper and Spencer [CS06] and Holroyd and Propp [HP10].

Let $\mathcal{N}$ be a finite irreducible abelian network that halts on all inputs. Let $K$ be its total kernel and $P : K \to \mathbb{Z}^A$ its production map. Write $I$ for the inclusion $K \hookrightarrow \mathbb{Z}^A$.

**Lemma 8.2.** *Suppose $\mathbf{k} \in K$ satisfies $P\mathbf{k} \leq \mathbf{k}$. If $\mathbf{q} \in \operatorname{Rec}\mathcal{N}$, then*

$$(I - P)\mathbf{k} \triangleright\triangleright \mathbf{q} = \mathbf{q}.$$

*Proof.* Let $\mathbf{x} = (I - P)\mathbf{k} \in \mathbb{N}^A$. By Theorem 7.14, if $\mathbf{k} \in K$ then $\mathbf{x} \in \ker\phi$. The conclusion now follows from Lemma 7.13. □

*Remark.* Recalling that $I - P$ is injective (Theorem 6.6) and that the matrix of $(I - P)^{-1}$ has nonnegative entries (Theorem 7.11), we see that the condition $P\mathbf{k} \leq \mathbf{k}$ in the above lemma implies $\mathbf{k} \geq \mathbf{0}$.

In the next theorem and its proof, we write $(I-P)^{-1}\mathbf{x}$ for the image of $\mathbf{x} \in \mathbb{Z}^A$ under this matrix (even when $\mathbf{x} \notin (I - P)K$).

For $\mathbf{u} \in \mathbb{Z}^A$ write $\|u\|_\infty = \max_{a\in A} |u_a|$.

**Theorem 8.3.** (Time to halt) *Let $\mathcal{N}$ be an irreducible finite abelian network that halts on all inputs. There is a constant $C$ depending only on $\mathcal{N}$, such that for all $\mathbf{x} \in \mathbb{N}^A$ and all $\mathbf{q} \in Q$,*

$$\big\| [\mathbf{x}.\mathbf{q}] - (I - P)^{-1}\mathbf{x} \big\|_\infty \leq C.$$

*Proof.* Since $(I - P)K$ has full rank in $\mathbb{Z}^A$, there is a constant $c$ such that for any $\mathbf{x} \in \mathbb{N}^A$ there exists $\mathbf{y} = (I - P)\mathbf{k} \in (I - P)K$ such that $\mathbf{0} \leq \mathbf{y} - \mathbf{x} \leq c\mathbf{1}$. Since $(I - P)^{-1}$ has nonnegative entries,

$$\mathbf{0} \leq \mathbf{k} - (I - P)^{-1}\mathbf{x} \leq (I - P)^{-1}c\mathbf{1}.$$

Therefore it suffices to bound $\|[\mathbf{x}.\mathbf{q}] - \mathbf{k}\|_\infty$.

To do this, fix $\mathbf{z} \in \mathbb{N}^A$ with $\tau(\mathbf{z}) = e$. We will show

$$\|[\mathbf{x}.e\mathbf{q}] - \mathbf{k}\|_\infty \leq C_1 \tag{12}$$

and

$$\|[\mathbf{x}.e\mathbf{q}] - [\mathbf{x}.\mathbf{q}]\|_\infty \leq C_2 \tag{13}$$

where $C_1 = \max_{\mathbf{q}' \in Q} \|[c\mathbf{1}.\mathbf{q}']\|_\infty$ and $C_2 = \max_{\mathbf{q}' \in Q} \|[\mathbf{z}.\mathbf{q}']\|_\infty$.

Since $\mathbf{y} \geq \mathbf{0}$, we have $P\mathbf{k} \leq \mathbf{k}$, so $\mathbf{y} \triangleright\!\triangleright e\mathbf{q} = e\mathbf{q}$ by Lemma 8.2. Hence by Lemmas 7.12 and 7.3

$$\mathbf{k} = [\mathbf{y}.e\mathbf{q}] = [\mathbf{x}.e\mathbf{q}] + [(\mathbf{y} - \mathbf{x}).(\mathbf{x} \triangleright\!\triangleright e\mathbf{q})]$$

which shows (12). Two more applications of Lemma 7.3 give

$$[\mathbf{z}.\mathbf{q}] + [\mathbf{x}.e\mathbf{q}] = [(\mathbf{x} + \mathbf{z}).\mathbf{q}] = [\mathbf{x}.\mathbf{q}] + [\mathbf{z}.(\mathbf{x} \triangleright\!\triangleright \mathbf{q})]$$

which shows (13). □

By Theorem 8.3 and the triangle inequality,

$$\|[\mathbf{x}.\mathbf{q}] - [\mathbf{x}.\mathbf{r}]\|_\infty \leq 2C$$

for all $\mathbf{x} \in \mathbb{N}^A$ and all $\mathbf{q}, \mathbf{r} \in Q$. We will use this bound in the next section.

8.2. **Large inputs.** In this section we show that states obtained from sufficiently large inputs to $\mathcal{N}$ must be recurrent. We consider two interpretations of "large": either the number of letters in the input $\mathbf{x}$ is large (Lemma 8.4), or the odometer $[\mathbf{x}.\mathbf{q}]$ is large (Lemmas 8.5 and 8.6).

**Lemma 8.4.** *Let $\mathbf{z} \in \mathbb{N}^A$ be such that $\tau(\mathbf{z}) = e$. If $\mathbf{x} \geq \mathbf{z}$ then $\mathbf{x} \triangleright\!\triangleright \mathbf{q}$ is recurrent for all $\mathbf{q} \in Q$.*

*Proof.* Writing $\mathbf{x} = \mathbf{y} + \mathbf{z}$ for $\mathbf{y} \in \mathbb{N}^A$, we have

$$\mathbf{x} \triangleright\!\triangleright \mathbf{q} = \mathbf{z} \triangleright\!\triangleright (\mathbf{y} \triangleright\!\triangleright \mathbf{q}) = em\mathbf{q}.$$

where $m = \tau(\mathbf{y})$. By Lemma A.3(3) the state $em\mathbf{q}$ is recurrent. □

Given $\mathbf{q}, \mathbf{r} \in Q$ we say that $\mathbf{q}$ is *locally accessible* from $\mathbf{r}$ if $\mathbf{q} = \mathbf{m}\mathbf{r}$ for some $\mathbf{m} \in \prod_{v \in V} M_v$. In particular, if there is an execution (say $w$) from $\mathbf{x}.\mathbf{r}$ to $\mathbf{y}.\mathbf{q}$, then $\mathbf{q}$ is locally accessible from $\mathbf{r}$ (namely $\mathbf{q} = t(|w|)\mathbf{r}$).

**Lemma 8.5.** *If $\mathbf{x} \triangleright\!\triangleright \mathbf{r} = \mathbf{r}$ and $[\mathbf{x}.\mathbf{q}] \geq \mathbf{1}$ for all states $\mathbf{q}$ locally accessible from $\mathbf{r}$, then $\mathbf{r}$ is recurrent.*

*Proof.* Let $\mathbf{z} \in \mathbb{N}^A$ be such that $\tau(\mathbf{z}) = e$. We will find a $\mathbf{y} \geq \mathbf{z}$ and a state $\mathbf{q}$ such that $\mathbf{r} = \mathbf{y} \triangleright\!\triangleright \mathbf{q}$.

Let $n = \sum_{a \in A} \mathbf{z}_a$. Fix any sequence $a_1, \dots, a_n$ such that $\mathbf{z} = \sum_{i=1}^n \mathbf{1}_{a_i}$. Let $\mathbf{q}^0 = \mathbf{r}$ and inductively define states $\mathbf{q}^1, \dots, \mathbf{q}^n$, all locally accessible from $\mathbf{r}$, as follows.

For each $i = 1, \dots, n$, since $\mathbf{q}^{i-1}$ is locally accessible from $\mathbf{r}$ we have $[\mathbf{x}.\mathbf{q}^{i-1}]_{a_i} \geq 1$, so there is a legal execution $w_i$ from $\mathbf{x}.\mathbf{q}^{i-1}$ to some state $\mathbf{y}^i.\mathbf{q}^i$ satisfying $\mathbf{y}^i_{a_i} \geq 1$. The concatenation $w_1 \cdots w_n$ is a legal execution from $(n\mathbf{x}).\mathbf{r}$ to $\mathbf{y}.\mathbf{q}^n$ with $\mathbf{y} = \mathbf{y}^1 + \cdots + \mathbf{y}^n \geq \mathbf{z}$. Hence by Lemma 7.1,

$$\mathbf{r} = (n\mathbf{x}) \triangleright\!\triangleright \mathbf{r} = \mathbf{y} \triangleright\!\triangleright \mathbf{q}^n$$

and the right side is recurrent by Lemma 8.4. □

For $\mathbf{u} \in \mathbb{N}^A$ and $\mathbf{q} \in Q$, define

$$R_{\mathbf{u}}(\mathbf{q}) = \left\{ \mathbf{x} \triangleright\!\triangleright \mathbf{q} \;\middle|\; \mathbf{x} \in \mathbb{N}^A,\, [\mathbf{x}.\mathbf{q}] \geq \mathbf{u} \right\}.$$

and write $R_{\mathbf{u}}$ for the set of states $\mathbf{q} \in Q$ such that $\mathbf{q} \in R_{\mathbf{u}}(\mathbf{q})$.

We have defined recurrent states in Lemma A.3 by a list of equivalent monoid-theoretic properties. Now we can add to this list a characterization that is specific to abelian networks. According to the next lemma, a state $\mathbf{q}$ is recurrent if and only if it there exist inputs with arbitrarily large odometers that fix $\mathbf{q}$.

**Lemma 8.6.**

$$\operatorname{Rec} \mathcal{N} = \bigcap_{\mathbf{u} \in \mathbb{N}^A} R_{\mathbf{u}}.$$

*Proof.* Suppose that $\mathbf{r} \in \bigcap R_{\mathbf{u}}$. Then for any $\mathbf{u} \in \mathbb{N}^A$ there is an input $\mathbf{x} \in \mathbb{N}^A$ with $\mathbf{r} = \mathbf{x} \triangleright\!\triangleright \mathbf{r}$ and $[\mathbf{x}.\mathbf{r}] \geq \mathbf{u}$. Take $\mathbf{u} = (2C+1)\mathbf{1}$. By Theorem 8.3 it follows that $[\mathbf{x}.\mathbf{q}] \geq \mathbf{1}$ for all $\mathbf{q} \in Q$. Now from Lemma 8.5 it follows that $\mathbf{r}$ is recurrent.

It remains to show that $\operatorname{Rec} \mathcal{N} \subset R_{\mathbf{u}}$ for all $\mathbf{u} \in \mathbb{N}^A$. Since $\operatorname{Crit} \mathcal{N}$ is a finite group, the kernel of $\phi : \mathbb{Z}^A \to \operatorname{Crit} \mathcal{N}$ has nonempty intersection with $\mathbb{N}^A + \mathbf{u}$. Let $\mathbf{x}$ be a point in this intersection. Then for any $\mathbf{q} \in \operatorname{Rec} \mathcal{N}$ we have $\mathbf{x} \triangleright\!\triangleright \mathbf{q} = \mathbf{q}$ by Lemma 7.13. Since $[\mathbf{x}.\mathbf{q}] \geq \mathbf{x} \geq \mathbf{u}$, it follows that $\mathbf{q} \in R_{\mathbf{u}}$. □

Finally, we show that if $\mathbf{k} \geq \mathbf{1}$ then the converse to Lemma 8.2 holds.

**Theorem 8.7.** (Burning Test) *Let $\mathbf{k} \in K$ be such that $\mathbf{k} \geq \mathbf{1}$ and $P\mathbf{k} \leq \mathbf{k}$. Then $\mathbf{q} \in Q$ is recurrent if and only if $(I-P)\mathbf{k} \triangleright\!\triangleright \mathbf{q} = \mathbf{q}$.*

*Proof.* If $\mathbf{q}$ is recurrent, then $(I-P)\mathbf{k} \triangleright\!\triangleright \mathbf{q} = \mathbf{q}$ by Lemma 8.2.

For the converse, let $\mathbf{x} = (I-P)\mathbf{k}$ and $\mathbf{u} = [\mathbf{x}.\mathbf{q}]$. Suppose that $\mathbf{x} \triangleright\!\triangleright \mathbf{q} = \mathbf{q}$. Then $\mathbf{x} = (I-P)\mathbf{u}$ by Lemma 7.12. Since $I-P$ is injective, we obtain $\mathbf{u} = \mathbf{k}$. Now for any $n \in \mathbb{N}$ we have $n\mathbf{x} \triangleright\!\triangleright \mathbf{q} = \mathbf{q}$ and $[n\mathbf{x}.\mathbf{q}] = n\mathbf{k}$. Hence

$$\mathbf{q} \in \bigcap_{n \geq 1} R_{n\mathbf{k}}.$$

Since $\mathbf{k} \geq \mathbf{1}$ the right side equals $\bigcap_{\mathbf{u} \in \mathbb{N}^A} R_{\mathbf{u}}$, which equals $\operatorname{Rec} \mathcal{N}$ by Lemma 8.6. □

*Remark.* The preceding theorem can be improved slightly by weakening $\mathbf{k} \geq \mathbf{1}$ to $\mathbf{k} \geq \mathbf{1}_C$, where $C$ is the set of all $a \in A$ that lie on a directed cycle of the production graph $\Gamma$ (Definition 5.10). The reason is that if $a \in A$ does not lie on a directed cycle, then $\{a\}$ is a strong component of $\Gamma$, and all locally recurrent states of the strong component $\mathcal{N}^a$ are recurrent. Writing $a^\omega$ for the minimal idempotent of the transition monoid of $\mathcal{N}^a$, one checks that $e = \prod_{a \in C} a^\omega$ and hence that $\operatorname{Rec} \mathcal{N} = \bigcap_{a \in C} \operatorname{Rec} \mathcal{N}^a$.

8.3. **Finding a burning element.** In this section we show that there is always a burning element $\beta$ satisfying $\mathbf{0} \leq \beta \leq \mathbf{r}$, where $\mathbf{r}_a$ is the smallest positive integer such that $\mathbf{r}_a \mathbf{1}_a \in K$.

**Definition 8.8.** A *burning odometer* is a vector $\mathbf{k}$ satisfying

$$(I-P)\mathbf{k} \geq \mathbf{0}, \qquad \mathbf{k} \geq \mathbf{1}, \qquad \mathbf{k} \in K. \tag{14}$$

The corresponding *burning element* is $\beta = (I-P)\mathbf{k}$.

According to Theorem 8.7, to check whether $\mathbf{q}$ is recurrent it suffices to find a burning odometer $\mathbf{k}$ and then compute $\beta \triangleright\triangleright \mathbf{q}$, where $\beta = (I - P)\mathbf{k}$. The proof also shows that if $\mathbf{q}$ is recurrent, then the local run time of this computation is $[\beta.\mathbf{q}] = \mathbf{k}$. Therefore we are interested in finding a burning odometer $\mathbf{k}$ as small as possible.

Recall that $K \subset D\mathbb{Z}^A$, with equality if $\mathcal{N}$ is rectangular. For a sandpile network $\mathcal{N} = \mathtt{Sand}(G, s)$, a burning odometer is $\mathbf{k} = D\mathbf{y}$ where $\mathbf{y}$ is the "burning script" of [Spe93]. More generally, if $\mathcal{N}$ is rectangular, then writing $\mathbf{k} = D\mathbf{y}$, (14) is equivalent to

$$L\mathbf{y} \geq \mathbf{0}, \qquad \mathbf{y} \geq \mathbf{1}, \qquad \mathbf{y} \in \mathbb{Z}^A \tag{15}$$

where $L = (I - P)D$ is the Laplacian of $\mathcal{N}$. Setting $\mathbf{x} = \mathbf{y} - \mathbf{1}$ we find that (14) is equivalent to

$$L\mathbf{x} \geq -L\mathbf{1}, \qquad \mathbf{x} \geq \mathbf{0}, \qquad \mathbf{x} \in \mathbb{Z}^A. \tag{16}$$

Minimizing $\mathbf{1}^T\mathbf{x}$ subject to these constraints is an integer program of the class solved by toppling networks (Theorem 4.11). Specifically, consider the sandpilization $\mathcal{S}(\mathcal{N}) = \mathtt{Topp}(L)$, which has vertex set $A$. In this network vertex $a$ has toppling threshold $\mathbf{r}_a$, where $\mathbf{r}$ is the vector of diagonal entries of the diagonal matrix $D$. For $\mathbf{q} \in \mathbb{Z}^A$, write $\mathbf{q}^\circ = \mathbf{q}^+ \triangleright\triangleright_{\mathcal{S}(\mathcal{N})} \mathbf{q}^-$ for the stabilization of $\mathbf{q}$ in $\mathcal{S}(\mathcal{N})$.

**Corollary 8.9.** ($\mathcal{S}(\mathcal{N})$ computes a minimal burning element for $\mathcal{N}$) *Let $\mathcal{N}$ be an irreducible rectangular network that halts on all inputs. Then $\mathcal{N}$ has a pointwise smallest burning odometer $\mathbf{k} = D\mathbf{y}$. The corresponding burning element is given by*

$$\beta = L\mathbf{y} = \mathbf{r} - \mathbf{1} - (\mathbf{r} - \mathbf{1} - L\mathbf{1})^\circ.$$

*Moreover, $\mathbf{0} \leq \beta \leq \mathbf{r}$.*

*Proof.* Let $\mathbf{q} = \mathbf{r} - \mathbf{1} - L\mathbf{1}$, and for $a \in A$ let $\mathbf{x}_a$ be the number of times $a$ topples on input $\mathbf{q}^+.\mathbf{q}^-$ to $\mathcal{S}(\mathcal{N})$. By Corollary 4.11, $\mathbf{x}$ is the pointwise smallest vector satisfying (16), and $\mathbf{q}^\circ = \mathbf{q} - L\mathbf{x}$. Setting $\mathbf{y} = \mathbf{x} + \mathbf{1}$, we conclude that $\mathbf{k} = D\mathbf{y}$ is the pointwise smallest vector satisfying (14), and

$$\beta = (I - P)\mathbf{k} = L(\mathbf{x} + \mathbf{1}) = L\mathbf{1} + \mathbf{q} - \mathbf{q}^\circ = \mathbf{r} - \mathbf{1} - \mathbf{q}^\circ.$$

Noting that $L\mathbf{1} \leq \mathbf{r}$, we have $-\mathbf{1} \leq \mathbf{q}^\circ \leq \mathbf{r} - \mathbf{1}$ and hence $\mathbf{0} \leq \beta \leq \mathbf{r}$. □

*Remark.* If we replace $\mathbf{1}$ by $\mathbf{1}_C$ in (14), then we obtain a slightly smaller burning element $\beta$ which satisfies $\mathbf{0} \leq \beta \leq \mathbf{r} - \mathbf{1}$. To compute this smaller element using $\mathcal{S}(\mathcal{N})$, take $\mathbf{q} = \mathbf{r} - \mathbf{1} - L\mathbf{1}_C$.

In practice, it is more direct to find the burning element by "untopplings" instead of topplings, which amounts to the following procedure to find the minimal $\mathbf{y}$ satisfying (15).

**Procedure 8.10.** *Start with $\mathbf{y} = \mathbf{1}$. If $L\mathbf{y} \geq \mathbf{0}$, then stop. Otherwise, choose some $a$ such $(L\mathbf{y})_a < 0$ and increase $\mathbf{y}_a$ by 1. Repeat until $L\mathbf{y} \geq \mathbf{0}$.*

*Remark.* In the case that the row sums of $L$ are nonnegative, the procedure halts immediately with $\mathbf{y} = \mathbf{1}$. In particular, this includes the special case $\mathcal{N} = \mathtt{Sand}(G, s)$ for an Eulerian graph $G$.

In the case $\mathcal{N}$ is not rectangular, the inclusion $K \subset D\mathbb{Z}^A$ is strict. Corollary 8.9 and Procedure 8.10 will identify the minimal burning odometer $\mathbf{k} = D\mathbf{y} \in D\mathbb{Z}^A$. Unlike the rectangular case, there may not be a unique minimal burning odometer in $K$.

If $\mathbf{k} = D\mathbf{y}$ is the minimal burning odometer in $D\mathbb{N}^A$, then the global burning test $(I - P)\mathbf{k} \triangleright\triangleright \mathbf{q}$ runs in time $\mathbf{k}$. Since $\beta = (I - P)\mathbf{k} \leq \mathbf{r}$, an upper bound for this run time is $\mathbf{k} \leq (I - P)^{-1}\mathbf{r}$.

## Appendix A. Commutative monoid actions

A finite commutative monoid $M$ contains an abelian group whose identity element is the minimal idempotent of $M$. Every monoid action of $M$ induces a corresponding group action. The main result of this section is Theorem A.5 relating these two actions.

We have not seen this theorem stated explicitly in the literature, but many of the lemmas in this section are well-known in the semigroup community. They trace their origins to the work of Green [Gre51] and Schützenberger [Sch57]; see [Gri01, Ste10] for modern treatments. We include their short proofs here in order to highlight the beauty and simplicity of the commutative case. For refinements of some of the lemmas below, and extensions to a certain class of infinite semigroups ($\pi$-regular semigroups) see [Gri07].

Let $M$ be a finite commutative monoid. An *idempotent* is an element $f \in M$ such that $ff = f$. Among the powers of any element $m \in M$ is an idempotent: indeed, since $M$ is finite $m^j = m^k$ for some $j < k$, and then $m^\ell = m^{2\ell}$ where $\ell$ is any integer multiple of $k - j$ such that $\ell \geq j$. Now consider the product of all idempotents in $M$:

$$e := \prod_{f \text{ idempotent}} f.$$

Note that $e$ is again an idempotent since $M$ is commutative. This *minimal idempotent* $e$ is accessible from all of $M$: that is, $e \in mM$ for all $m \in M$.

**Lemma A.1.** *$eM$ is an abelian group with identity element $e$.*

*Proof.* Since $e$ is an idempotent we have $e(em) = (ee)m = em$ for all $m \in M$, which verifies the identity axiom. Since $e$ is accesible from all of $M$, we have $e \in (eme)M = (em)(eM)$ which verifies existence of inverses. □

Denote by $\epsilon$ the identity element of $M$. Then $e = \epsilon$ if and only if $M$ is a group. Let

$$\mu : M \times X \to X$$
$$(m, x) \mapsto mx$$

be a monoid action of $M$ on a set $X$; that is, $\epsilon x = x$ and $m(m'x) = (mm')x$ for all $m, m' \in M$ and all $x \in X$. We say that $\mu$ is *irreducible* if there does not exist a partition of $X$ into nonempty subsets $X_1$ and $X_2$ such that $MX_1 \subset X_1$ and $MX_2 \subset X_2$.

**Lemma A.2.** *If $\mu$ is irreducible, then for any $x, x' \in X$ there exist $m, m' \in M$ such that $mx = m'x'$.*

*Proof.* Define $x \sim x'$ if there exist $m, m' \in M$ such that $mx = m'x'$. Since $M$ is commutative, $\sim$ is an equivalence relation: to check transitivity, note that $mx = m'x'$ and $m''x' = m'''x''$ imply

$$m''mx = m''m'x' = m'm''x' = m'm'''x''$$

so that $x \sim x''$.

Now fix $x \in X$ and let $X_1 = \{x' \in X \mid x' \sim x\}$ be the equivalence class of $x$. Let $X_2 = X - X_1$. For any $x' \in X$ and any $m \in M$ we have $x' \in X_1$ if and only if $mx' \in X_1$. Thus $MX_1 \subset X_1$ and $MX_2 \subset X_2$. Since $\mu$ is irreducible and $X_1$ is nonempty (it contains $x$) we conclude that $X_1 = X$. □

In general, if $\mu$ is not irreducible then $X$ can be written as a disjoint union of irreducible components $X_\alpha$, which are the equivalence classes of the relation $\sim$ defined in the proof of Lemma A.2.

**Lemma A.3.** (Recurrent Elements Of A Monoid Action) *Let $M$ be a finite commutative monoid and $\mu : M \times X \to X$ an irreducible action. The following are equivalent for $x \in X$:*

(1) $x \in My$ *for all* $y \in X$.
(2) $x \in mX$ *for all* $m \in M$.
(3) $x \in eX$.
(4) $x = ex$.

*Proof.* (1) ⇒ (2): Fix $m \in M$ and let $y = mx$. If $x \in My$, then there exists $m' \in M$ such that $x = m'y = m'mx = mm'x \in mX$.
(2) ⇒ (3): trivial.
(3) ⇒ (4): If $x = ey$, then $ex = e(ey) = (ee)y = ey = x$.
(4) ⇒ (1): By Lemma A.2, given $x, y \in X$ there exist $m, m' \in M$ such that $mx = m'y$. Let $m''$ be such that $m''m = e$. If $x = ex$, then $x = m''mx = m''m'y \in My$. □

Any commutative monoid action

$$\mu : M \times X \to X$$

induces by restriction a corresponding group action

$$eM \times eX \to eX.$$

To see this, note that for any $m \in M$ and $x \in X$ we have $m(ex) = (me)x = (em)x = e(mx) \in eX$, so the action of $M$ on $X$ restricts to a monoid action of $M$ on $eX$. Since $e(ex) = (ee)x = ex$, the element $e$ acts by identity on $eX$. Since $eM$ is a group with identity element $e$, it follows that $eM \times eX \to eX$ is a group action.

In fact slightly more is true. We say that $m \in M$ *acts invertibly* on a subset $Y \subset X$ if the map $y \mapsto my$ is a bijection $Y \to Y$.

**Lemma A.4.** *Let $M$ be a finite commutative monoid and $\mu : M \times X \to X$ a monoid action. Then every $m \in M$ acts invertibly on $eX$.*

*Proof.* For any $m \in M$ and $x \in X$ we have $(em)(ex) = (eme)x = (mee)x = (me)x = m(ex)$, so $em$ and $m$ have the same action on $eX$. Since $eM \times eX \to eX$ is a group action, $em$ and hence $m$ acts invertibly on $eX$. □

We say that a monoid action $\mu : M \times X \to X$ is *faithful* if there do not exist distinct elements $m, m' \in M$ such that $mx = m'x$ for all $x \in X$. The next lemma

shows that relatively weak properties of a monoid action (faithful and irreducible) imply stronger properties of the corresponding group action (free and transitive).

Let $G$ be a group with identity element $e$. Recall that a group action $G \times Y \to Y$ is called *transitive* if $Gy = Y$ for all $y \in Y$, and is called *free* if for all $g \neq e$ there does not exist $y \in Y$ such that $gy = y$. If the action is both transitive and free, then for any two elements $y, y' \in Y$ there is a unique $g \in G$ such that $gy = y'$; in particular, $\#G = \#Y$.

**Theorem A.5.** (Group Actions Arising From Monoid Actions) *Let $M$ be a finite commutative monoid and $\mu : M \times X \to X$ an irreducible monoid action. Then the restriction of $\mu$ to $eM \times eX$ is a transitive group action*

$$e\mu : eM \times eX \to eX.$$

*If $\mu$ is also faithful, then $e\mu$ is free.*

*Proof.* To show transitivity, let $R = \bigcap_{y \in X}(My)$. Then for any $x \in eX$ we have

$$R \subset Mx = M(ex) = (Me)x = (eM)x = e(Mx) \subset eX.$$

But $R = eX$ by the equivalence of (1) and (3) in Lemma A.3, so the above inclusions are equalities. In particular, $(eM)x = eX$ for all $x \in eX$, which shows that $eM$ acts transitively on $eX$.

To show freeness, suppose that $g(ex) = ex$ for some $g \in eM$ and some $x \in X$. Now fix $y \in X$. By transitivity, $ey = h(ex)$ for some $h \in eM$, hence

$$gy = gey = ghex = hgex = hex = ey.$$

If $\mu$ is faithful, it follows that $g = e$. Hence $eM$ acts freely on $eX$. □

## Appendix B. Dickson's Lemma

The following lemma follows from the Hilbert basis theorem applied to the monomial ideal $(\mathbf{t}^{\mathbf{x}_1}, \mathbf{t}^{\mathbf{x}_2}, \dots)$ in the polynomial ring $\mathbb{Q}[\mathbf{t}] = \mathbb{Q}[t_1, \dots, t_k]$. One can also prove it directly by induction on $k$ using the infinite pigeonhole principle.

**Lemma B.1.** (Dickson's Lemma, [Dic13]) *Let $k \geq 1$ be an integer. For any sequence $\mathbf{x}_1, \mathbf{x}_2, \dots \in \mathbb{N}^k$, there exist indices $m < n$ such that $\mathbf{x}_m \leq \mathbf{x}_n$ in the coordinatewise partial ordering.*

See [FFSS11] for an effective version, giving bounds on $n$ in terms of $k$ and the largest jump in the sequence $\mathbf{x}_i$.

## Appendix C. Toppling matrices

We will use the following form of the Perron-Frobenius theorem (see [HJ90, §8] for part (i) and [Ash87] for part (ii)).

**Lemma C.1.** (Perron-Frobenius) *Let $P$ be a square matrix with nonnegative real entries.*

(i) *$P$ has a nonnegative real eigenvector $\mathbf{x}$ with nonnegative real eigenvalue $\lambda$, such that the absolute values of all other eigenvalues of $P$ are $\leq \lambda$.*

(ii) *If $P$ has rational entries and $\lambda$ is rational, then $\mathbf{x}$ can be taken to have integer entries.*

Following [PS04], we call $L$ a *toppling matrix* if it satisfies the equivalent conditions of the following lemma. See [FK62, Theorem 4.3] for a proof of the equivalence. Such matrices are also known as "$M$-matrices."

For a vector $\mathbf{x}$, we write $\mathbf{x} > \mathbf{0}$ to mean that all coordinates of $\mathbf{x}$ are nonnegative and $\mathbf{x} \neq \mathbf{0}$.

**Lemma C.2.** (Toppling Matrices) *Let $L$ be a square matrix with real entries such that $L_{ij} \leq 0$ for all $i \neq j$. The following are equivalent.*

(1) *All principal minors of $L$ are positive.*
(2) *All eigenvalues of $L$ have positive real part.*
(3) *There exists a vector $\mathbf{x} > \mathbf{0}$ such that $L\mathbf{x} > \mathbf{0}$.*
(4) *There exists a vector $\mathbf{y} > \mathbf{0}$ such that $L^T\mathbf{y} > \mathbf{0}$.*
(5) *$L$ is invertible, and all entries of $L^{-1}$ are nonnegative.*

The next lemma encapsulates how toppling matrices arise in our setting as Laplacians of abelian networks that halt on all inputs. In particular, the damped Laplacian $\Delta^s$ of a connected graph (or more generally, of a directed graph in which every vertex has a directed path to $s$) is a toppling matrix. It corresponds to the special case of a sandpile network with a sink (§3.3). Toppling matrices are a bit more general than graph Laplacians in that they need not have all column sums nonnegative.

**Lemma C.3.** *Let $L = (I - P)D$, where $P$ is a nonnegative $n \times n$ matrix, $D$ is an $n \times n$ positive diagonal matrix, and $I$ is the $n \times n$ identity matrix. Then $L$ is a toppling matrix if and only if the Perron-Frobenius eigenvalue of $P$ is strictly less than 1.*

*Proof.* The off-diagonal entries of $L$ are nonpositive, so Lemma C.2 applies. Let $\mathbf{x}$ and $\lambda$ be a Perron-Frobenius eigenvector and eigenvalue of $P$, and let $\mathbf{y} = D^{-1}\mathbf{x}$. Then $\mathbf{y} > \mathbf{0}$ and

$$L\mathbf{y} = (I - P)\mathbf{x} = (1 - \lambda)\mathbf{x}. \tag{17}$$

If $\lambda < 1$, then $L\mathbf{y} > \mathbf{0}$, so $L$ is a toppling matrix. Conversely, if $L$ is a toppling matrix, then $L^{-1}$ has nonnegative entries. Applying $L^{-1}$ to (17) yields $\mathbf{y} = (1 - \lambda)L^{-1}\mathbf{x}$. Since $L^{-1}\mathbf{x} > \mathbf{0}$ and $\mathbf{y} > \mathbf{0}$, we conclude that $\lambda < 1$. □

## Concluding Remarks

We indicate here a few directions for further research on abelian networks.

**Halting problem for a given input.** Theorem 6.6 gives a polynomial time algorithm to check whether a finite abelian network $\mathcal{N}$ halts on all inputs. An interesting question is whether there is an efficient algorithm to check whether $\mathcal{N}$ halts on a *given* input $\mathbf{x}_0.\mathbf{q}_0$.

An inefficient algorithm runs as follows. Let $\mathbf{x}_n.\mathbf{q}_n = \mathbf{x}_{n-1} \triangleright \mathbf{q}_{n-1}$ for $n \geq 1$. By Dickson's Lemma B.1 there exist $m < n$ such that $\mathbf{q}_m = \mathbf{q}_n$ and $\mathbf{x}_m \leq \mathbf{x}_n$. Each

time we generate a new state $\mathbf{x}_n.\mathbf{q}_n$, exhaustively check for such an $m < n$. When we find one, if $\mathbf{x}_n = \mathbf{0}$ then $\mathcal{N}$ has already halted, and if $\mathbf{x}_n \neq \mathbf{0}$ then $\mathcal{N}$ will never halt.

Of course, bounds obtained from Dickson's Lemma grow very quickly [FFSS11]. Lemma 6.7 suggests a possibly more efficient approach. Given a strong amplifier $\alpha.\mathbf{q}$, what is a bound for the time it takes for $\mathcal{N}$ either to halt or be certified by Lemma 6.7 to run forever? With such a bound in hand, there remains the question of which abelian networks have small (i.e. polynomial in the size of description of $\mathcal{N}$) strong amplifiers.

**Critical networks.** Let us call an abelian network $\mathcal{N}$ *critical* if its production matrix has Perron-Frobenius eigenvalue $\lambda = 1$. By Theorem 6.6, a critical network has inputs that cause it to run forever, so it does not have a critical group in the sense of §7. However, one could try to define $\operatorname{Crit}\mathcal{N}$ by generalizing the sinkless construction of the sandpile group: $(\mathbb{Z}^V)_0/L\mathbb{Z}^V$, where $(\mathbb{Z}^V)_0$ is the kernel of the map $\mathbf{x} \mapsto \mathbf{1}^T\mathbf{x}$.

For $\alpha, \beta \in \mathbb{N}^A \times Q$, write $\alpha \to \beta$ if there exists a sequence of legal updates to $\alpha$ resulting in $\beta$. Let us call $\alpha$ *recurrent* if for any $\beta$ such that $\alpha \to \beta$ we have $\beta \to \alpha$. In a critical network $\mathcal{N}$, is there an efficient test analogous to the burning algorithm to check whether $\alpha$ is recurrent?

In the case of a simple rotor network $\texttt{Rotor}(G)$ with no sink, the recurrent amplifiers with a single letter are in bijection with *cycle-rooted spanning trees* of $G$ (spanning subgraphs with a single oriented cycle, where the letter lies on the cycle) [H+08, Theorem 3.8]. Does this result generalize to the case of abelian mobile agents (§3.8)? What if there is more than one letter?

**Characterization of recurrent states.** The recurrent states of a sandpile network $\texttt{Sand}(G, s)$ on an undirected (or Eulerian directed) graph $G$ have a characterization in terms of "forbidden subconfigurations" [Dha90] which puts them naturally in bijection with the $G$-parking functions of Postnikov and Shapiro [PS04]. The recurrent states of the rotor network $\texttt{Rotor}(G, s)$ are the progressed oriented spanning trees of $G$ rooted at $s$. It would be interesting to find combinatorial characterizations of the recurrent states of other abelian networks.

**Homotopy via embedding.** Does Lemma 5.13 have a converse? Given irreducible, strongly connected abelian networks $\mathcal{N}_1$ and $\mathcal{N}_2$ with $\mathcal{N}_1 \approx \mathcal{N}_2$ is there an irreducible $\mathcal{N}$ such that $\mathcal{N}_1$ and $\mathcal{N}_2$ are local components of a strong component of $\mathcal{N}$?

**Finer algebraic invariants.** The critical group $\operatorname{Crit}\mathcal{N}$ depends only on the homotopy type of $\mathcal{N}$ (Corollary 7.15). On the other hand, the transition monoid can detect finer information about $\mathcal{N}$. To see that the monoid *action* $M \times Q \to Q$ is not a homotopy invariant, note that the sandpile network $\texttt{Sand}(G, s)$ has a state $\mathbf{0} \in Q$ which can access all other states: $M\mathbf{0} = Q$. If $G - \{s\}$ has two directed cycles that share an edge, then $\texttt{Rotor}(G, s)$ has no such state, because a progressed cycle of rotors once broken can never be reformed.

Neither is $M$ itself a homotopy invariant: for instance, for the graph $G = \mathbb{Z}/n \times \mathbb{Z}/n$ one can show that the minimal burning element $\beta$ of $M(\texttt{Rotor}(G, s))$ has $\beta^n = e$, whereas the corresponding exponent for $\beta$ in $M(\texttt{Sand}(G, s))$ grows quadratically in $n$.

Recall that there are many distinct rotor networks $\texttt{Rotor}(G, s)$ depending on the choice of ordering of the outgoing edges of each vertex. An interesting question is whether $M$ can distinguish between these networks.

**Infinite abelian networks.** Questions about the recurrence or transience of rotor walk [LL09, AH11a, AH11b, FGLP13] and the explosiveness of sandpiles [FLP10] are cases of the halting problem for spatially infinite abelian networks. In this setting, "halting" means that *each processor* processes only finitely many letters, even though the total number of letters processed may be infinite. Generalizing the least action principle to infinite executions, along the lines of [FMR09], may be useful in approaching these and related questions.

Among the many hard questions in this area, let us single out one. Suppose $\mathcal{N}$ is a locally finite abelian network whose underlying graph is the square grid $\mathbb{Z}^2$. Let $\mathbf{q}$ be an initial state, and suppose that $\mathcal{N}$ and $\mathbf{q}$ are periodic in the sense that there is a full rank sublattice $\Lambda \subset \mathbb{Z}^2$ such that $\mathcal{P}_v$ and $\mathbf{q}_v$ depend only on $v + \Lambda$. Given the finite data of $\mathcal{P}_v$ and $\mathbf{q}_v$ on a fundamental domain, is it decidable whether there exists an input $\mathbf{x}$ with finite support such that $\mathcal{N}$ does not halt on $\mathbf{x}.\mathbf{q}$?

**Abelian networks with shared memory.** In §2.1 we have emphasized that abelian networks do not rely on shared memory. Yet there are quite a few examples of processes with a global abelian property that *do* rely on shared memory. Perhaps the simplest is *sorting by adjacent transpositions*: suppose $G$ is the path of length $n$ and each vertex $v$ has state space $Q_v = \mathbb{Z}$. The processors now live on the edges: for each edge $e = (v, v+1)$ the processor $\mathcal{P}_e$ acts by swapping the states $q(v)$ and $q(v+1)$ if $q(v) > q(v+1)$. This example does not fit our definition of abelian network because the processors of edges $(v-1, v)$ and $(v, v+1)$ share access to the state $q(v)$. Indeed, from our list of five goals in §2 this example satisfies items (a)–(c) only: The final output is always sorted, and the run time does not depend on the execution, but the local run times do depend on the execution.

What is the right definition of an abelian network with shared memory? Examples could include the numbers game of Mozes [Moz90], $k$-cores of graphs and hypergraphs, Wilson cycle popping [Wil96] and its extension by Gorodezky and Pak [GP13], source reversal [GP00] and cluster firing [H$^+$08, CPS11, Bac12].

**Abelian networks with coefficients.** We can define an abelian network purely in terms of monoids and without any reference to automata. The free commutative monoid $\mathbb{N}^A$ played an important role in our theory. In particular, we used heavily the fact that $\mathbb{N}^A$ is cancellative (for instance in the proof of Lemma 5.9). What happens if we replace $\mathbb{N}^A$ by a different monoid?

To make this question more precise, suppose $M$ and $M'$ are commutative monoids, written additively. Define an *action of $M$ on $Q$ metered by $M'$* as a

monoid action $\nu : M \times Q \to Q$ together with a map

$$\mu : M \times Q \to M'$$

satisfying $\mu(\mathbf{0}, \mathbf{q}) = \mathbf{0}'$ and

$$\mu(\mathbf{x} + \mathbf{y}, \mathbf{q}) = \mu(\mathbf{x}, \nu(\mathbf{y}, \mathbf{q})) + \mu(\mathbf{y}, \mathbf{q}) \tag{18}$$

for all $\mathbf{x}, \mathbf{y} \in M$ and all $\mathbf{q} \in Q$. The interpretation is that $\mu(\mathbf{x}, \mathbf{q})$ measures the "cost" (or "byproduct") of $\mathbf{x}$ acting on $\mathbf{q}$, and that costs are additive.

An example of a metered action is the local action $\triangleright$ of an abelian network (§5): we take $M = M' = \mathbb{N}^A$ with $\mu$ and $\nu$ defined by

$$\mathbf{x} \triangleright \mathbf{q} = \mu(\mathbf{x}, \mathbf{q}).\nu(\mathbf{x}, \mathbf{q}).$$

The byproduct of $\mathbf{x}$ acting on $\mathbf{q}$ is that some messages are passed, namely $\mu(\mathbf{x}, \mathbf{q})$.

Let us define an *abstract abelian network* on a directed graph $G = (V, E)$ as a collection of 4-tuples $(M_v, Q_v, \mu_v, \nu_v)$ indexed by $v \in V$, such that $M_v$ is a commutative monoid, $Q_v$ is a set, and $(\mu_v, \nu_v)$ is an action of $M_v$ on $Q_v$ metered by

$$\prod_{(v,u) \in E} M_u.$$

As a special case, fix a commutative monoid $C$ and an alphabet $A = \coprod_{v \in V} A_v$. An *abelian network with coefficients in* $C$ is an abstract abelian network with $M_v = C^{A_v}$ for all $v \in V$.

It would be interesting to compare the computational power of $C$-networks for different monoids $C$. For example, taking $C = \mathbb{Z}$ we obtain the class of locally recurrent abelian networks (i.e., those satisfying $Q_v = e_v Q_v$ for all $v \in V$). Taking $C = \mathbb{R}_+$ gives a class of networks with continuous input, which includes the abelian avalanche model of [Gab93] and the divisible sandpile of [LP09]. The latter computes the linear program relaxation of the integer program of Corollary 4.11. Other $\mathbb{R}_+$-networks (analogous to the oil-and-water model of §3.9) should compute the linear programs of [Tse90]. What about $C = \mathbb{Z}/p\mathbb{Z}$?

**Nonabelian networks.** The seminal work of Krohn and Rhodes [KR65, KR68] led to a detailed study of how the algebraic structure of monoids relates to the computational strength of corresponding classes of automata. It would be highly desirable to develop such a dictionary for classes of automata *networks*. Thus one would like to weaken the abelian property and study networks of solvable automata, nilpotent automata, etc. Such networks are nondeterministic — the output depends on the order of execution — so their theory promises to be rather different from that of abelian networks.

It could be fruitful to look for networks that exhibit only limited nondeterminism. A concrete example is a sandpile network with annihilating particles and antiparticles, studied by Robert Cori (unpublished) and in [CPS11] under the term "inverse toppling."

Let us point out that the definition (18) of a metered action makes sense for arbitrary monoids $M$ and $M'$, which allows us to define networks with coefficients in an arbitrary monoid $C$. Are there interesting examples with $C$ noncommutative?

## Acknowledgments

The authors thank Spencer Backman, Olivier Bernardi, Deepak Dhar, Anne Fey, Sergey Fomin, Christopher Hillar, Michael Hochman, Alexander Holroyd, Benjamin Iriarte, Mia Minnes, Ryan O'Donnell, David Perkinson, James Propp, Leonardo Rolla, Farbod Shokrieh, Allan Sly and Peter Winkler for helpful discussions.

This research was supported by an NSF postdoctoral fellowship and NSF grants DMS-1105960 and DMS-1243606, and by the UROP and SPUR programs at MIT.

## References

[AR08] F. C. Alcaraz and V. Rittenberg, Directed abelian algebras and their applications to stochastic models *Phys. Rev. E* **78**:041126, 2008. arXiv:0806.1303

[APR09] F. C. Alcaraz, P. Pyatov and V. Rittenberg, Two-component abelian sandpile models *Phys. Rev. E* **79**:042102, 2009. arXiv:0810.4053

[AH11a] Omer Angel and Alexander E. Holroyd, Rotor walks on general trees. *SIAM J. Discrete Math.* **25**(1):423–446, 2011. arXiv:1009.4802

[AH11b] Omer Angel and Alexander E. Holroyd, Recurrent rotor-router configurations. 2011. arXiv:1101.2484

[Ash87] Jonathan Ashley, On the Perron-Frobenius eigenvector for nonnegative integral matrices whose largest eigenvalue is integral, *Lin. Alg. Appl.* **94**:103–108, 1987.

[BT10] László Babai and Evelin Toumpakari, A structure theory of the sandpile monoid for directed graphs, 2010. http://people.cs.uchicago.edu/~laci/REU10/evelin.pdf

[Bac12] Spencer Backman, A bijection between the recurrent configurations of a hereditary chip-firing model and spanning trees. arXiv:1207.6175

[BTW87] Per Bak, Chao Tang and Kurt Wiesenfeld, Self-organized criticality: an explanation of the $1/f$ noise, *Phys. Rev. Lett.* **59**(4):381–384, 1987.

[BW03] Itai Benjamini and David B. Wilson, Excited random walk, *Elect. Comm. Probab.* **8**:86–92, 2003.

[Big99] Norman L. Biggs, Chip-firing and the critical group of a graph, *J. Algebraic Combin.* **9**(1):25–45, 1999.

[BW97] Norman L. Biggs and Peter Winkler, Chip-firing and the chromatic polynomial. Technical Report LSE-CDAM-97-03, London School of Economics, Center for Discrete and Applicable Mathematics, 1997.

[BLS91] Anders Björner, László Lovász and Peter Shor, Chip-firing games on graphs, *European J. Combin.* **12**(4):283–291, 1991.

[CPS11] Sergio Caracciolo, Guglielmo Paoletti and Andrea Sportiello, Multiple and inverse toppplings in the abelian sandpile model. arXiv:1112.3491

[CCG13] Melody Chan, Thomas Church and Joshua A. Grochow, Rotor-routing and spanning trees on planar graphs. arXiv:1308.2677

[C$^+$13] Scott Chapman, Rebecca Garcia, Luis David Garca-Puente, Martin E. Malandro and Ken W. Smith, Algebraic and combinatorial aspects of sandpile monoids on directed graphs, *J. Comb. Theory A* **120**(1):245–265, 2013. arXiv:1105.2357

[CP05] Denis Chebikin and Pavlo Pylyavskyy, A family of bijections between $G$-parking functions and spanning trees, *J. Combin. Theory A* **110**(1):31–41, 2005.

[CS06] Joshua Cooper and Joel Spencer, Simulating a random walk with constant error, *Combin. Probab. Comput.* **15**:815–822, 2006.

[CL03] Robert Cori and Yvan Le Borgne. The sand-pile model and Tutte polynomials. *Adv. in Appl. Math.* **30**(1-2):44–52, 2003.

[DR04] Arnoud Dartois and Dominique Rossin, Height-arrow model, *Formal Power Series and Algebraic Combinatorics*, 2004.

[Dha90] Deepak Dhar, Self-organized critical state of sandpile automaton models, *Phys. Rev. Lett.* **64**:1613–1616, 1990.

[Dha99a] Deepak Dhar, The abelian sandpile and related models, *Physica A* **263**:4–25, 1999. arXiv:cond-mat/9808047

[Dha99b] Deepak Dhar, Studying self-organized criticality with exactly solved models, 1999. arXiv:cond-mat/9909009

[Dha99c] Deepak Dhar, Some results and a conjecture for Manna's stochastic sandpile model, *Physica A* **270**:69–81, 1999. arXiv:cond-mat/9902137

[Dha06] Deepak Dhar, Theoretical studies of self-organized criticality, *Physica A* **369**:29–70, 2006.

[DSC09] Deepak Dhar, Tridib Sadhu and Samarth Chandra, Pattern formation in growing sandpiles, *Europhysics Lett.* **85**:48002, 2009. arXiv:0808.1732

[DF91] Persi Diaconis and William Fulton, A growth model, a game, an algebra, Lagrange inversion, and characteristic classes, *Rend. Sem. Mat. Univ. Pol. Torino* **49**(1):95–119, 1991.

[DFR05] Emeric Deutsch, Luca Ferrari and Simone Rinaldi, Production matrices, *Adv. Appl. Math.* **34**(1):101–122, 2005.

[DRS10] Ronald Dickman, Leonardo T. Rolla and Vladas Sidoravicius, Activated random walkers: facts, conjectures and challenges, *J. Stat. Phys.* **138**(1-3):126–142, 2010. arXiv:0910.2725

[Dic13] Leonard E. Dickson, Finiteness of the odd perfect and primitive abundant numbers with distinct prime factors, *Amer. J. Math.* **35**(4):413–422, 1913.

[DFS08] Benjamin Doerr, Tobias Friedrich and Thomas Sauerwald, Quasirandom rumor spreading, *Proceedings of the nineteenth annual ACM-SIAM symposium on discrete algorithms (SODA '08)*, pages 773–781, 2008. arXiv:1012.5351

[Ent87] Aernout C. D. van Enter, Proof of Straley's argument for bootstrap percolation. *J. Stat. Phys.* **48**(3-4):943–945, 1987.

[Eri96] Kimmo Eriksson, Chip-firing games on mutating graphs, *SIAM J. Discrete Math.* **9**(1):118–128, 1996.

[FLP10] Anne Fey, Lionel Levine and Yuval Peres, Growth rates and explosions in sandpiles, *J. Stat. Phys.* 138:143–159, 2010. arXiv:0901.3805

[FMR09] Anne Fey, Ronald Meester, and Frank Redig, Stabilizability and percolation in the infinite volume sandpile model, *Ann. Probab.* **37**(2):654-675, 2009. arXiv:0710.0939

[FFSS11] Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen, Ackermannian and primitive-recursive bounds with Dickson's lemma, *26th Annual IEEE Symposium on Logic in Computer Science*, 2011. arXiv:1007.2989

[FGLP13] Laura Florescu, Shirshendu Ganguly, Lionel Levine and Yuval Peres, Escape rates for rotor walks in $\mathbb{Z}^d$, arXiv:1301.3521.

[Fre93] Vidar Frette, Sandpile models with dynamically varying critical slopes, *Phys. Rev. Lett.* **70**:2762–2765, 1993.

[FL12] Tobias Friedrich and Lionel Levine, Fast simulation of large-scale growth models, *Random Struct. Alg.*, to appear, 2012. arXiv:1006.1003.

[FK62] M. Fiedler AND V. Ptak, On matrices with non-positive off-diagonal elements and positive principal minors, *Czechoslovak Math. J.* **12**:382–400, 1962.

[Gab93] Andrei Gabrielov, Abelian avalanches and Tutte polynomials, *Physica A* **195**:253–274, 1993.

[Gab94] Andrei Gabrielov, Asymmetric abelian avalanches and sandpiles. Preprint, 1994. http://www.math.purdue.edu/~agabriel/asym.pdf

[GP00] Eric Goles and Erich Prisner, Source reversal and chip firing on graphs, *Theoret. Comp. Sci.* **233**:287–295, 2000.

[GP13] Igor Gorodezky and Igor Pak, Generalized loop-erased random walks and approximate reachability, *Random Struct. Alg.*, to appear.

[GLPZ12] Giuliano Pezzolo Giacaglia, Lionel Levine, James Propp and Linda Zayas-Palmer, Local-to-global principles for the hitting sequence of a rotor walk, *Electr. J. Combin.* **19**:P5, 2012. arXiv:1107.4442

[Gre51] J. A. Green, On the structure of semigroups, *Ann. Math.* 54:163–172, 1951.

[Gri01] Pierre A. Grillet, *Commutative semigroups*, Klower Academic Publishers, 2001.

[Gri07] Pierre A. Grillet, Commutative actions, *Acta Sci. Math. (Szeged)* **73**:91–112, 2007.

[Hol03] Alexander E. Holroyd, Sharp metastability threshold for two-dimensional bootstrap percolation, *Probab. Theory Related Fields*, **125**(2):195–224, 2003. arXiv:math/0206132

[H$^+$08] Alexander E. Holroyd, Lionel Levine, Karola Mészáros, Yuval Peres, James Propp and David B. Wilson, Chip-firing and rotor-routing on directed graphs, *In and out of equilibrium 2*, 331–364, Progr. Probab. **60**, Birkhäuser, 2008. arXiv:0801.3306

[HP10] Alexander E. Holroyd and James G. Propp, Rotor walks and Markov chains, in *Algorithmic Probability and Combinatorics*, American Mathematical Society, 2010. arXiv:0904.4507

[HJ90] Roger A. Horn and Charles R. Johnson, *Matrix Analysis*, Cambridge Univ. Press, 1990.

[HS11] Wilfried Huss and Ecaterina Sava, Rotor-router aggregation on the comb, *Electr. J. Combin.* **18**(1):P224, 2011. arXiv:1103.4797

[KL10] Wouter Kager and Lionel Levine, Rotor-router aggregation on the layered square lattice, *Electr. J. Combin.* **17**:R152, 2010. arXiv:1003.4017

[KS05] Harry Kesten and Vladas Sidoravicius, The spread of a rumor or infection in a moving population, *Ann. Probab.* **33**:2402–2462, 2005. arXiv:math/0312496

[KS08] Harry Kesten and Vladas Sidoravicius, A shape theorem for the spread of an infection, *Ann. Math.* **167**:701–766, 2008. arXiv:math/0312511

[KR65] Kenneth Krohn and John Rhodes, Algebraic theory of machines. I. Prime decomposition theorem for finite semigroups and machines, *Trans. Amer. Math. Soc.* **116**:450–464, 1965.

[KR68] Kenneth Krohn and John Rhodes, Complexity of finite semigroups, *Ann. Math.* **88**:128–160, 1968.

[LL09] Itamar Landau and Lionel Levine, The rotor-router model on regular trees, *J. Combin. Theory A* **116**: 421–433, 2009. arXiv:0705.1562

[LP09] Lionel Levine and Yuval Peres, Strong spherical asymptotics for rotor-router aggregation and the divisible sandpile, *Potential Anal.* **30**:1–27, 2009. arXiv:0704.0688

[Lor89] Dino J. Lorenzini, Arithmetical graphs, *Math. Ann.* **285**(3):481–501, 1989.

[Lor91] Dino J. Lorenzini, A finite group attached to the Laplacian of a graph, *Discrete Math.* **91**(3):277–282, 1991.

[Man91] S. S. Manna, Two-state model of self-organized criticality, *J. Phys. A: Math. Gen.* **24**:L363, 1991.

[MM11] Juan Andres Montoya and Carolina Mejia, The computational complexity of the abelian sandpile model, 2011. http://matematicas.uis.edu.co/jmontoya/sites/default/files/notas-ASM.pdf

[MN99] Cristopher Moore and Martin Nilsson. The computational complexity of sandpiles. *J. Stat. Phys.* **96**:205–224, 1999.

[Moz90] Shahar Mozes, Reflection processes on graphs and Weyl groups, *J. Comb. Theory A* **53**(1):128–142, 1990.

[Ost03] Srdjan Ostojic, Patterns formed by addition of grains to only one site of an abelian sandpile, *Physica A* **318**:187–199, 2003.

[PDDK96] V. B. Priezzhev, Deepak Dhar, Abhishek Dhar and Supriya Krishnamurthy, Eulerian walkers as a model of self-organised criticality, *Phys. Rev. Lett.* **77**:5079–5082, 1996. arXiv:cond-mat/9611019

[PPW11] David Perkinson, Jacob Perlman and John Wilmes, Primer for the algebraic geometry of sandpiles. arXiv:1112.6163

[PS04] Alexander Postnikov and Boris Shapiro, Trees, parking functions, syzygies, and deformations of monomial ideals. *Trans. Amer. Math. Soc.* **356**(8):3109–3142, 2004. arXiv:math.CO/0301110

[Pro03] James Propp, Random walk and random aggregation, derandomized, 2003. http://research.microsoft.com/apps/video/default.aspx?id=104906

[Pro10] James Propp, Discrete analog computing with rotor-routers. *Chaos* **20**:037110, 2010. arXiv:1007.2389

[RS11] Leonardo T. Rolla and Vladas Sidoravicius, Absorbing-state phase transition for driven-dissipative stochastic dynamics on $\mathbb{Z}$, *Inventiones Math.*, to appear. arXiv:0908.1152

[Sch57] Marcel-Paul Schützenberger, $\overline{\mathcal{D}}$ représentation des demi-groupes, *C. R. Acad. Sci. Paris* 244:1994–96, 1957.

[Spe93] Eugene R. Speer, Asymmetric abelian sandpile models. *J. Stat. Phys.* **71**:61–74, 1993.

[Ste10] Benjamin Steinberg, A theory of transformation monoids: combinatorics and representation theory, *Electr. J. Combin.* **17**:R164, 2010. arXiv:1004.2982

[Tar88] Gábor Tardos, Polynomial bound for a chip firing game on graphs, *SIAM J. Disc. Math.* **1**(3):1988.

[Tse90] Paul Tseng, Distributed computation for linear programming problems satisfying a certain diagonal dominance condition, *Mathematics of Operations Research* **15**(1):33–48, 1990.

[WLB96] Israel A. Wagner, Michael Lindenbaum and Alfred M. Bruckstein, Smell as a computational resource — a lesson we can learn from the ant, *4th Israeli Symposium on Theory of Computing and Systems*, pages 219–230, 1996.

[Wil96] David B. Wilson, Generating random spanning trees more quickly than the cover time, *28th Annual ACM Symposium on the Theory of Computing (STOC '96)*, pages 296–303, 1996.

Benjamin Bond, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139.

Lionel Levine, Department of Mathematics, Cornell University, Ithaca, NY 14853.
http://www.math.cornell.edu/~levine