Computational Complexity Studies of Synchronous Boolean Finite Dynamical Systems

Mitsunori Ogihara[1,a] Kei Uchizawa[2,b]

Abstract: This paper studies the subclass of finite dynamical systems the *synchronous boolean finite dynamical system* (*synchronous BFDS*, for short), where the states are boolean and the state update takes place in discrete time and at the same on all objects. The present paper is concerned with some problems regarding the behavior of synchronous BFDS in which the state update functions (or the local state transition functions) are chosen from a predetermined finite basis of boolean functions $\mathcal{B}$. Specifically the following three behaviors are studied: (1) Does a system at hand converge on a given initial state configuration? (2) Will a system starting in given two state configurations produce a common configuration? (3) Since the state space is finite, every BFDS on a given initial state configuration either converges or enters a cycle having length greater than 1; if the latter is the case, what is the length of the loop? We prove that the three problems are each PSPACE-complete if the boolean function basis contains NAND, NOR or both AND and OR, while the problem (1) is in P, the problem (2) is in UP, and the problem (3) is in UP ∩ coUP if the set B is one of {AND}, {OR} and {XOR, NXOR}.

1. Introduction

The finite dynamical system is a system consisting of some finite number of objects that each take upon a value from some domain D. After receiving an initial state assignment the system evolves over time by means of state updates, where the updates occur in discrete time and are governed by a global state-update schedule and a local (meaning assigned to each node individually) state-update functions (or local state-transition functions) that take as input the states of the objects in the system.

Because of its flexibility the finite dynamical system has been used as a mathematical model for time-dependent systems and can contain in itself other multi-object computational models, such as cellular and graph automata and Hopfield networks.

Classes of finite dynamical systems can be defined by giving certain requirements to their operation. First, classes can be defined by specifying the domain, that is, the set of permissible states: *infinite*, *finite*, and *boolean*. Next, classes can be defined based upon the types of the state update functions.

It is usually assumed that at each time step, all the objects conduct their state updates exactly once, and so, classes can be defined depending on the order in which the state updates occur in the objects . Specifically we have the *asynchronous* (any update order), the *sequential* (a fixed predetermined order), and the *synchronous* (all at the same time) finite dynamical systems.

For each $n \geq 1$, the underlying structure of an n-object dynamical system over domain D can be represented as a node- and edge-labeled directed graph G of n nodes. The nodes of G represent the objects, the edges of G represent the direct dependencies among the objects in updating their states in a natural way: an edge from a node u to a node v indicates that the state updating function of v takes the state of u as input. Also, for each node v, v is labeled by the state update function of v and the incoming edges of v are labeled by the input positions of the source node in the state update function. Because of this representation, classes of the finite dynamical systems can be defined in terms of the properties of the underlying graph, e.g., whether the graph is planar, whether the graph is regular, and whether the edges are undirected in the sense, that if there is an edge from node u to node v, there is an edge from v to u.

The subject of this paper is the *synchronous boolean finite dynamical systems* (*synchronous BFDS*, for short). A synchronous BFDS is the subclass of BFDS in which the domain is boolean and the update is synchronous.

Given a finite dynamical system we are naturally interested in its behavior. For example, we may ask questions about fixed points, such as whether the system has a fixed point (that is, whether there is a state configuration in which the state update of the system produces no change). In the case where the state domain is finite, there are a finite number of state configurations, and so we can ask such questions how many fixed points the system has and how many initial state configurations lead to fixed points. Furthermore, we can ask about the behavior of the system on a particular initial state configuration, such as, whether a given initial state configuration leads the system to a fixed point, and if not, since the system eventually enters a cycle of state configurations, how many steps it will take for the system to enter a cycle

[1] Department of Computer Science, University of Miami 1365 Memorial Drive Coral Gables, FL 33146, USA
[2] Faculty of Engineering, Yamagata University, Jonan 4-3-16, Yonezawa, Yamagata, 992-8510 Japan
[a] ogihara@cs.miami.edu
[b] uchizawa@yz.yamagata-u.ac.jp

and how long the cycle is.

That the underlying structure of finite dynamical systems can be represented as a graph suggests that the classes of finite dynamical systems can be studied using the number of objects as the size parameter and so the behavioral properties of a class of finite dynamical systems can be studied in terms of its computational complexity. In other words, for a class of finite dynamical systems C and for a question Q, we ask how computationally hard it is to answer Q for class C: *Is it polynomial time solvable? If not, is the problem hard for a known complexity class, such as* NP *and* PSPACE*?*

Much work has been done to explore the computational complexity of behavioral properties of finite dynamical systems. Barrett *et al.* [3] study the computational complexity of the sequential finite dynamical systems, the model first introduced by Barrett, Mortveit, and Reidys in [1]. Barrett *et al.* [3] study particularly the sequential boolean finite dynamical systems regarding the existence of fixed points. For a variety of permissible state update functions, they ask which combinations of the functions make the problem easy or difficult. They show that the problem is NP-complete if the set of permissible local transition functions is either $\{\text{NAND}, \text{XNOR}\}, \{\text{NAND}, \text{XOR}\}, \{\text{NOR}, \text{XNOR}\}$ or $\{\text{NOR}, \text{XOR}\}$. They also show that the problem is solvable in polynomial time if the functions are chosen from $\{\text{AND}, \text{OR}, \text{NAND}, \text{NOR}\}$.

The above results have been strengthened by Kosub [7], who shows a dichotomy result in the sense of Schaefer [12]; i.e., the problem in question is either NP-hard or polynomial time solvable. Kosub obtains a complete complexity-theoretic characterization of the fixed-point problem about boolean finite dynamical systems with respect to the state update function classes, which Kosub calls *Post Classes*, as well as with respect to the structure of the underlying graph. He shows exactly in which case the problem is NP-hard and for all the remaining cases the problem is polynomial-time solvable. Kosub and Homan [10] prove a dichotomy result on the counting version of the fixed point problem, in the sense that the problem is either #P-complete or polynomial-time solvable.

Another set of natural problems that arise in finite dynamical systems is the reachability; that is, given a system and two state configurations $\boldsymbol{a}$ and $\boldsymbol{b}$, can $\boldsymbol{b}$ be reached from $\boldsymbol{a}$? A variant of this problem is whether any fixed point can be reached from a given configuration $\boldsymbol{a}$. Barrett *et al.* [2] study these problems for the sequential and synchronous dynamical systems in which the underlying graph is an undirected graph. They show that the problems are PSPACE-complete in general but polynomial time solvable if the state update functions are symmetric and monotone boolean functions.

In this paper, as a follow-up of the aforementioned prior work [2], [3], [7], [10]), we study the computational complexity of the synchronous boolean finite dynamical systems in which the basis $\mathcal{B}$ of the state update functions is finite. We are particularly interested in three questions:

(1) Convergence($\mathcal{B}$): Given a system $\mathcal{F}$ and an initial state configuration $\boldsymbol{a}$, decide whether the system converges to any fixed point.
(2) PathIntersection($\mathcal{B}$): Given an n-object system $\mathcal{F}$ and two state configurations $\boldsymbol{a}$ and $\boldsymbol{b}$, do there exist time steps s and t, such that the state configuration of $\mathcal{F}$ on $\boldsymbol{a}$ at step s is equal to the state configuration of $\mathcal{F}$ on $\boldsymbol{b}$ at step t?
(3) CycleLength($\mathcal{B}$): Given a system $\mathcal{F}$, an initial state configuration $\boldsymbol{a}$, and an integer t, decide whether the state configuration sequence generated by the system starting from $\boldsymbol{a}$ contains a cycle having length greater than or equal to t. Note that the complement of this problem with $t = 2$ is Convergence($\mathcal{B}$).

Although our work may seem reminiscent of the previous work, our focus is on the dynamical systems whose underlying graph is directed, not undirected. It is known that the dynamical behavior of the Hopfield networks is different depending on whether they are symmetric or not [11]. There is thus no *a priori* reason to believe that the results regarding directed graph structures are derived from the results regarding undirected graph structures.

We first show that the above three problems are all PSPACE-complete if $\mathcal{B}$ contains NAND or NOR. While we provide a proof for the result, these follow from an earlier paper by Floréen and Orponen [5]. We note that Barrett *et al.* [2] study this problem too, but in their setting the underlying graph is undirected.

We then prove that if $\mathcal{B}$ is one of $\{\text{AND}\}$, $\{\text{OR}\}$, and $\{\text{XOR}, \text{NXOR}\}$, Convergence is solvable in polynomial time and that the same assumption implies that PathIntersection belongs to UP and CycleLength belongs to UP $\cap$ coUP. We suspect that the latter two problems are polynomial time solvable, but we do not have at hand yet proofs that the problems are in P.

The rest of the paper is organized as follows. In Section 2, we formally define the dynamical systems and the problems we will study. In Section 3, we give algorithms for the convergence problem and the path intersection problem. In Section 4, we show that the cycle length problem is in UP $\cap$ coUP.

2. Preliminaries

2.1 Definitions

Below, following the definition of the sequential dynamical systems by Laubenbacher and Pareigis [8] we define synchronous boolean finite dynamical systems.

Let $n \geq 1$ be an integer. A *synchronous boolean finite dynamical system* (synchronous BFDS, for short) of n variables is an n-tuple $\mathcal{F} = (f_1, f_2, \ldots, f_n)$ such that $f_1, \ldots, f_n$ are boolean functions of n variables.

Let $\mathcal{F} = (f_1, f_2, \ldots, f_n)$ be an n-variable synchronous BFDS. A *state configuration* (or simply a *configuration*) of $\mathcal{F}$ is an n-dimensional boolean vector. We use the vector notation $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ to denote a state configuration, where $x_1, \ldots, x_n$ are boolean variables.

The action of $\mathcal{F}$ on an state configuration $\boldsymbol{x}$ is defined as:

$$\mathcal{F}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$$

In other words, the elements of $\mathcal{F}(\boldsymbol{x})$ are obtained by applying the n boolean functions $f_1, \ldots, f_n$ concurrently on the variables $x_1, \ldots, x_n$. Given an initial state configuration $\boldsymbol{x}^0 = (x_1^0, x_2^0, \ldots, x_n^0)$, the synchronous BFDS defines n sequences of boolean values $\{x_i^t\}$, $1 \leq i \leq n$ and $t \geq 0$ by iterative applications of $\mathcal{F}$ on the initial state configuration vector:

$$\text{for all } t \geq 0, \boldsymbol{x}^{t+1} = \mathcal{F}(\boldsymbol{x}^t),$$

where for all $t \geq 0$, $\boldsymbol{x}^t = (x_1^t, x_2^t, \ldots, x_n^t)$. In other words, for all $t \geq 0$,

$$\boldsymbol{x}^t = \mathcal{F}^t(\boldsymbol{x}^0).$$

For an n-state boolean finite dynamical system, there are exactly 2^n possible state configurations. This implies that in an n-state synchronous BFDS, regardless of which initial state configuration $\boldsymbol{x}^0$ it starts, the state configuration sequence generated from $\boldsymbol{x}^0$ *enters a cycle*; that is, in the sequence there exist indices s and t, $0 \leq s < t$, such that $\boldsymbol{x}^s = \boldsymbol{x}^t$. Clearly, for all such pairs (s, t), it holds:

$$\text{for all } i \geq 0, \boldsymbol{x}^{s+i} = \boldsymbol{x}^{t+i}.$$

This implies that there is the smallest value of s for which there exists some $t > s$ such that $\boldsymbol{x}^s = \boldsymbol{x}^t$ and that, for that smallest value of s, there exists the smallest value of $t > s$ such that $\boldsymbol{x}^s = \boldsymbol{x}^t$. Let s_0 and t_0 respectively be the values of s and t thus defined. Then we have:

- $t_0 \leq 2^n$ and
- for all i and j, $0 \leq i < j \leq t_0 - 1$, $\boldsymbol{x}^i \neq \boldsymbol{x}^j$.

We say that $\mathcal{F}$ on $\boldsymbol{x}$ *enters a cycle* (or *enters a loop*) at step s_0 and its cycle has *length* $t_0 - s_0$. We call s_0 the *tail length of* $\mathcal{F}$ *on* $\boldsymbol{x}$. We define $L_{\mathcal{F}}(\boldsymbol{x}^0)$ to be the length of the cycle $t_0 - s_0$.

In the case where $t_0 = s_0 + 1$, the cycle length is 1, and so, for all $s \geq s_0$ it holds that $\boldsymbol{x}^{s_0} = \boldsymbol{x}^s$. In such a case we say that the vector $\boldsymbol{x}^{s_0}$ is a *fixed point* of $\mathcal{F}$; we also say that $\mathcal{F}$ *converges on the initial state configuration* $\boldsymbol{x}^0$.

A *function family* is a collection of boolean functions $\mathcal{H} = \{h_i\}_{i\geq 1}$ such that for each $i \geq 1$, h_i takes i inputs. For example, the disjunction of any input size, which can be described as

$$\{h_i\}_{i\geq 1}, h_i(x_1, \ldots, x_i) = x_1 \vee \cdots \vee x_i,$$

is a function family. For a function family $\mathcal{H}$, we write $\mathcal{H}_k$ to mean the element of $\mathcal{H}$ for input size k. For example, OR is the family of the disjunction functions while OR_2 is the binary disjunction function.

A *basis boolean function* is either a single boolean function or a function family. Let f be a boolean function of n variables and let g be a boolean function of m variables for some $m < n$. We say that g is *equivalent* to f if there exist indices $x_{i_1}, \ldots, x_{i_m}$ such that for all $x_1, \cdots, x_n \in \{0, 1\}$, it holds that

$$f(x_1, \ldots, x_n) = g(x_{i_1}, \ldots, x_{i_m}).$$

In other words, f is a function that depends only on the variables $x_{i_1}, \ldots, x_{i_m}$ and g characterizes the behavior of f on those m inputs.

Let $\mathcal{B}$ be a finite set of basis functions. We say that a synchronous BFDS $\mathcal{F} = (f_1, \ldots, f_n)$ *has basis* $\mathcal{B}$ if each function of $\mathcal{F}$ is either a function family in $\mathcal{B}$ or equivalent to a boolean function in $\mathcal{B}$. In this paper, we consider specifically the bases that are chosen from function families AND, NAND, OR, NOR, XOR, and NXOR.

We say that a function family $H = \{h_i\}_{i\geq 1}$ is *polynomial-time (respectively, polynomial-space) computable* if there exists an algorithm for computing, given an integer $i \geq 1$ and $a_1, \ldots, a_i \in \{0, 1\}$, the value of $h_i(a_1, \ldots, a_i)$ in time (respectively, space) polynomial in i. We say that a function base $\mathcal{B}$ is *polynomial-time (respectively, polynomial-space) computable* if each function family in $\mathcal{B}$ is polynomial-time (respectively, polynomial-space) computable.

Given the above formulation it is now possible to discuss how to encode the synchronous BFDS $\mathcal{F}$ over a basis $\mathcal{B}$. An n-object synchronous BFDS f over a basis $\mathcal{B}$ is encoded as a labeled directed graph $G = (V, E)$ in which V is the object set and E represents the dependency of the objects in terms of their state update. The nodes are labeled with their basis function. The number of incoming edges to each node is no more than the number of inputs to the basis function it is associated with, and those edges are labeled to indicate the positions of the variables in the input of the basis functions. Thus, any basis $\mathcal{B}$, the synchronous BFDS $\mathcal{F}$ over $\mathcal{B}$ has an encoding whose length is bounded by a fixed polynomial in the number of objects. Note that such an encoding may not exist if $\mathcal{B}$ contains a function family that does not have a polynomial-size encoding.

We now formally define the three decision problems we consider in the paper. Let $\mathcal{B}$ be a boolean function basis.

(1) Convergence($\mathcal{B}$) is the problem of deciding, given a synchronous BFDS $\mathcal{F}$ having basis $\mathcal{B}$ and an initial state configuration $\boldsymbol{a}$ of $\mathcal{F}$, whether $\mathcal{F}$ converges on $\boldsymbol{a}$.

(2) PathIntersection($\mathcal{B}$) is the problem of deciding, given a synchronous BFDS $\mathcal{F}$ having basis $\mathcal{B}$ and two initial state configurations $\boldsymbol{a}$ and $\boldsymbol{b}$ of $\mathcal{F}$, whether there exist some s and t, $0 \leq s, t \leq 2^n - 1$, such that $\mathcal{F}^s(\boldsymbol{a}) = \mathcal{F}^t(\boldsymbol{b})$.

(3) CycleLength($\mathcal{B}$) is the problem of deciding, given a synchronous BFDS $\mathcal{F}$ having basis $\mathcal{B}$, an initial state configuration $\boldsymbol{a}$ of $\mathcal{F}$, and an integer t, whether the cycle length of $\mathcal{F}$ on $\boldsymbol{a}$, i.e., $L_{\mathcal{F}}(\boldsymbol{a})$, is greater than t.

We assume that the reader is familiar with introductory-level complexity classes (see, e.g., Hemaspaandra and Ogihara [6], for reference). The class PSPACE consists of all decision problems that can be decided by a polynomial space-bounded Turing machines. The class UP consists of all decision problems that can be decided by a polynomial time-bounded nondeterministic Turing machines with a special property that given as input each positive (respectively, negative) instance, the number of accepting computation paths of the machine is 1 (respectively, 0). The class coUP is the class of all decision problems that are the complement of some decision problem in UP.

2.2 PSPACE-completeness

Here we prove that the aforementioned three problems are PSPACE-complete if the basis contains NAND, NOR or both AND and OR.

Proposition 1. *For all polynomial-space computable bases* $\mathcal{B}$, Convergence($\mathcal{B}$), PathIntersection($\mathcal{B}$) *and* CycleLength($\mathcal{B}$) *are in* PSPACE.

Proof. Let M be a Turing machine that, given as input a synchronous BFDS S of some n objects, an initial state configuration $\boldsymbol{a}$, and an integer $t, 0 \leq t \leq 2^n$, outputs $S^t(\boldsymbol{a})$. Since the basis $\mathcal{B}$ is polynomial-space computable, M can be made to run in polynomial space. Using this machine M as a subroutine, the three problems can be solved as follows:

- Convergence($\mathcal{B}$): Test whether there exists a $t, 0 \leq t \leq 2^n$, such that $M(S, \boldsymbol{a}, t) = M(S, \boldsymbol{a}, t+1)$.
- PathIntersection($\mathcal{B}$): Test whether there exist s and t, $0 \leq s, t \leq 2^n - 1$, such that $M(S, \boldsymbol{a}, s) = M(S, \boldsymbol{b}, t)$.
- CycleLength($\mathcal{B}$): Test whether there are no k and l, $0 \leq k, l \leq 2^n$ and $l - k \leq t$, such that $M(S, \boldsymbol{a}, k) = M(S, \boldsymbol{a}, l)$.

Clearly, each of the above search can be run using $O(n)$ space. Thus, all three problems are in PSPACE.

□

The following theorem follows from [5], Corollary 3.2; we omit the proof due to the page limitation.

Theorem 1. *If the basis $\mathcal{B}$ contains either* NAND, NOR *or* {AND, OR}*, the problems* Convergence($\mathcal{B}$)*,* PathIntersection($\mathcal{B}$) *and* CycleLength($\mathcal{B}$) *are* PSPACE*-hard.*

The theorem immediately implies the following corollaries.

Corollary 1. *If the basis $\mathcal{B}$ contains either* NAND *or* NOR*, the problems* Convergence($\mathcal{B}$)*,* PathIntersection($\mathcal{B}$) *and* CycleLength($\mathcal{B}$) *are* PSPACE*-complete.*

Corollary 2. *If the basis $\mathcal{B}$ contains both* AND *and* OR*, the problems* Convergence($\mathcal{B}$)*,* PathIntersection($\mathcal{B}$) *and* CycleLength($\mathcal{B}$) *are* PSPACE*-complete.*

3. Algorithms for Convergence and PathIntersection

In this section, we prove the following theorem.

Theorem 2. *If $\mathcal{B}$ is one of* {AND}, {OR}*, and* {XOR, NXOR}*,* Convergence($\mathcal{B}$) *is polynomial-time computable and* PathIntersection($\mathcal{B}$) *belongs to* UP.

The theorem is built upon the following lemma, which states that the state configuration at any time step $t, 0 \leq t \leq 2^n$, of a synchronous BFDS of n objects can be computed in time polynomial in n for a basis chosen from {AND}, {OR}, or {XOR, NXOR}.

Lemma 1. *Let $\mathcal{B}$ be one of* {AND}, {OR}*, and* {XOR, NXOR}*. Given an n-object synchronous BFDS $\mathcal{F}$ over basis $\mathcal{B}$, a state configuration $\boldsymbol{a} \in \{0,1\}^n$, and an integer $k \geq 0$, we can compute $\mathcal{F}^k(\boldsymbol{a})$ in time polynomial in $n + \log k$.*

Proof. In this proof we will think of the state configurations to be column vectors. We first consider the case where $\mathcal{B} = \{OR\}$. Let $\mathcal{F} = (f_1, f_2, \ldots, f_n)$ and $\boldsymbol{a} \in \{0,1\}^n$ be respectively an n-object BFDS over $\mathcal{B}$ and its state configuration. Let A be the adjacency matrix of the system $\mathcal{F}$ in terms of its graph-based encoding; that is, for all i and j, $1 \leq i, j \leq n$, the entry (i, j) of A is 1 if there is an edge from node j to node i and 0 otherwise. We then have

$$\mathcal{F}(\boldsymbol{a}) = A\boldsymbol{a},$$

where the multiplication is interpreted as AND and the addition as OR. It follows from this that for all $k \geq 0$

$$\mathcal{F}^k(\boldsymbol{a}) = A^k\boldsymbol{a}$$

and that $\mathcal{F}^k$ therefore can be computed by way of the standard iterated multiplication. Thus, for all k, $\mathcal{F}^k(\boldsymbol{a})$ can be computed in time polynomial in $n + \log k$.

Next we consider the case where $\mathcal{B} = \{\text{AND}\}$. For each vector $\boldsymbol{a}$, $\boldsymbol{a}^c$ be the component-wise complement of $\boldsymbol{a}$, that is, the vector constructed from $\boldsymbol{a}$ by flipping each element. Then we have

$$\mathcal{F}(\boldsymbol{a})^c = A\boldsymbol{a}^c.$$

This implies that for all $k \geq 0$,

$$(\mathcal{F}(\boldsymbol{a})^k)^c = A^k\boldsymbol{a}^c$$

and so

$$\mathcal{F}(\boldsymbol{a})^k = (A^k\boldsymbol{a}^c)^c.$$

Thus, from the previous discussion, the lemma holds in the case where the basis is {AND}.

Finally we consider the case where $\mathcal{B} = \{\text{XOR}, \text{NXOR}\}$. We will consider each state to be an element of Z_2 and perform the arithmetic over Z_2. For each i, $1 \leq i \leq n$, f_i can be represented by a linear function over Z_2:

$$f_i(\boldsymbol{x}) = \left(\bigoplus_{j \in X_i} x_j\right) \oplus b_i$$

where $\oplus$ is the addition over Z_2, X_i is a set of all indices of variables involved in f_i, and $b_i = 1$ if f_i is equivalent to NXOR and 0 otherwise. By using the same adjacency matrix as before and using the column vector $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)^T$, we have:

$$\mathcal{F}(\boldsymbol{a}) = A\boldsymbol{a} \oplus \boldsymbol{b},$$

and so for all $k \geq 0$,

$$\mathcal{F}^k(\boldsymbol{a}) = A^k\boldsymbol{a} \oplus (A^{k-1} \oplus A^{k-2} \oplus \cdots \oplus I)\boldsymbol{b}, \qquad (1)$$

where I is the $n \times n$ identity matrix. By the standard iterated multiplication, for each $k, 0 \leq k \leq 2^n$, we can compute A^k in polynomial time. Thus, it suffices to show that the second term of Eq. (1) is computable in time polynomial in $n + \log k$.

Let $Q(k)$ denote the summation in question. Suppose k is a power of 2. Let $p = \log k$. We have $k = 2^p$ and

$$Q(k) = (A^{2^{p-1}} \oplus I)(A^{2^{p-2}} \oplus I) \cdots (A \oplus I).$$

Since $p = \log k$, by the iterative multiplication, we can compute all the components on the right-hand side in time polynomial in $n + \log k$, and so the left-hand side can be obtained in time polynomial in $n + \log k$.

Now suppose k is not a power of 2. There exist p and k' such that $2^p < k < 2^{p+1}$ and $k' = k - 2^p \leq k/2$. We have

$$Q(k) = Q(2^p) \oplus A^{2^p}Q(k').$$

Since $1 \leq k' < k/2$, this allows us to establish a recursive method for computing $Q(k)$. The depth of recursion is at most $\log k$, and each term of the form either $Q(2^m)$ or A^{2^m} during the recursion can be computed in time polynomial in $n + \log k$. Thus, $Q(k)$ can be computed in time polynomial in $n + \log k$. Hence, the claim holds, and the proof is complete. □

Theorem 2 can be proven using Lemma 1 as follows.

Proof of Theorem 2. To show that Convergence is polynomial-time computable, let $\mathcal{F}$ be an n-object synchronous BFDS over one of the three bases and let $\boldsymbol{a}$ be an initial state configuration. By the definition of convergence, we have that $\mathcal{F}$ converges on $\boldsymbol{a}$ if and only if $\mathcal{F}^{2^n-1}(\boldsymbol{a}) = \mathcal{F}^{2^n}(\boldsymbol{a})$ holds. By Lemma 1, we can compute $\mathcal{F}^{2^n-1}(\boldsymbol{a})$ and $\mathcal{F}^{2^n}(\boldsymbol{a})$ in polynomial time, and thus we complete the proof.

The following algorithm shows that PathIntersection($\mathcal{B}$) is in UP: Given $\mathcal{F}$, $\boldsymbol{a}$, and $\boldsymbol{b}$,

Step 1 Nondeterministically choose s, $0 \le s \le 2^n - 1$.
Step 2 Nondeterministically choose t, $0 \le t \le 2^n - 1$.
Step 3 Test whether $\mathcal{F}^s(\boldsymbol{a}) = \mathcal{F}^t(\boldsymbol{b})$. If the test fails, reject.
Step 4 If either $s = 0$ or $t = 0$, then accept. Otherwise, test whether $\mathcal{F}^{s-1}(\boldsymbol{a}) \ne \mathcal{F}^{t-1}(\boldsymbol{b})$. If the inequality holds, accept; otherwise, reject.

Clearly the algorithm runs in time polynomial in n. If the two state configuration paths intersect, then there is a unique combination of s and t for which the tests pass. Thus, the algorithm runs in UP.

4. Algorithm for CycleLength

In this section we prove the following theorem.

Theorem 3. *If $\mathcal{B}$ is one of* {AND}, {OR}, *and* {XOR, NXOR}, *then* CycleLength($\mathcal{B}$) *belongs to* UP ∩ coUP.

This result together with the latter statement of Theorem 2 can be used as evidence that for the bases mentioned in the theorems PathIntersection($\mathcal{B}$) and CycleLength($\mathcal{B}$) are unlikely to be NP-hard.

We first prove the following proposition.

Proposition 2. *Let $\mathcal{F}$ be an n-object BFDS and let $\boldsymbol{a}$ be an initial state configuration. For all integers $p \ge 0$ and $q \ge 1$, $\mathcal{F}$ on $\boldsymbol{a}$ has tail length p and cycle length q if and only if the following properties hold:*
(1) $\mathcal{F}^p(\boldsymbol{a}) = F^{p+q}(\boldsymbol{a})$.
(2) If $p > 0$, then $\mathcal{F}^{p-1}(\boldsymbol{a}) \ne F^{p+q-1}(\boldsymbol{a})$.
(3) For all prime numbers d dividing q, $\mathcal{F}^p(\boldsymbol{a}) \ne F^{p+q/d}(\boldsymbol{a})$.

Proof. Let $\mathcal{F}$ and $\boldsymbol{a}$ be as in the statement of the proposition. Suppose $\mathcal{F}$ on $\boldsymbol{a}$ enters a cycle at step p and the cycle length is q. Then, we have $\mathcal{F}^p(\boldsymbol{a}) = F^{p+q}(\boldsymbol{a})$. This is identical to Property 1 in the above. Also, by the minimality of p, we have: for all i, $0 \le i \le p-1$ and for all $j \ge i$, $\mathcal{F}^i(\boldsymbol{a}) \ne \mathcal{F}^j(\boldsymbol{a})$. By setting $i = p-1$ and $j = p+q-1$, we get Property 2. Finally, by the minimality of q, we have for all $i \ge 0$ and $s, 1 \le s \le q-1$, $\mathcal{F}^i(\boldsymbol{a}) \ne \mathcal{F}^{i+s}(\boldsymbol{a})$. In particular, if d is a prime number dividing q, then $q/d < q$, and so by setting $i = p$ and $s = q/d$, we have Property 3.

Conversely, suppose that one of the three properties in the statement of the proposition fails to hold for p and q. If Property 1 fails to hold, clearly q is not the cycle length. If Property 2 fails to hold, $\mathcal{F}^{p-1}(\boldsymbol{a}) = F^{p+q-1}(\boldsymbol{a})$, and so $\mathcal{F}$ on $\boldsymbol{a}$ enters a cycle earlier than step p. If Property 3 fails to hold, there is a divisor $e = p/d$ for some prime number d such that $\mathcal{F}^p(\boldsymbol{a}) = F^{p+e}(\boldsymbol{a})$. This implies that the cycle length is smaller than q.

This proves the proposition.

□

For a total function g, we say that g *is* UP-*computable* if there exists a polynomial-time nondeterministic Turing machine M such that for all inputs x, M on x accepts along exactly one computational path and in that unique computation path M on x outputs $g(x)$.

In the following lemma, we show that the cycle length is UP-computable, which immediately implies Theorem 3.

Lemma 2. *Suppose $\mathcal{B}$ is one of* {AND}, {OR}, *and* {XOR, NXOR}. *Then for all synchronous BFDS $\mathcal{F}$ and initial configurations $\boldsymbol{a}$, the tail length and the cycle length of $\mathcal{F}$ on $\boldsymbol{a}$ are* UP-*computable.*

Proof. Let $\mathcal{B}$ be one of {AND}, {OR}, and {XOR, NXOR}. Since the tail length p and the cycle length q are uniquely determined for each combination of $\mathcal{F}$ and $\boldsymbol{a}$ and since the prime factorization is in UP∩coUP [4], we can design a UP-algorithm for calculating p and q given $\mathcal{F}$ and $\boldsymbol{a}$ as follows:

Step 1 Our algorithm nondeterministically guesses p and q such that $0 \le p < 2^n$ and $1 \le q \le 2^n - p$.
Step 2 Using the algorithm presented in [4], we compute the prime factorization of q in UP. If the factorization is successful, the algorithm proceeds to the next step.
Step 3 Our algorithm tests the three properties in Proposition 2.
Step 4 Our nondeterministic algorithm accepts and outputs p and q if and only if all the tests pass.

The prime factorization part is carried out nondeterministically and since it is in UP, there is exactly one computation path along which the factorization is successfully obtained. Since $q \le 2^n$, the number of distinct prime factors of q is at most n. This implies that there will be at most $n + 2$ equalities to be tested in Step 3. Since both p and q are bounded from above by 2^n, we have from Lemma 1 that each equality can be tested in time polynomial in n. Thus, the above algorithm runs in time polynomial in n. The algorithm has exactly one accepting computation path for all $\mathcal{F}$ and $\boldsymbol{a}$, and on that unique accepting computation path computes p and q. Thus, the algorithm is an UP-algorithm.

This proves the lemma. □

References

[1] C. L. Barrett , H. S. Mortveit , C. M. Reidys. Elements of a theory of simulation II: Sequential dynamical systems. *Applied Mathematics and Computation*, **107**(2-3):121-136, 2000.

[2] C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. Complexity of reachability problems for finite discrete dynamical systems. *Journal of Computer and System Sciences*, **72**(8):1317-1345, 2006.

[3] C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns, and P. T. Tošić. Gardens of Eden and fixed points in sequential dynamical systems. In *Proceedings of Discrete Models: Combinatorics, Computation, and Geometry*, pages 95-110, 2001.

[4] M. R. Fellows and N. Koblitz. Self-witnessing polynomial-time complexity and prime factorization. In *Proceedings of the Seventh Annual*

Conference on Structure in Complexity Theory, pages 107-110, 1992.

[5] P. Floréen and P. Orponen. Complexity issues in discrete Hopfield networks. *Neuro-COLT Technical Report Series*, NC-TR-94-009, 1994.

[6] L. A. Hemaspaandra and M. Ogihara. *A Complexity Theory Companion*. Springer-Verlag, 2001.

[7] S. Kosub. Dichotomy results for fixed-point existence problems for boolean dynamical systems. *Mathematics in Computer Science*, **1**(3):487-505, 2008.

[8] R. Laubenbacher and B. Pareigis. Equivalence relations on finite dynamical systems. *Advances in Applied Mathematics*, **26**(3):237-251, 2001.

[9] A. Maruoka. *Concise Guide to Computation Theory*. Springer-Verlag, 2011.

[10] S. Kosub. and C. M. Homan. Dichotomy Results for Fixed Point Counting in Boolean Dynamical Systems. In *Proceedings of the Tenth Italian Conference on Theoretical Computer Science (ICTCS 2007)*, pages 163-174, 2007.

[11] I. Parberry. *Circuit Complexity and Neural Networks*. MIT Press, Cambridge, MA, 1994.

[12] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth ACM Symposium on Theory of Computing*, pages 216-226, 1978.