99

LA-UR-01-0668

Approved for public release; distribution is unlimited.

Title: PREDECESSOR AND PERMUTATION EXISTENCE PROBLEMS FOR SEQUENTIAL DYNAMICAL SYSTEMS

Author(s): Christopher L. Barrett, D-2
Harry B. Hunt III, D-2
Madhav V. Marathe, D-2
S. S. Ravi, University at Albany, NY
Daniel J. Rosenkrantz, University at Albany, NY
Richard E. Stearns, University at Albany, NY

Submitted to: International Conference on Symbolic and Algebraic Computations, University of Western Ontario, London, Ontario, Canada
January, 2001

Predecessor and Permutation Existence Problems for Sequential Dynamical Systems

CHRIS BARRETT [1] HARRY B. HUNT III [2,3] MADHAV V. MARATHE [1]
S. S. RAVI [2,3] DANIEL J. ROSENKRANTZ [2] RICHARD E. STEARNS [2]

Abstract

Motivated by the aim to build computer simulations for large scale systems [BMR99, BR99], we study a class of finite discrete dynamical systems called **Sequential Dynamical Systems** (SDSs). An SDS $\mathcal{S}$ is a triple $(G, \mathcal{F}, \pi)$, where (i) $G(V, E)$ is an undirected graph with n nodes with each node having a 1-bit state, (ii) $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$, with f_i denoting a symmetric Boolean function associated with node v_i and (iii) π is a permutation of (or total order on) the nodes in V. A configuration of an SDS is a bit vector $(b_1, b_2, \ldots, b_n)$, where b_i is the value of the state of node v_i. A single SDS transition from one configuration to another is obtained by updating the states of the nodes by evaluating the function associated with each of them in the order given by π. Here, we address the complexity of two problems for SDSs.

Given an SDS $\mathcal{S}$ and a configuration $\mathcal{C}$, the PREDECESSOR EXISTENCE (or PRE) problem is to determine whether there is a configuration $\mathcal{C}'$ such that $\mathcal{S}$ goes from $\mathcal{C}'$ to $\mathcal{C}$ in one step. We show that the PRE problem **NP**-complete for several simple classes of SDSs (e.g. SDSs for which the set of node functions is {AND, OR}, SDSs whose underlying graphs are planar). We also identify several classes of SDSs for which the PRE problem can be solved efficiently (e.g. SDSs where each node function is from {OR, NOR} or {AND, NAND} or {XOR, XNOR}). We also show that the PRE problem is solvable in polynomial time when the function at each node is linear or when the underlying graph G is of bounded treewidth. Many of the easiness results extend to the case where we want to find an ancestor configuration that precedes a given configuration by a polynomial number of steps.

Given the underlying graph $G(V, E)$, and two configurations $\mathcal{C}$ and $\mathcal{C}'$ of an SDS $\mathcal{S}$, the PERMUTATION EXISTENCE (or PME) problem is to determine whether there is a permutation of nodes that takes $\mathcal{S}$ from $\mathcal{C}'$ to $\mathcal{C}$ in one step. We show that the PME problem is **NP**-complete even when the function associated with each node is a simple-threshold function. We also show that a generalized version of the PME (GEN-PME) problem is **NP**-complete for SDSs where each node function is NOR and the underlying graph has a maximum node degree of 3. When each node computes the OR function or when each node computes the AND function, we show that the GEN-PME problem is solvable in polynomial time.

Our results extend some of the earlier results by Sutner [Su95] and Green [Gr87] on the complexity of the PREDECESSOR EXISTENCE problem for 1-dimensional cellular automata.

[1]Los Alamos National Laboratory, P. O. Box 1663, MS M997, Los Alamos, NM 87545. Email: {barrett, marathe}@lanl.gov. The work is supported by the Department of Energy under Contract W-7405-ENG-36.

[2]Department of Computer Science, University at Albany - SUNY, Albany, NY 12222. Email: {hunt, ravi, djr, res}@cs.albany.edu. Supported by a grant by Los Alamos National Laboratory and by NSF Grant CCR-97-34936.

[3]Part of the work was done while the authors were visiting scientists in the Basic and Applied Simulation Sciences Group (TSA-2) at the Los Alamos National Laboratory.

1 Introduction and Motivation

We study the computational complexity of some basic problems that arise in the context of a new class of discrete finite dynamical systems, called **Sequential Dynamical Systems** (henceforth referred to as SDS), proposed in [BR99, BMR99, BMR00]. A formal definition of such a system is given in Section 2. SDSs are closely related to classical Cellular Automata (CA), a widely studied class of dynamical systems in physics and complex systems. They are also closely related to a recently proposed extension of CA called **graph automata** [NR98, Ma98]. Decidability issues for dynamical systems in general and CA in particular have been widely studied in the literature [Wo86, Gu89]. In contrast, computational complexity questions arising in the study of CA and related dynamical systems have received comparatively less attention.

In simple terms, an SDS $\mathcal{S} = (G, \mathcal{F}, \pi)$ consists of three components. $G(V, E)$ is an undirected graph with n nodes with each node having a 1-bit state[4]. $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$, with f_i denoting a symmetric Boolean function associated with node v_i. π is a permutation of (or a total order on) the nodes in V. A **configuration** of an SDS is an n-bit vector $(b_1, b_2, \ldots, b_n)$, where b_i is the value of the state of node v_i $(1 \leq i \leq n)$. A single SDS transition from one configuration to another is obtained by updating the state of each node using the corresponding Boolean function. These updates are carried out in the order specified by π. The **phase space** of SDS $\mathcal{S}$, denoted by $\mathcal{P}_{\mathcal{S}}$, is a directed graph with one node for each of the 2^n possible configurations; there is a directed edge from the node representing configuration $\mathcal{C}'$ to that representing configuration $\mathcal{C}$ if $\mathcal{S}$ moves from $\mathcal{C}'$ to $\mathcal{C}$ in one transition.

The above definition of an SDS can be easily extended to allow (i) non-symmetric node functions, (ii) functions with ranges of cardinality larger than two and (iii) allowing a partial order on the nodes rather than a total order. See Section 2 for a discussion of such extensions.

The research reported here is a part of a program to provide a formal basis for the design and analysis of *large-scale computer simulations*, especially for socio-technical systems. Examples of such systems include various national infrastructures including transportation, power and communication. See `http://tsasa.lanl.gov` for additional details on this program. It is difficult to give a precise definition of a computer simulation that is applicable to the various settings where it is used. Nevertheless, an important aspect of any computer simulation is the generation of global dynamics by iterated composition of local mappings. Thus, we view simulations as comprised of a collection of entities with state values, local rules (functions) for state transitions, an interaction graph capturing the local dependency of an entity on other neighboring entities and an update sequence or schedule such that the causality in the system is modeled by the iterated composition of local functions. The informal description of SDS given above is seen to capture exactly these features.

We now discuss an example from [BB+99] to illustrate how an SDS-like model is used in the TRANSIMS (Transportation Analysis and Simulation System) project at the Los Alamos National Laboratory. For ease of exposition, we assume a single lane road which can be modeled as a one dimensional array of cells, with each cell representing a certain segment of the road. The state of each car (driver) may assume one of $v_{max} + 1$ possible values; these values correspond to discrete speeds from 0 to v_{max}. The state of each cell may assume one of $v_{max} + 2$ different values, the additional value being used to represent an empty cell. In the TRANSIMS system implementation, v_{max} was usually a small number (such as 5). At each instant, the behavior of a car (e.g. whether the speed increases, decreases or remains the same) is a function of its state

[4]The restriction to binary states is a mathematical convenience, and allows us to present stronger lower bound results.

and the state of the car that is immediately ahead. By associating a variable with each grid cell, the time evolution of the system can be cast as the time evolution of the corresponding SDS. An important point to note is that unlike CA (which are synchronous), the order of updating the cells yields completely different dynamics in many cases. For instance, updating the states from front to back acts like a perfect predictor and thus never yields clusters of vehicles. On the other hand, updating downstream yields more realistic traffic dynamics [BB+99].

In [BHM+00], we studied the complexity of determining some phase space properties (e.g. reachability of a given configuration, fixed point reachability) of SDSs. Here, we focus on the complexity of two basic problems for SDSs, namely PREDECESSOR EXISTENCE and PERMUTATION EXISTENCE. We now discuss these problems informally and defer the formal definitions to Section 2. Given an SDS $\mathcal{S} = (G, \mathcal{F}, \pi)$ and a configuration $\mathcal{C}$, the PREDECESSOR EXISTENCE (or PRE) problem is to determine whether the configuration $\mathcal{C}$ has a predecessor; that is, whether there is a configuration $\mathcal{C}'$ such that $\mathcal{S}$ moves from $\mathcal{C}'$ to $\mathcal{C}$ in one transition. Given a partially specified SDS $\mathcal{S}$ consisting of the underlying graph $G(V, E)$, the set $\mathcal{F}$ of Boolean functions associated with the nodes and two configurations $\mathcal{C}$ and $\mathcal{C}'$, the PERMUTATION EXISTENCE (or the PME) problem is to determine whether there is a permutation π of the nodes such that under permutation π, $\mathcal{S}$ moves from $\mathcal{C}'$ to $\mathcal{C}$ in one transition. The PRE problem is a classical problem studied by the dynamical systems community in the context of CA [Su95, Gr87]. The PME problem is important in the context of SDSs since two different node permutations may give rise to totally different behaviors of the underlying dynamical system. An investigation of these problems is helpful in obtaining a better understanding of the dynamical systems modeled by SDSs.

The remainder of the paper is organized as follows. In Section 2 we provide the necessary definitions. Section 3 summarizes our results and related results from the literature. Sections 4 and 5 present our results for PREDECESSOR EXISTENCE and PERMUTATION EXISTENCE problems respectively. Finally, Section 6 offers some concluding remarks.

2 Definitions and Problem Formulations

2.1 Sequential Dynamical Systems

We begin with a formal definition of sequential dynamical systems. Our definition closely follows the original definition of SDS in [BMR99, BMR00, MR99, Re00]. We also recall basic definitions of phase space parameters studied in this paper.

A **Sequential Dynamical System** (SDS) $\mathcal{S}$ is a triple $(G, \mathcal{F}, \pi)$, whose components are as follows:

1. $G(V, E)$ is an undirected graph without multi-edges or self loops. G is referred to as the **underlying graph** of $\mathcal{S}$. We use n to denote $|V|$ and m to denote $|E|$. The nodes of G are numbered using the integers $1, 2, \ldots, n$.

2. Each node has one bit of memory, called its **state**. The state of node i, denoted by s_i, takes on a value from $\mathbb{F}_2 = \{0, 1\}$. We use δ_i to denote the degree of node i. Each node i is associated with a *symmetric Boolean function* $f_i : \mathbb{F}_2^{\delta_i+1} \rightarrow \mathbb{F}_2$, $(1 \leq i \leq n)$. We refer to f_i as a **local transition function**. The inputs to f_i are the state of i and the states of the neighbors of i. By "symmetric" we mean that the function value does not depend on the order in which the input bits are specified; that is, the function value depends only on how many of its inputs are 1. We use $\mathcal{F}$ to denote $\{f_1, f_2, \ldots, f_n\}$.

3. Finally, π is a permutation of $\{1, 2, \ldots, n\}$ specifying the order in which nodes update their states using their local transition functions. Alternatively, π can be envisioned as a total order on the set of nodes.

Computationally, the transition of an SDS from one configuration to another involves the following steps:

```
for $i = 1$ to $n$ do
    (1) Node $\pi(i)$ evaluates $f_{\pi(i)}$. (This computation uses the current values of the state of $\pi(i)$ and those of the neighbors of $\pi(i)$.)
    (2) Node $\pi(i)$ sets its state $s_{\pi(i)}$ to the Boolean value computed in Step (1).
end-for
```

Stated another way, the nodes are processed in the *sequential* order specified by permutation π. The "processing" associated with a node consists of computing the value of the node's Boolean function and changing its state to the computed value.

We point out that the assumption of symmetric Boolean functions can be easily relaxed to yield more general SDSs. We give special attention to the symmetry condition for two reasons. First, our lower bounds for such SDSs imply stronger lower bounds for computing phase space properties of CA and communicating finite state machines (CFSMs). Second, symmetry provides one possible way to model "mean field effects" used in statistical physics and studies of other large-scale systems. A similar assumption has been made in [BPT91].

Recall that a configuration of an SDS is a bit vector $(b_1, b_2, \ldots, b_n)$. A configuration $\mathcal{C}$ of an SDS $\mathcal{S} = (G, \mathcal{F}, \pi)$ can also be thought of as a function $\mathcal{C} : V \to \mathbb{F}_2$. Given a configuration $\mathcal{C}$, the state of a node v in $\mathcal{C}$ is denoted by $\mathcal{C}(v)$; for a subset W of nodes, $\mathcal{C}(W)$ denotes the states of the nodes in W. We refer to $\mathcal{C}(W)$ as a **subconfiguration** of $\mathcal{C}$. The function computed by SDS $\mathcal{S}$, denoted by $F_{\mathcal{S}}$, specifies for each configuration $\mathcal{C}$, the next configuration $\mathcal{C}'$ reached by $\mathcal{S}$ after carrying out the update of node states in the order given by π. Thus, $F_{\mathcal{S}} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is a global function on the set of configurations. The function $F_{\mathcal{S}}$ can therefore be considered as defining the dynamic behavior of SDS $\mathcal{S}$. We also say that SDS $\mathcal{S}$ moves from a configuration $\mathcal{C}$ to a configuration $F_{\mathcal{S}}(\mathcal{C})$ in one time unit. The configuration reached by applying the global transition function for t time units to a configuration $\mathcal{C}$ is denoted by ${F_{\mathcal{S}}}^t(\mathcal{C})$.

2.2 Problems Considered

Given an SDS $\mathcal{S}$, let $|\mathcal{S}|$ denote the size of the representation of $\mathcal{S}$. In general, this includes the number of nodes, edges and the description of the local transition functions. When Boolean local transition functions are given as tables, $|\mathcal{S}| = O(m + |T|n)$, were $|T|$ denotes the maximum size of the table, n is the number of nodes and m is the number of edges in the underlying graph. For a node v with degree δ_v, the size of the table specifying an arbitrary Boolean function is $O(2^{\delta_v})$, while the size of the table specifying a symmetric Boolean function is $O(\delta_v)$. We assume that evaluating any local transition function given values for its inputs can be done in polynomial time.

In this paper, we study two basic problems and their extensions that arise in the context of SDSs. Some of these problems have been studied in the context of CA. We provide formal definitions of the problems below.

1. Given an SDS $\mathcal{S} = (G(V, E), \mathcal{F}, \pi)$ and a configuration $\mathcal{C}$, the PREDECESSOR EXISTENCE problem (abbreviated as PRE) is to determine whether there is a configuration $\mathcal{C}'$ such that $F_{\mathcal{S}}(\mathcal{C}') = \mathcal{C}$.

2. Given a partially specified SDS $\mathcal{S}$ consisting of graph $G(V, E)$, the set $\mathcal{F}$ of symmetric Boolean functions associated with the nodes of G, an initial configuration $\mathcal{C}'$ and a final configuration $\mathcal{C}$, the PERMUTATION EXISTENCE problem (abbreviated as PME) is to determine whether there is a permutation π for $\mathcal{S}$ such that $F_{\mathcal{S}}(\mathcal{C}') = \mathcal{C}$.

As stated, the PRE problem asks for an immediate predecessor; that is, whether there is a configuration $\mathcal{C}'$ from which a given configuration $\mathcal{C}$ can be reached in one transition. It is possible to generalize the problem to the case where we are given an integer $t \geq 1$, and the goal is to determine whether there is a configuration $\mathcal{C}'$ from which $\mathcal{C}$ can be reached in exactly t transitions. We call this the t-PRE problem.

Given an SDS $\mathcal{S}$= $(G(V, E), \mathcal{F}, \pi)$, let $W = \{v_{i_1}, \ldots, v_{i_k}\}$ be a subset of nodes in V. The states of nodes in W can be represented as the bit vector $(b_{i_1}, b_{i_2}, \ldots, b_{i_k})$. Green [Gr87] considered the following three problems in the context of infinite CA.[5] These problems can be viewed as extensions of the t-PRE problem.

3. Given an SDS $\mathcal{S} = (G(V, E), \mathcal{F}, \pi)$, a subset of nodes W, a bit vector $B = (b_{i_1}, b_{i_2}, \ldots, b_{i_k})$ that specifies the state values for the nodes in W and an integer $t \geq 1$, the t-SUB-CONFIGURATION PREDECESSOR EXISTENCE problem (abbreviated as t-SUB-PRE) is to determine whether there are configurations $\mathcal{C}'$ and $\mathcal{C}$ such that ${F_{\mathcal{S}}}^t(\mathcal{C}') = \mathcal{C}$ and B is a subconfiguration of $\mathcal{C}$.

4. Given an SDS $\mathcal{S} = (G(V, E), \mathcal{F}, \pi)$, a subset of nodes W, a bit vector $B = (b_{i_1}, b_{i_2}, \ldots, b_{i_k})$ that specifies the state values for the nodes in W and an integer $t \geq 1$, the t-SUB-CONFIGURATION RECURRENCE problem (abbreviated as t-SUB-RECUR) is to determine whether there are configurations $\mathcal{C}'$ and $\mathcal{C}$ such that ${F_{\mathcal{S}}}^t(\mathcal{C}') = \mathcal{C}$ and B is a subconfiguration of both $\mathcal{C}'$ and $\mathcal{C}$ (i.e., whether the subconfiguration represented by B will occur again after exactly t time steps).

5. Given an SDS $\mathcal{S} = (G(V, E), \mathcal{F}, \pi)$, an integer $t \geq 1$ and a temporal sequence $\langle\, b_v(1),\, b_v(2), \ldots, b_v(t) \,\rangle$ of t state values of a given node v, the t-TEMPORAL SEQUENCE PREDECESSOR EXISTENCE problem (abbreviated as t-TEMP-SEQ-PRE) is to determine whether there is a configuration $\mathcal{C}$ such that ${F_{\mathcal{S}}}^j(\mathcal{C})(v) = b_v(i),\ 1 \leq j \leq t$.

Another way of thinking about subconfigurations is to assume that a configuration specifies Boolean state values for some nodes and "don't care" values for other nodes. The corresponding subconfiguration is obtained by retaining only the Boolean values. Using this idea, it is possible to formulate a generalized version of the PME problem as follows.

6. Given a partially specified SDS $\mathcal{S}$ consisting of graph $G(V, E)$, the set $\mathcal{F}$ of symmetric Boolean functions associated with the nodes of G, an initial configuration $\mathcal{C}'$ and a final configuration $\mathcal{C}$ possibly containing don't care values, the GENERALIZED PERMUTATION EXISTENCE problem (abbreviated as GEN-PME) is to determine whether there is a permutation π for $\mathcal{S}$ such that $F_{\mathcal{S}}(\mathcal{C}')$ and $\mathcal{C}$ agree in all the components of $\mathcal{C}$ that have Boolean values. (In other words, $\mathcal{C}$ specifies succinctly a set of configurations, and $F_{\mathcal{S}}(\mathcal{C}')$ may be any one of these configurations.)

[5]The actual definitions in [Gr87] are slightly different. Since the main goal of [Gr87] was to prove **NP**-completeness, the problems were formulated with time t equal to the number of nodes in the subconfiguration.

A summary of our results for these problems is provided in Section 3.

2.3 Variants of SDS

As mentioned earlier, the definition of an SDS can be extended to obtain several variants. A brief description of these SDS variants is given below.

As defined, the state of an SDS is a Boolean value and the functions associated with the nodes are symmetric and Boolean. When we allow the state of each node to to take on values from a finite domain and the node functions to produce values from the domain, we obtain a **Finite Range SDS** (FR-SDS). If the states may store unbounded values and the node functions may also produce unbounded values, we obtain a **Generalized SDS** (Gen-SDS).

A **Linear SDS** is one in which each local transition function is a linear combination of its inputs. To be more precise, consider each node v_i, and let $N(i) = \{v_{i_1}, v_{i_2}, \ldots, v_{i_r}\}$ denote the neighbors of v_i. Let $N'(i) = N(i) \cup \{v_i\}$. In a linear SDS, each local transition function f_i has the following form:

$$f_i(s_i, s_{i_1}, \ldots s_{i_r}) = \alpha_i + \sum_{v_j \in N'(i)} a_{ij} s_j. \qquad (1)$$

Here, α_i and a_{ij} ($1 \leq i \leq n$ and $1 \leq j \leq r$) are (scalar) constants, s_j is the state value of node v_j and the arithmetic operations (addition and scalar multiplications) are assumed to be carried out over a *field.* We assume that the field operations can be carried out efficiently. Under this assumption, it is well known (see for example [Von93]) that solving a set linear equations over the field can be done in polynomial time. We use this fact in Section 4.2. When the state of each node is Boolean, each linear local transition function is either XOR (exclusive or) or XNOR (the complement of exclusive or).

A **Synchronous Dynamical System** (SyDS) is an SDS *without* the node permutation. In an SyDS, in each time step, all the nodes synchronously compute and update their state values. Thus, SyDSs are similar to CA. We can extend the definition of an SyDS to obtain an FR-SyDS and a Gen-SyDS in a manner similar to that of SDS.

2.4 Other Relevant Definitions

Two special classes of Boolean functions considered in this paper are that of k-**simple-threshold functions** and **exactly-k-functions**. We provide formal definitions of these classes below.

Definition 2.1 *1. The* k-**simple-threshold** *(Boolean) function has value 1 if at least k of the inputs have value 1; otherwise, the value of the function is zero.*

2. The **exactly-k** *(Boolean) function has value 1 if exactly k of the inputs have value 1; otherwise, the value of the function is zero.*

It should be noted that k-simple-threshold functions are a special case of threshold functions [Ko70]. An SDS in which each local transition function is a k-simple-threshold (exactly-k) function for some k is referred to as a **simple-threshold-SDS (exact-SDS)**. Finally, SDSs in which all nodes compute a transition function f of the same type (but not necessarily of the same arity) are referred to as f-SDSs. Thus for example, OR-SDSs are SDSs in which each local transition function is the OR function. Extending this notation, an (f, g)-SDS is an SDS in which each local transition function is either of type f or of type g.

In this paper, **NP**-completeness results are established using reductions from variants of the Satisfiability (SAT) problem. An instance of SAT is specified by a collection $X = \{x_1, x_2, \ldots, x_n\}$ of n Boolean variables and a collection $C = \{c_1, c_2, \ldots, c_m\}$ of m clauses, where each clause is a set of literals. The question is whether there is an assignment of Boolean values to the variables so that each clause is satisfied (i.e., contains at least one true literal). The **bipartite graph** BG associated with an instance of SAT has one node for each variable and one node for each clause; there is an edge between a variable node and a clause node if the variable occurs (positively or negatively) in the clause. Definitions of the various forms of SAT used in the paper are given below. Each of these variants is known to be **NP**-complete [GJ79, DF86].

Definition 2.2 *(a)* 3SAT *is the restricted version of SAT in which each clause contains exactly three literals.*

(b) 3SAT-2OCCUR *is the restricted version of SAT in which each clause contains exactly three literals and each literal occurs in at most two clauses.*

(c) MONOTONE 3SAT *is the restricted version of SAT in which each clause contains exactly three positive (unnegated) literals or exactly three negated literals.*

(d) In PLANAR POSITIVE EXACTLY 1-IN-3 3SAT *(abbreviated as* PL-PE3SAT*), each clause contains exactly three positive literals, the associated bipartite graph is planar, and the question is whether there is a truth assignment to the variables such that each clause contains exactly one true literal.*

Finally, we recall the concept of *treewidth.*

Definition 2.3 *[ALS91, Bo88, RS86] Let $G(V, E)$ be a graph. A* **tree-decomposition** *of G is a pair $(\{X_i \mid i \in I\}, T = (I, \mathcal{F}))$, where $\{X_i \mid i \in I\}$ is a family of subsets of V and $T = (I, \mathcal{F})$ is a tree with the following properties:*

1. $\bigcup_{i\in I} X_i = V$.

2. *For every edge $e = (v, w) \in E$, there is a subset X_i, $i \in I$, with $v \in X_i$ and $w \in X_i$.*

3. *For all $i, j, k \in I$, if j lies on the path from i to k in T, then $X_i \bigcap X_k \subseteq X_j$.*

The **treewidth** *of a tree-decomposition $(\{X_i \mid i \in I\}, T)$ is $\max_{i\in I}\{|X_i| - 1\}$. The treewidth of a graph is the minimum over the treewidths of all tree decompositions. A class of graphs is* **treewidth bounded** *if there is a constant k such that the treewidth of every graph in the class is at most k.*

A number of graph classes are known to have bounded treewidth. They include k-outerplanar graphs, k-bandwidth bounded graphs (both for constant k), series parallel graphs, Halin graphs, chordal graphs of bounded clique size, etc. For many optimization problems that are **NP**-hard for general graphs, optimal solutions can be computed in polynomial time when attention is restricted to the class of treewidth-bounded graphs. A considerable amount of work has been done in this area (see [ALS91, Bo88, RS86] and the references therein).

3 Summary of Results and Related Work

3.1 PREDECESSOR EXISTENCE Problem

Sutner [Su95] and Green [Gr87] considered the PRE problem and its generalizations in the context of CA. Their work motivated our study the PRE problem for SDSs.

We show that the PRE problem is **NP**-complete for any of the following restricted classes of SDSs: (i) Each node of the SDS computes the k-simple-threshold function, for any $k \geq 2$, (ii) each node computes the exactly-k function for any $k \geq 1$, (iii) the set of node functions used in the SDS is $\{$OR, AND$\}$ and (iv) SDSs whose underlying graphs are planar. We present polynomial algorithms for the PRE problem for SDSs for which the set of node functions is any non-empty subset of one of the following sets: $\{$OR, NOR$\}$, $\{$AND, NAND$\}$ and $\{$XOR, XNOR$\}$. Note the interesting contrast between this easiness result and the hardness result (iii) above. These results can be extended to obtain polynomial time algorithms for the t-SUB-PRE, t-SUB-RECUR and t-TEMP-SEQ-PRE problems when when t is polynomial in $|S|$.

We also show that when the underlying graph of an FR-SDS (or an FR-SyDS) is of bounded treewidth, the PRE problem can be solved efficiently with *no restrictions on the node functions* other than that the function evaluation can be done efficiently. This result can be extended to obtain polynomial time algorithms for the t-SUB-PRE, t-SUB-RECUR and t-TEMP-SEQ-PRE problems when either (i) $t = O(\log n)$ and the domain of state values is of constant size or (ii) when t is a constant and the domain of state values is bounded by a polynomial in $|S|$. Many of these polynomial time algorithms are obtained by reducing the corresponding problem to a generalized satisfiability problem [Sc78] that can be solved in polynomial time.

Our results extend the earlier work of Sutner [Su95] and Green [Gr87] on the complexity of the PREDECESSOR EXISTENCE problem for CA. For instance, our results on polynomial time solvability when restricted to the class of treewidth bounded graphs extend Sutner's result since CA can be easily seen to have bounded treewidth (in fact, they are of bounded bandwidth). Second, the results also show that Green's **NP**-completeness results are close to being tight since the corresponding problems are efficiently solvable when $t = O(\log n)$. Finally, our polynomial time results can be extended to solve a number of other variants when instances are restricted to treewidth bounded graphs. Examples other than those mentioned above include the problem of finding a predecessor with maximum (or minimum) number of state values being 1 (or 0), constraining the states of a specified subset of nodes, etc.

3.2 The PERMUTATION EXISTENCE Problem

The PME problem is unique to SDSs since state updates in CA are carried out in a synchronous fashion. As mentioned in Section 1, the motivation for the PME problem comes from the fact that the behavior of an SDS under two different permutations may be very different.

We show that the PME problem is **NP**-complete even when each local transition function is a simple-threshold function. We also show that the GEN-PME problem is **NP**-complete for any of the following restricted classes of SDSs: (i) NOR-SDSs (or NAND-SDSs) where the underlying graph has maximum node degree of 3 and (ii) SDSs whose underlying graphs are planar. We present polynomial time algorithms for the GEN-PME problem for OR-SDSs and AND-SDSs. We also present polynomial time algorithms for the PME problem (without don't care values) for NOR-SDSs and NAND-SDSs. These results show an interesting contrast between the complexity of GEN-PME and PME problems for NOR-SDSs and NAND-SDSs.

3.3 Previous Work

Computational aspects of CA have been studied by a number of researchers; see for example [Mo90, Mo91, CPY89, Wo86, Gu89, Gr87, Su95]. Much of this work addresses decidability of properties for infinite CA. Barrett, Mortveit and Reidys [BMR99, BMR00, MR99, Re00, Re00a] and Laubenbacher and Pareigis [LP00] investigate the mathematical properties of sequential dynamical systems. The PRE problem was shown to be **NP**-complete for finite CA by Sutner [Su95] and Green [Gr87]. Sutner also showed that PRE problem for finite 1-dimensional CA with a fixed neighborhood radius can be solved in polynomial time. As mentioned earlier, Green [Gr87] studied generalized versions of the PRE problem for infinite CA. The problems were so formulated that his results are also applicable to finite 1-dimensional CA. References [Su95, Gr87] do not consider other restrictions on CA that lead to polynomial algorithms for the PRE problem. Our approach, which relies on an efficient reduction to the generalized satisfiability problem, allows us to identify a number of restricted classes of SDSs for which the PRE problem can be solved efficiently.

4 The PREDECESSOR EXISTENCE Problem

4.1 NP-Completeness Results

Theorem 4.1 *The* PRE *problem is* **NP***-complete for the following classes of SDSs:*

1. *k-simple-threshold-SDSs, for each $k \geq 2$.*
2. *Exactly-k-SDSs, for each $k \geq 1$.*
3. *SDSs for which the set of local transition functions is $\{AND, OR\}$.*
4. *SDSs whose underlying graphs are planar.*

Proof: It is obvious that PRE is in **NP**. We establish the **NP**-hardness of PRE for each of the above cases through an appropriate reduction from a restricted version of SAT. In each case, we assume that corresponding SAT instance has n variables and m clauses.

Part 1: We use a reduction from 3SAT and construct an SDS S as follows. First, the underlying graph $G(V, E)$ has the following vertices and edges. $V = V_1 \cup V_2 \cup V_3$, where $V_1 = \{a_1, a_2 \ldots, a_k\}$, $V_2 = \{x_i, \overline{x_i}, y_i \mid \text{for each variable } x_i\}$ and $V_3 = \{c_j, d_j \mid \text{for each clause } c_j\}$. The set E has the following edges.

1. For each p, q, $1 \leq p < q \leq k$, the edge $\{a_p, a_q\}$. (Thus, the k nodes in V_1 form a clique.)
2. For each i, $1 \leq i \leq n$, edges $\{x_i, y_i\}$ and $\{\overline{x_i}, y_i\}$.
3. For each j, $1 \leq j \leq m$, edge $\{c_j, d_j\}$, and an edge from c_j to the nodes for each of the three literals occurring in clause c_j.
4. For each p, i, $1 \leq p \leq k-2$ and $1 \leq i \leq n$, the edge $\{a_p, y_i\}$.
5. For each p and i, $1 \leq p \leq k$ and $1 \leq i \leq n$, edges $\{a_p, x_i\}$ and $\{a_p, \overline{x_i}\}$.
6. For each p and j, $1 \leq p \leq k-1$ and $1 \leq j \leq m$, edges $\{a_p, d_j\}$ and $\{a_p, c_j\}$.

The permutation π is given by

$$\pi = (a_1, \ldots, a_k, y_1, \ldots, y_n, d_1, \ldots, d_m, c_1, \ldots, c_m, x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}).$$

The required final configuration $\mathcal{C}$ has value 1 for each a_p, 0 for each y_i, 0 for each d_j, 1 for each c_j, 1 for each x_i, and 1 for each $\overline{x_i}$.

Suppose that there is a configuration $\mathcal{C}'$ such that $\mathcal{S}$ can reach $\mathcal{C}$ in one transition from $\mathcal{C}'$. Setting the initial value of each a_p to 1 will ensure that each $\mathcal{C}(a_p) = 1$. Because $\mathcal{C}(y_i) = 0$, at most one of x_i and $\overline{x_i}$ has initial value 1. Because $\mathcal{C}(d_j) = 0$, the initial value of each c_j must be 0. Because $\mathcal{C}(c_j) = 1$, the initial value for at least one of the three literals occurring in clause c_j must be 1. Because each x_i and $\overline{x_i}$ is connected to all the a_p's, each of which has a final value of 1, the required final value of 1 for x_i and $\overline{x_i}$ imposes no restriction on initial values.

Using the above observations, it can be verified that the 3SAT instance has a solution if and only if the constructed PRE instance has a solution.

Part 2: The reduction is again from 3SAT. The underlying graph $G(V, E)$ has the following nodes and edges. $V = V_1 \cup V_2 \cup V_3$, where $V_1 = \{a_1, a_2, \ldots, a_k, b\}$, $V_2 = \{c_j \mid \text{for each clause } c_j\}$ and $V_3 = \{x_i, \overline{x_i}, z_i, y_{i,1}, y_{i,2}, \ldots, y_{i,k} \mid \text{for each variable } x_i\}$. (Note that V_3 has k+3 nodes for each variable x_i, $1 \le i \le n$.) The edge set E consists of the following.

1. For each p, q such that $1 \le p < q \le k$, the edge $\{a_p, a_q\}$. (Thus, these k nodes form a clique.)

2. For each p, $1 \le p \le k$, the edge $\{a_p, b\}$.

3. For each p, j, $1 \le p \le k - 1$ and $1 \le j \le m$, the edge $\{a_p, c_j\}$.

4. For each j, $1 \le j \le m$, the edge $\{b, c_j\}$, and an edge from c_j to the nodes for each of the three literals occurring in clause c_j.

5. For each i, $1 \le i \le n$, edges $\{x_i, z_i\}$ and $\{\overline{x_i}, z_i\}$.

6. For each i and p, $1 \le i \le n$ and $1 \le p \le k$, the edges $\{x_i, y_{i,p}\}$ and $\{\overline{x_i}, y_{i,p}\}$.

7. For each i, p, and q, such that $1 \le i \le n$ and $1 \le p < q \le k$, the edge $\{y_{i,p}, y_{i,q}\}$. (Thus, each set of k nodes $\{y_{i,1}, \ldots, y_{i,k}\}$ forms a clique.)

8. For each i and p, $1 \le i \le n$ and $1 \le p \le k - 1$, the edge $\{z_i, y_{i,p}\}$.

The permutation π is as follows.

$$(a_1, \ldots, a_k, b, c_1, \ldots, c_m, y_{1,1}, y_{1,2}, \ldots, y_{1,k}, z_1, y_{2,1}, y_{2,2}, \ldots, y_{2,k}, z_2, \ldots,$$
$$y_{n,1}, y_{n,2}, \ldots, y_{n,k}, z_n, x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}).$$

The required final state $\mathcal{C}$ has value 1 for each a_p, 1 for b, 0 for each c_j, 1 for each $y_{i,p}$, 1 for each z_i, 0 for each x_i, and 0 for each $\overline{x_i}$.

Suppose that there is a configuration $\mathcal{C}'$ such that $\mathcal{S}$ can reach $\mathcal{C}$ in one step. Setting the initial value of each a_p to 1 will ensure that each $\mathcal{C}(a_p) = 1$. Next, consider node b. Because $\mathcal{C}(b) = 1$, and the k nodes that precede b in π are all connected to b and have final values of 1, the initial value of b and all the clause nodes is forced to be 0. Next, consider each node c_j. Of the nodes that precede c_j in π, exactly k are connected to

c_k and have final value 1. Since the initial and the final values of c_j are both 0, the initial value of at least one of the three literals occurring in clause c_j must be 1. Next, consider each node $y_{i,k}$, with final value 1. Because all the $k-1$ nodes $y_{i,1}, \ldots, y_{i,k-1}$ that precede $y_{i,k}$ in π, have final value 1, and are connected to $y_{i,k}$, at most one of x_i and $\overline{x_i}$ can have an initial value of 1. The purpose of z_i is to ensure that x_i and $\overline{x_i}$ will have final value 0.

Part 3: The reduction is from MONOTONE 3SAT. The constructed graph $G(V, E)$ has $V = \{x_1, x_2, \ldots, x_n, c_1, c_2, \ldots, c_m, a, b, d\}$. The edges in E are as follows.

1. For each i, $1 \leq i \leq n$, the edge $\{a, x_i\}$.
2. For each i, j, $1 \leq i \leq n$, $1 \leq j \leq m$, the edge $\{x_i, c_j\}$ whenever the literal x_i or $\overline{x_i}$ appears in clause c_j.
3. For each j, $1 \leq j \leq m$, the edge $\{b, c_j\}$ if c_j has all positive literals.
4. For each j, $1 \leq j \leq m$, the edge $\{d, c_j\}$ if c_j has all negative literals.

The node functions are as follows. For a, b, each node x_i and each clause c_j with only positive literals, the function is OR; for d and each clause c_j with only negative literals, the function is AND. The permutation π is $(a, b, d, c_1, c_2, \ldots, c_m, x_1, x_2, \ldots, x_n)$. The required final configuration $\mathcal{C}$ has $\mathcal{C}(a) = 1$, $\mathcal{C}(b) = 0$, $\mathcal{C}(d) = 1$, $\mathcal{C}(c_j) = 1$ if c_j has all positive literals, $\mathcal{C}(c_j) = 0$ if c_j has all negative literals and $\mathcal{C}(x_i) = 1$, for $1 \leq i \leq n$.

We can now argue that the PRE instance has a solution if and only if the instance of MONOTONE 3SAT has a solution. The reason is as follows. The initial value $\mathcal{C}(b) = 0$ forces the initial value of each clause containing only positive literals to be 0. The initial value $\mathcal{C}(d) = 1$ forces the initial value of each clause containing only negative literals to be 1. Since each positive literal clause has initial value 0 and final value 1, at least one of the variables in the clause must have initial value 1. Similarly, since each negative literal clause has initial value 1 and final value 0, at least one of the variables in the clause must have initial value 0. Node a enables each of the variable nodes to have the final value 1.

Part 4: The reduction is from PL-PE3SAT. We create the following SDS $\mathcal{S}$. For each variable $x_i \in X$, $\mathcal{S}$ has one node (denoted by x_i), $1 \leq i \leq n$. For each clause $c_j \in C$, $\mathcal{S}$ has two nodes (denoted by c_j and c'_j), $1 \leq j \leq m$. There is an edge between c_j and c'_j for each j, $1 \leq j \leq m$. Further, if the clause c_j contains positive literals x_{j_1}, x_{j_2} and x_{j_3}, then there are edges between c_j and x_{j_r} for $r = 1, 2, 3$. This completes the specification for the undirected graph of $\mathcal{S}$. Note that underlying graph of the resulting SDS is planar since it is obtained from the (planar) bipartite graph corresponding to the PL-PE3SAT instance by simply attaching a node c'_j of degree 1 to each clause node c_j $(1 \leq j \leq m)$.

The symmetric Boolean functions associated with nodes are as follows. For each node x_i $(1 \leq i \leq n)$ and c'_j $(1 \leq j \leq m)$, the associated Boolean function is the logical OR function. For the node c_j $(1 \leq j \leq m)$, the associated Boolean function takes on the value 1 if exactly one of the inputs is 1; otherwise, the function value is 0.

The permutation π for $\mathcal{S}$ is $(c'_1, c'_2, \ldots, c'_m, c_1, c_2, \ldots, c_m, x_1, x_2, \ldots, x_n)$. The required final configuration $\mathcal{C}$ has value 0 for each node c'_j $(1 \leq j \leq m)$ and 1 for all other nodes.

We now show that there is a configuration $\mathcal{C}'$ such that $\mathcal{S}$ can reach the configuration $\mathcal{C}$ in one transition from $\mathcal{C}'$ if and only if the PL-PE3SAT instance is satisfiable.

Suppose there is a satisfying truth assignment to the PL-PE3SAT instance. We construct the following configuration C': For each of the nodes c_j and c'_j ($1 \leq j \leq m$), the initial state is set to 0. For each node x_i ($1 \leq i \leq n$), the initial state is set to the truth value given by the satisfying assignment to the PL-PE3SAT instance. Using the permutation π, it is straightforward to verify that starting from C', $\mathcal{S}$ reaches C in one step.

Now, suppose there is a configuration C' such that $\mathcal{S}$ can reach the configuration C in one step. We claim that C' must set the state of each node c_j and c'_j ($1 \leq j \leq m$) to 0. To see this, suppose C' initializes some node c_j (or c'_j) to 1. Since c'_j appears before c_j in the permutation, when c'_j executes, its value would become 1. This is a contradiction since C specifies the value of c'_j to be 0. The claim follows.

In view of the claim, when a node c_j executes, its initial value is 0. Therefore, exactly one of the nodes corresponding to the variables in clause c_j must be set to 1 by C' so that the final value of c_j becomes 1. In other words, the values assigned by C' to the states of the nodes x_1, x_2, ..., x_n must form a satisfying assignment to the PL-PE3SAT. This completes the proof of Part 4 and also that of the theorem. ■

4.2 Polynomial Time Results

We now present polynomial algorithms for the PRE problem for restricted classes of SDSs. Our algorithms are based on efficiently reducing the PRE problem and its variants to polynomial time solvable instances of systems of linear equations or generalized satisfiability problems. As will be seen, whenever the answer to a problem is "yes", our algorithms can also generate a feasible solution with the desired property.

We begin with our results for linear Gen-SDSs using the form of local transition functions given in Equation (1).

Theorem 4.2 *Let $\mathcal{S}$ be a linear Gen-SDS with n nodes such that for each i, $1 \leq i \leq n$, the scalar constant a_{ii} used in the expression for the local transition function f_i is nonzero. For such a linear Gen-SDS, the answer to the* PRE *problem is always "yes". Moreover, for a given final configuration C, there is a* unique *predecessor configuration C', which can be found in time linear in the size of the underlying graph.*

Proof: Let $C = (b_1, b_2, \ldots, b_n)$ denote the required final configuration. To solve the PRE problem for $\mathcal{S}$, we associate a variable x_i with each node v_i of $\mathcal{S}$ and construct a system of linear equations over the algebraic field corresponding to $\mathcal{S}$. This construction is done in such a way that any solution to the system of linear equations provides a solution to the PRE problem for $\mathcal{S}$. When the condition $a_{ii} \neq 0$ is satisfied for each i, we show that the resulting system of equations has a unique solution.

To construct the system of linear equations, consider the node v_i. Let $N(i)$ denote the set of neighbors of v_i. In $N(i)$, let v_{i_1}, v_{i_2}, ..., v_{i_r} denote the nodes that *precede* v_i in the permutation and let v_{j_1}, v_{j_2}, ..., v_{j_p} denote the nodes that *follow* v_i in the permutation. Using Equation (1), the linear equation for v_i, where the arithmetic operations are carried out over the field corresponding to $\mathcal{S}$, is the following:

$$\alpha_i + a_{ii}x_i + \sum_{q=1}^{r} a_{ii_q}b_{i_q} + \sum_{q=1}^{p} a_{ij_q}x_{j_q} \;=\; b_i. \tag{2}$$

There is one such equation for each node v_i. It can be verified that any solution to the above system of equations over the field corresponding to $\mathcal{S}$is a solution to the PRE problem.

The above construction produces n equations in n unknowns. Suppose that we envision the nodes being enumerated in reverse order of π. Then the n equations are in triangular form, and such a system of equations has a unique solution.

When node v_i is being considered, for all nodes v_j with $j < i$ (i.e., $\pi(v_j) > \pi(v_i)$), the unique value $C'(v_j)$ has already been determined. Therefore, in the equation for determining the new value of v_i, the only unknown is $C'(v_i)$. This is because the other values in the equation are from C for neighboring nodes before v_i in π, the already computed valued from C for neighboring nodes after v_i in π, and $C(v_i)$. Since the entry a_{ii} is not zero, this equation has a unique solution. ■

For linear Gen-SDSs that do not satisfy the condition mentioned in Theorem 4.2 and for linear Gen-SyDSs, the linear equation approach can be used to obtain an efficient algorithm to determine whether the PRE problem has a solution. This is shown in the next theorem.

Theorem 4.3 *The* PREDECESSOR EXISTENCE *problem for linear Gen-SDSs and linear Gen-SyDSs can be solved in polynomial time.*

Proof: First consider a linear Gen-SDS. Using the steps mentioned in the proof of Theorem 4.2, we construct a system of equations. When one or more of the a_{ii} entries are zero, the resulting system may not have a solution or may have multiple solutions. Since the feasibility of any system of linear equations over a field can be determined in polynomial time [Von93], it follows that the PRE problem for linear SDSs can be solved in polynomial time.

For linear Gen-SyDSs, since the nodes update their states synchronously, the form of linear equations is slightly different from the one given in Equation 2. Using $N'(i)$ to denote the set consisting of node v_i and its neighbors, the equation for node v_i in the case of a linear Gen-SyDS is as follows.

$$\alpha_i + \sum_{v_q \in N'(i)} a_{iq} x_q \;=\; b_i. \tag{3}$$

Again, the feasibility of the set of linear equations can be determined in polynomial time. ■

As mentioned earlier, when the state values are Boolean, XOR and XNOR are linear functions over the field $\mathbb{F}_2$ under addition modulo 2. Thus, by Theorem 4.2, for any SDS where each local transition function is from the set {XOR, XNOR}, the PRE problem has a unique solution which can be found efficiently.

An approach similar to that used in the proof of Theorems 4.2 and 4.3 can be used to obtain polynomial algorithms for the PRE problem for other restricted classes of SDSs characterized by specific (Boolean) local transition functions. The idea is to efficiently reduce the PRE problem for such SDSs to a polynomial time solvable version of the Satisfiability (SAT) problem for Boolean formulas. We now outline this reduction.

Let $\mathcal{S}$ denote the given linear SDS and let $C = (b_1, b_2, \ldots, b_n)$ denote the required final configuration, with $b_i \in \{0, 1\}$. To solve the PRE problem for $\mathcal{S}$, we associate a Boolean variable x_i with each node v_i of $\mathcal{S}$ and construct a set $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ of terms, with term T_i corresponding to node v_i of $\mathcal{S}$. To construct term T_i, consider the node v_i with local transition function f_i. Let $N(i)$ denote the set of neighbors of v_i. In $N(i)$, let v_{i_1}, v_{i_2}, …, v_{i_r} denote the nodes that *precede* v_i in the permutation and let v_{j_1}, v_{j_2}, …, v_{j_p} denote the nodes that *follow* v_i in the permutation. We first construct the formula $F_i = f_i(x_i, b_{i_1}, b_{i_2}, \ldots, b_{i_r}, x_{j_1}, x_{j_2}, \ldots, x_{j_p})$. If $b_i = 1$, the term T_i is F_i itself; if $b_i = 0$, the term T_i is $\overline{F_i}$. The resulting instance of SAT is the conjunction of the terms in $\mathcal{T}$. It can be verified that the PRE problem for $\mathcal{S}$ has a solution if and only if the resulting SAT instance has a solution.

We need the following concepts introduced by Schaefer [Sc78] to state and prove our next few results.

Definition 4.1 *([Sc78])*
Let $\mathsf{S} = \{R_1, R_2, \cdots, R_m\}$ be a finite set of finite arity Boolean relations. (A Boolean relation is defined to be any subset of $\{0,1\}^p$ for some integer $p \geq 1$. The integer p is called the **rank** *or* **arity** *of the relation.) An* S*-formula is a conjunction of clauses each of the form $\hat{R}_i(\xi_1, \xi_2, \cdots)$, where $\xi_1, \xi_2, \cdots$ are distinct, unnegated variables whose number matches the rank of $R_i, i \in \{1, \cdots, m\}$ and $\hat{R}_i$ is the relation symbol representing the relation R_i. The* S*-satisfiability problem (denoted by* $\mathsf{SAT(S)}$*) is the problem of deciding whether a given* S*-formula is satisfiable.*

The problem $\mathsf{SAT_c(S)}$ *is the variant of the problem* $\mathsf{SAT(S)}$ *in which the constants* 0 *and* 1 *are also allowed to occur in* S*-formulas.*

A logical relation R is **weakly positive** *if $R(x_1, x_2, \ldots)$ is logically equivalent to some* CNF *formula having at most one negated variable in each conjunct. A logical relation R is* **weakly negative**[6] *if $R(x_1, x_2, \ldots)$ is logically equivalent to some* CNF *formula having at most one unnegated variable in each conjunct. A logical relation R is* **affine** *if $R(x_1, x_2, \ldots)$ is logically equivalent to some system of linear equations over the two-element field $\mathbb{F}_2 = \{0,1\}$.*

Theorem 4.4 *[Sc78] Let* S *be a finite set of finite arity Boolean relations such that one of the following condition holds: (i) Every relation in* S *is weakly positive (ii) Every relation in* S *is weakly negative (iii) Every relation in* S *is affine. Then* SAT(S) *can be solved in polynomial time.* ■

Theorem 4.5
1. *The* PRE *problem for SDSs in which each local transition function is OR or NOR is efficiently reducible to the* SAT(S) *problem where every relation in* S *is weakly positive.*
2. *The* PRE *problem for SDSs in which each local transition function is AND or NAND is efficiently reducible to the* SAT(S) *problem where every relation in* S *is weakly negative.*
3. *The* PRE *problem for SDSs in which each local transition function is XOR or XNOR is efficiently reducible to the* SAT(S) *problem where every relation in* S *is affine.*

Thus, the PRE *problem for the above restricted classes of SDSs can be solved efficiently. Moreover, these results hold even for SyDSs.*

Proof sketch: Since the proofs for the three parts are very similar, we will only present the proof for for Part 1 of the theorem. (A proof for Part 3 is implicit in the proof of Theorem 4.3.)

Consider a given node v_i and its associated variable x_i. Suppose the local transition function f_i associated with v_i is OR. (The proof for NOR is similar.) If the bit b_i corresponding to v_i in the final configuration C is 1, then the term corresponding to v_i is the OR of a collection of variables; the resulting term contains no negated literals. If the bit b_i corresponding to v_i in the final configuration C is 0, then the term for v_i is the NOR of a collection of variables. By DeMorgan's law, the NOR of a collection of variables is equivalent to the conjunction of their negations; in this case, each conjunct contains exactly one negated literal. Thus, by definition, each resulting term corresponds to a weakly positive Boolean relation. The reduction for SyDSs can be done in an analogous manner. ■

[6]Such clauses are also referred to as HORN clauses.

Recall that given an integer $t \geq 1$ and the t-PREDECESSOR EXISTENCE (t-PRE) problem is to determine whether there is a configuration C' from which C can be reached in exactly t transitions. Our polynomial results for the PRE problem for SDSs with restricted local transition functions (Theorems 4.3 and 4.5) also extend to the t-PRE problem, when t is bounded by a polynomial in the size of the given SDS. This can be seen through a straightforward modification of the reduction to SAT (or to a set of linear equations). For each node of the SDS, we have t Boolean variables, one corresponding to each (backward) time step. The term for node v_i and time j is constructed from the variables of adjacent nodes corresponding to time j or $j+1$ depending on whether the node follows or precedes v_i in the permutation.

We illustrate the construction for SDSs with $t = 2$; the generalization is straightforward. Let x_i^1 and x_i^2 denote the two variables corresponding to node v_i. (The interpretation is that x_i^j represents the value of the state of node v_i at the j^{th} preceding time step, $j = 1, 2$.) The construction produces two terms corresponding to node v_i. To describe the construction, let $N(i)$ denote the set of neighbors of v_i. In $N(i)$, let v_{i_1}, v_{i_2}, $\ldots$, v_{i_r} denote the nodes that *precede* v_i in the permutation and let v_{j_1}, v_{j_2}, $\ldots$, v_{j_p} denote the nodes that *follow* v_i in the permutation. Let $F_i^1 = f_i(x_i^1, b_{i_1}, b_{i_2}, \ldots, b_{i_r}, x_{j_1}^1, x_{j_2}^1, \ldots, x_{j_p}^1)$. If $b_i = 1$, the first term corresponding to v_i is F_i^1 itself; if $b_i = 0$, the first term corresponding to v_i is $\overline{F_i^1}$. Let F_i^2 represent the expression $\left(x_i^1 \equiv f_i(x_i^2, x_{i_1}^1, x_{i_2}^1, \ldots, x_{i_r}^1, x_{j_1}^2, x_{j_2}^2, \ldots, x_{j_p}^2)\right)$, where $\equiv$ denotes logical equivalence. This logical equivalence can be easily converted into a collection of subformulas connected by conjunctions. The resulting collection of subformulas is the second term corresponding to node v_i.

In general, when t is bounded by a polynomial in the size of S, the number of terms in the resulting SAT instance is also bounded by a polynomial. Thus, the polynomial time results extend to this case as stated in the following theorem.

Theorem 4.6 *When t is polynomial in $|S|$, the t-PRE problem can be solved efficiently for any of the following classes of SDSs or SyDSs: (i) SDSs (or SyDSs) in which each local transition function is from {OR, NOR} (ii) SDSs (or SyDSs) in which each local transition function is from {AND, NAND} (iii) SDSs (or SyDSs) in which each local transition function is from {XOR, XNOR}.* ■

Finally, we note that the above results can be extended to obtain polynomial time algorithms for the t-SUB-PRE, t-TEMP-SEQ-PRE and t-SUB-RECUR problems. This follows by noting that in our reduction only the subconfiguration values are prescribed in advance; the state values of the remaining nodes can be anything so far as the subconfiguration values are preserved. Thus, we get:

Theorem 4.7 *When t is polynomial in $|S|$, the t-SUB-PRE, the t-SUB-RECUR and the t-TEMP-SEQ-PRE problems can be solved efficiently for any of the following classes of SDSs or SyDSs: (i) SDSs (or SyDSs) in which each local transition function is from {OR, NOR} (ii) SDSs (or SyDSs) in which each local transition function is from {AND, NAND} (iii) SDSs (or SyDSs) in which each local transition function is from {XOR, XNOR}.* ■

4.3 Treewidth Bounded Graphs

In this section, we give polynomial algorithms for the predecessor existence problems when the underlying graph of the SDS is of bounded treewidth. To discuss this result, we need a definition and a result from the literature.

Definition 4.2 *[HM+94] Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of Boolean variables and let $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$ be a collection of terms, where each term is a Boolean function of a subset of the variables in X. The* **interaction graph** *for $\mathcal{T}$ has one node for each variable in X; there is an edge between two nodes if the two corresponding variables appear together in some term of $\mathcal{T}$.*

Theorem 4.8 *[HM+94] Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of Boolean variables and let $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$ be a collection of terms, where each term is a Boolean function of a subset of the variables in X. If the interaction graph of $\mathcal{T}$ has bounded treewidth, then the SAT problem for $\mathcal{T}$ can be solved in polynomial time.*
■

Theorem 4.9 *Let S be an SDS where each local transition function is Boolean and the underlying graph has bounded treewidth. The* PRE *problem for S can be solved in polynomial time.*

Proof: Given SDS S and the required final configuration $\mathcal{C}$, we construct the set of terms as described earlier. The key observation about this construction is the following. The interaction graph of the resulting set of terms is a subgraph of the underlying graph of the SDS. Since the latter has bounded treewidth by assumption, the former has bounded treewidth as well. By Theorem 4.8, the SAT problem for the resulting set of terms can be solved in polynomial time. Thus, the PRE problem for SDS S can also be solved in polynomial time.
■

We now briefly mention a number of extensions of the above result.

1. We first note that the result of Theorem 4.9 also extends to SyDSs (finite CA). This follows by a straightforward modification of the construction of the set of linear equations or the set of terms described above. The result for finite CA was previously obtained by Sutner [Su95] who also showed that the PRE problem for finite 1-dimensional CA with a fixed neighborhood radius r can be solved in polynomial time. The latter result also follows from the extension of Theorem 4.9 to SyDSs since 1-dimensional CA with neighborhood radius r give rise to graphs of treewidth at most $2r + 1$.

2. A second extension concerns the counting problem associated with the PRE problem. Here, the goal is to determine the number of predecessors of a given configuration. When the interaction graph of a set of Boolean terms is treewidth bounded, it is known that the number of satisfying assignments for the terms can also be found in polynomial time [SH96]. It follows that for SDSs whose underlying graphs are treewidth bounded, the number of predecessors of a given configuration can be determined in polynomial time. A simple consequence is that for such SDSs, the problem of determining whether *a configuration has a unique predecessor can also be solved in polynomial time.*

3. The t-PRE problem for SDSs whose underlying graphs are treewidth bounded can be solved in polynomial time when $t = O(\log n)$, where n is the number of nodes. This result holds with no restrictions on the local transition functions except that they can be computed efficiently. To see this, note that when we reduce the t-PRE problem for a treewidth bounded SDS to SAT, the treewidth of the resulting interaction graph is $O(t \cdot w)$, where w is the treewidth of the underlying graph of the SDS. The algorithm for SAT when the interaction graph has treewidth q runs in time $n|D|^{O(q)}$ [HM+94], where n is the number of variables and $|D|$ is the size of the domain of state values. Since $q = O(t \cdot w)$ and w is fixed, we obtain polynomial algorithms for the t-PRE problem for treewidth bounded graphs under the following two scenarios: (i) when $t = O(\log n)$ and $|D|$ is a constant or (ii) when t is a constant and $|D|$ is bounded by a polynomial in $|S|$.

4. Finally note that combining the ideas used to prove Theorem 4.7 and the above ideas for treewidth bounded graphs, we can obtain polynomial time algorithms for the t-SUB-PRE, t-SUB-RECUR and t-TEMP-SEQ-PRE problems when either (i) $t = O(\log n)$ and $|D|$ is a constant or (ii) when t is a constant and $|D|$ is bounded by a polynomial in $|S|$.

5 The PERMUTATION EXISTENCE Problem

5.1 NP-Completeness Results

Theorem 5.1 *The* PME *problem is* **NP***-complete for SDSs in which each local transition function is a simple-threshold function.*

Proof: The membership of PME in **NP** is obvious. So, we focus on proving the **NP**-hardness through a reduction from 3SAT.

Let $X = \{x_1, x_2, \ldots, x_n\}$ denote the variables and $C = \{c_1, c_2, \ldots, c_m\}$ denote the clauses in the given instance of 3SAT. For each variable x_i, the underlying graph of the SDS has five nodes, denoted by x_i^r, $1 \le r \le 5$. For each clause c_j, the underlying graph of the SDS has one node, denoted by c_j. The edges in the graph are as follows.

1. For each i, $1 \le i \le n$, add the four edges $\{x_i^1, x_i^5\}$, $\{x_i^2, x_i^5\}$, $\{x_i^3, x_i^5\}$ and $\{x_i^4, x_i^5\}$.
2. For each i and j, ($1 \le i \le n$ and $1 \le j \le m$), if variable x_i appears positively in clause c_j, then add the two edges $\{x_i^1, c_j\}$ and $\{x_i^2, c_j\}$; if variable x_i appears negatively in clause c_j, then add the two edges $\{x_i^3, c_j\}$ and $\{x_i^4, c_j\}$.

The threshold functions associated with nodes are as follows. For each i ($1 \le i \le n$), the local transition function associated with nodes x_i^1 and x_i^3 is the q-simple-threshold function, where q is one more than the degree of the node; in other words, for every input, the value of this threshold function is zero. The local transition function associated with nodes x_i^2 and x_i^4 is the 0-simple-threshold function; in other words, for every input, the value of this threshold function is 1. The node x_i^5 has five inputs; the the local transition function associated with nodes x_i^5 is the 3-simple-threshold function. Each clause node c_j ($1 \le j \le m$) has seven inputs; the local transition function associated with nodes c_j is the 4-simple-threshold function.

The initial configuration C' sets each of the nodes x_i^1, x_i^3, x_i^5 ($1 \le i \le n$) to 1 and the other nodes to 0. The final configuration C requires each of the nodes x_i^1, x_i^3, x_i^5 ($1 \le i \le n$) to have the value 0 and the other nodes to have the value 1.

For each i, the point of x_i^5 is that, in order to change it to 0, x_i^1 must be made 0 before x_i^4 is made 1 or x_i^3 must be made 0 before x_i^2 is made 1. Stated differently, if x_i^1 and x_i^2 are ever simultaneously 1, x_i^3 and x_i^4 are never simultaneously 1, and vice versa. The first case is associated with making the variable x_i true and the second with making x_i false.

Suppose a satisfying assignment sets x_i to true. Then updates can be made in the following order: change x_i^3 to 0, x_i^5 to 0, x_i^2 to 1, change clause variables, change x_i^4 to 1 and change x_i^1 to 0. Because x_i^1 and x_i^2 are both 1 when clauses are changed, a contribution of 2 is made by x_i toward the threshold of 4 for the clause. With contributions of at least 1 from the other literals, the clause variable is changed to 1. A similar update order can be obtained when the satisfying assignment sets x_i to false.

Using the above observations, it can be verified that the resulting PME instance has a solution if and only if the 3SAT instance has a solution. ■

Recall that in the generalized version of the PME (GEN-PME) problem, the final configuration may contain don't care values. Our next theorem shows that the GEN-PME problem is **NP**-complete even for simple SDSs.

Theorem 5.2 *The* GENERALIZED PERMUTATION EXISTENCE *problem is* **NP***-complete for each of the following classes of SDSs.*

1. *SDSs in which every local transition function is NOR and whose underlying graphs have a maximum node degree of 3.*
2. *SDSs in which every local transition function is NAND and whose underlying graphs have a maximum node degree of 3.*
3. *SDSs whose underlying graphs are planar.*

Proof: The GEN-PME problem is obviously in **NP**. We establish **NP**-hardness through reductions from restricted versions of SAT.

Part 1: The reduction is from 3SAT-2OCCUR. The underlying graph of the SDS has one node for each literal and one node for each clause. There is an edge between each clause and the three literals in that clause. There is also an edge between the two literals corresponding to a variable. Since each literal occurs in at most two clauses and each clause has only three literals, the degree of each node in the resulting graph is at most 3.

For each clause node, the initial and final state values are 0. For each literal node, the initial state value is 0 and the final state value is don't care.

We now argue that the GEN-PME problem has a solution if and only if the given instance of 3SAT-2OCCUR has a solution. Suppose there is a satisfying assignment. We choose the final value 1 for all true literals and the final value 0 for all false literals. The permutation is obtained by first having all the true literals (in an arbitrary order) followed by the other nodes (also in an arbitrary order). It can be verified that the nodes reach the specified final values. For the converse, suppose there is a permutation that makes each clause node have the final value 0. Since there is an edge between the two literals corresponding to a variable, and each node function is NOR, at most one of the two literals can have a final value of 1. Since each clause node goes from 0 to 0, at the time the clause node is evaluated, at least one of the literals in the clause must have the value 1. In other words, each clause can be satisfied.

Part 2: The proof is along the same lines as that of Part 1 using a dual argument.

Part 3: We use a reduction from the PL-PE3SAT problem. Given an instance of PL-PE3SAT with variable set X and clause set C, we create the following partial SDS $\mathcal{S}$. For each variable $x_i \in X$, $\mathcal{S}$ has two nodes (denoted by x_i and x_i'), $1 \leq i \leq n$. For each clause $c_j \in C$, $\mathcal{S}$ has one node (denoted by c_j), $1 \leq j \leq m$. There is an edge between x_i and x_i' for each i, $1 \leq i \leq n$. Further, If the clause c_j contains positive literals x_{j_1}, x_{j_2} and x_{j_3}, then there is an edge between x_{j_r} and c_j for $r = 1, 2, 3$. This completes the specification for the undirected graph of $\mathcal{S}$. Note that underlying graph of the resulting SDS is planar since it is obtained from the (planar) bipartite graph corresponding to the PL-PE3SAT instance by simply attaching a node x_i' of degree 1 to each variable node x_i $(1 \leq i \leq n)$.

The Boolean functions associated with each node are as follows. For each node x_i and x_i' ($1 \le i \le n$), the associated Boolean function is the NOR function. For the node c_j ($1 \le j \le m$), the associated Boolean function takes on the value 1 if exactly one of the inputs is 1; otherwise, the function value is 0.

The initial configuration C' assigns the value 0 to each node. The final configuration C requires each node c_j ($1 \le j \le m$) to have the value 1; the values for every other node is "don't care".

This completes the specification of of the partial SDS S. We now show that there is a permutation π such that S can reach one of the final configurations specified by C in one transition if and only if the PL-PE3SAT instance is satisfiable.

Suppose there is a satisfying truth assignment to the PL-PE3SAT instance. We construct the following permutation π for S. The first $2n$ entries of π specify the order for the nodes x_i and x_i' ($1 \le i \le n$). If the satisfying truth assignment sets x_i to 1, then π puts x_i ahead of x_i'; otherwise, π puts x_i' ahead of x_i. The last m entries of π are $c_1, c_2, \ldots, c_m$. It can be verified that when the SDS executes one step in the order specified by π, the final value of x_i is identical to the value given by the satisfying assignment. As a consequence, when the nodes corresponding to the clauses execute, each of them has a final value of 1. This is in the set of allowable final configurations specified by C. Thus, π is a valid solution to the constructed generalized PME instance.

Now, suppose there is a permutation π for S such that S, starting from C', reaches one of the final configurations specified by C in one step. We carry out this step of S using π and claim that the final values assigned to the $x_1, x_2, \ldots, x_n$ give an exactly 1-in-3 satisfying assignment to the PL-PE3SAT instance. To see this, assume that some clause c_j is not satisfied in an exactly 1-in-3 fashion. Let x_{j_1}, x_{j_2} and x_{j_3} denote the three variables appearing in c_j. Since the final value of node c_j is 1, when node c_j was executed, exactly one of x_{j_1}, x_{j_2} and x_{j_3} was 1. Without loss of generality, suppose x_{j_1} was 1 and the other two nodes were 0 when c_j executed. Now, in the final state if x_{j_2} is also 1, then x_{j_2} must have executed after c_j. However, node c_j with value 1 is one of the inputs to x_{j_2}. So, if x_{j_2} executes after c_j, then x_{j_2} would be set to 0, contradicting our assumption that the final value of x_{j_2}. Therefore, each clause of the PL-PE3SAT is satisfied in an exactly 1-in-3 fashion. This completes the proof of the theorem. ■

5.2 Polynomial Algorithms

Theorem 5.3 *The generalized* PME *problem can be solved in polynomial time for OR-SDSs and AND-SDSs.*

Proof sketch: We present the proof for OR-SDSs. A proof for AND-SDSs can be obtained by a dual argument.

Consider an OR-SDS. If there is any node that goes from 1 in the initial configuration to 0 in the final configuration, stop and output NO. Similarly, if there is any node that goes from 0 to 0 and has a neighbor whose initial value is 1, stop and output NO. Change the final state to 1 for every node that goes from 1 to don't care. Define a node to be a *stable-0* if its initial and final values are both 0. Define a node to be a *stable-1* if its initial and final values are both 1. Define a given node to be a *viable-1* node if its initial value is 0, its final value is either 1 or don't care, and there exists a path of zero or more viable-1 nodes connecting the given node to a stable-1 node. (Computationally, this can be computed by starting from the stable-1 nodes, and identifying viable-1 nodes as a node whose initial value is 0, final value is either 1 or don't care, and is adjacent to either a stable-1 node or a node that has been determined to be a viable-1 node.)

If there is any node whose initial value is 0, final value is 1, and is not a viable-1 node, then stop and output NO, otherwise, output YES.

If the algorithm outputs "yes", the permutation can be determined as follows. First, the don't cares are resolved. All nodes whose initial value is 0, final value is don't care, and are a viable-1, have their final value changed to 1. All nodes whose initial value is 0, final value is don't care, and are not a viable-1, have their final value changed to 0.

The permutation first has all the nodes with final value 0. Then it has all the stable-1 nodes. Then it has the viable-1 nodes, in the order of their minimum distance from the set of stable-1 nodes, where distance is determined using the subgraph of nodes whose final value is 1. ■

The following result shows that for NOR-SDSs and NAND-SDSs, the PME problem (without don't care values) can be solved in polynomial time. This result brings out the contrast in the complexity of GEN-PME and PME problems for NOR-SDSs and NAND-SDSs.

Theorem 5.4 *The* PME *problem for NOR-SDSs and NAND-SDSs can be solved in linear time.*

Proof: We will give the proof for NOR-SDSs. The proof for NAND-SDSs is similar.

If there is any node that goes from 1 to 1, stop and output "no". If there are two adjacent nodes that both go from 0 to 1, stop and output "no". (Whichever node goes first in the permutation, the other node must have the final value of 0.) If there is any node v that go from 0 to 0, and all of the neighbors of v also go from 0 to 0, then stop and output "no".

Otherwise, stop and output "yes". The permutation can be constructed as follows. (The permutation is described by specifying disjoint groups of nodes. Within each group, nodes can be ordered arbitrarily.)

The first group of nodes in the permutation are those that go from 0 to 0, and have at least one neighbor that goes from 1 to 0. (The initial value of the neighbor justifies the transitions.) The next group of nodes are those that go from 1 to 0. The next group of nodes are those that go from 0 to 1. (At this point in the algorithm, the neighbors of these nodes all have value 0.) The final group consists of nodes that go from 0 to 0, and which have not yet been placed in the permutation. (Each such node has a neighbor that went from 0 to 1, justifying its transition.) ■

6 Concluding Remarks

We considered the PREDECESSOR EXISTENCE and PERMUTATION EXISTENCE problems in the context of SDSs. We showed that these problems are **NP**-complete even for several simple classes of SDSs. We also identified several restricted classes of SDSs for which the two problems can be solved efficiently. We close by mentioning some open problems resulting from our work.

It would be of interest to investigate whether the complexities of PRE and PME problems can be characterized in a manner similar to Schaefer's characterization of generalized SAT [Sc78]. It is also of interest to identify other useful classes of SDSs for which the PRE or the PME problems and their generalizations can be solved efficiently. In Sections 4.2 and 4.3, we discussed the t-PRE problem where the goal is to find an ancestor configuration that precedes the given configuration by exactly t steps. Our polynomial results hold when t is given in unary. When t is given in binary, the complexity of the t-PRE problem is open.

The close relationship between PRE problem and the generalized SAT problem allowed us to obtain polynomial algorithms for the problem for several restricted classes of SDSs. It would be useful to identify a suitable problem that is closely related to the PME problem. Similar to t-PRE, one can also formulate the t-PME problem in conjunction with the PME problem. Results on t-PME for different representations of t will provide additional insights into the foundations of SDSs.

Acknowledgements: This research was also funded by the LDRD-DR project *Foundations of Simulation Science* and the LDRD-ER project *Extremal Optimization* at the Los Alamos National Laboratory. We thank Henning Mortveit, Christian Reidys, Predrag Tosic and Paul Wollan (all from Los Alamos National Laboratory) for several discussions on topics related to this paper.

References

[ALS91] Arnborg, S., Lagergren, J., and Seese, D. (1991), Easy Problems for Tree-Decomposable Graphs, *J. Algorithms*, **12**, pp. 308–340

[ARV94] S. Arora, Y. Rabani and U. Vazirani. Simulating quadratic dynamical systems is **PSPACE**-complete," *Proc 26th. Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 459-467, Montreal, Canada, May 1994.

[BB+99] C. Barrett, B. Bush, S. Kopp, H. Mortveit and C. Reidys. Sequential Dynamical Systems and Applications to Simulations. Technical Report, Los Alamos National Laboratory, Sept. 1999.

[BHM+00] C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz and R. E. Stearns. Dichotomy Results for Sequential Dynamical Systems. Submitted for publication, Dec. 2000.

[BR99] C. Barrett and C. Reidys. Elements of a theory of computer Simulation I: sequential CA over random graphs. *Applied Mathematics and Computation*, 98, pp. 241–259, 1999.

[BMR99] C. Barrett, H. Mortveit, and C. Reidys. Elements of a theory of simulation II: sequential dynamical systems. *Applied Mathematics and Computation*, 1999, vol 107/2-3, pp. 121-136.

[BMR00] C. Barrett, H. Mortveit and C. Reidys. Elements of a theory of computer simulation III: equivalence of SDS. to appear in *Applied Mathematics and Computation*, 2000.

[BH+00a] C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz and R. E. Stearns. Elements of a theory of computer simulation V: computational complexity and universality. to be submitted, January 2001.

[BH+00b] C. Barrett, H. Hunt III, M. Marathe, S. Ravi, D. Rosenkrantz. Computational aspects of sequential dynamical systems II: design problems. in preparation, 2000.

[BHR84] P.A. Bloniarz, H.B. Hunt III and D.J. Rosenkrantz. Algebraic Structures with Hard Equivalences and Minimization Problems. *Journal of the ACM (JACM)*, 31, pp. 879-904, 1984.

[Bo88] Bodlaender, H. L. (1988), NC Algorithms for Graphs with Bounded Treewidth, *in* "Proceedings, Workshop on Graph Theoretic Concepts in Computer Science," pp. 1–10.

[BPT91] S. Buss, C. Papadimitriou and J. Tsitsiklis. On the predictability of coupled automata: An allegory about Chaos. *Complex Systems*, 1(5), pp. 525-539, 1991. Preliminary version appeared in *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, Oct 1990.

[CPY89] K. Cullik, J. Pachl and S. Yu. On the limit sets of cellular automata. *SIAM J. Computing*, 18(4), pp. 831-842, 1989.

[CY88] K. Cullik and S. Yu. Undecidability of CA classification schemes. *Complex Systems*, 2(2), pp. 177-190, 1988.

[DF86] M. E. Dyer and A. M. Frieze. Planar 3DM is NP-Complete. *J. Algorithms*, Vol. 7, No. 2, March 1986, pp. 174–184.

[Du94] B. Durand. Inversion of 2D cellular automata: some complexity results. *Theoretical Computer Science*, 134(2), pp. 387-401, November 1994.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman and Co., San Francisco, CA, 1979.

[Ga97] P. Gacs. Deterministic computations whose history is independent of the order of asynchronous updating. Tech. Report, Computer Science Dept, Boston University, 1997.

[Gr87] F. Green. NP-Complete Problems in Cellular Automata. *Complex Systems*, Vol. 1, No. 3, 1987, pp. 453–474.

[Gu89] H. Gutowitz (Editor). *Cellular Automata: Theory and Experiment* North Holland, 1989.

[GSW94] R. Gunther, B. Schapiro and P. Wagner. Complex Systems, Complexity Measures, grammars and model-Inferring. *Chaos Solitons and Fractals* 4(5), pp. 635-651, 1994

[HG99] B. Huberman and N. Glance. Evolutionary games and computer simulations. *Proc. National Academy of Sciences*, 1999.

[HM+94] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz and R. E. Stearns. Designing Approximation Schemes using L-reductions. *Proc. 14th Conference on Foundations of Software Technology and Theoretical Computer Science* (FST&TCS'94), Madras, India, Dec. 1994, pp. 342–353. (Complete version to appear in *Information and Computation*, 2001.)

[Hu87] L.P. Hurd, On Invertible cellular automata. *Complex Systems,* 1(1), pp. 69-80, 1987.

[IMR00] G. Istrate, M. Marathe and S. Ravi, Adversarial models in evolutionary game dynamics. To appear in *Proc. of ACM Symposium on Discrete Algorithms* January 2001.

[Ko70] Z. Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill Book Co., New York, NY, 1970.

[LP00] R. Laubenbacher and B. Pareigis. Finite Dynamical Systems. Technical report, Department of Mathematical Sciences, New Mexico State University, Las Cruces, 2000.

[Ma98] B. Martin. A Geometrical Hierarchy of Graphs via Cellular Automata. Proc. MFCS'98 Satellite Workshop on Cellular Automata, Brno, Czech Republic, Aug. 1998.

[MHRS98] M.V. Marathe, H.B. Hunt III, D.J. Rosenkrantz and R.E. Stearns. Theory of periodically specified problems: Complexity and Approximability. *Proc. 13th IEEE Conference on Computational Complexity*, Buffalo, NY, June, 1998.

[Mo91] C. Moore. Generalized shifts: unpredictability and undecidability in Dynamical Systems. *Nonlinearity*, 4, pp. 199-230, 1991.

[Mo90] C. Moore. Unpredictability and undecidability in dynamical Systems. *Physical Review Letters*, 64(20), pp 2354-2357, 1990.

[MR99] H. Mortveit, and C. Reidys. Discrete sequential dynamical systems. *Discrete Mathematics,* 2000 accepted.

[NR98] C. Nichitiu and E. Remila. Simulations of Graph Automata. Proc. MFCS'98 Satellite Workshop on Cellular Automata, Brno, Czech Republic, Aug. 1998.

[Pa94] C. Papadimitriou. *Computational Complexity,* Addison-Wesley, Reading, Massachusetts, 1994.

[Rk94] Z. Roka. One-way cellular automata on Cayley graphs. *Theoretical Computer Science,* 132(1-2), pp. 259-290, September 1994.

[RSW92] Y. Rabinovich, A. Sinclair and A. Wigderson. Quadratic dynamical systems. *Proc. 33rd Annual Symposium on Foundations of Computer Science (FOCS),* pp. 304-313, Pittsburgh, October 1992.

[Re00] C. Reidys. On acyclic orientations and SDS. Advances in Applied Mathematics, to appear in 2000.

[Re00a] C. Reidys. Sequential dynamical systems: phase space properties. Advances in Applied Mathematics, to appear.

[RS86] Robertson, N., and Seymour, P. D. (1986), Graph Minors II, Algorithmic Aspects of Tree-Width, *Journal of Algorithms* 7, pp. 309–322.

[Ro99] C. Robinson. *Dynamical systems: stability, symbolic dynamics and chaos.* CRC Press, New York, 1999.

[Sc78] T. Schaefer. The Complexity of Satisfiability Problems. *Proc. 10th ACM Symposium on Theory of Computing* (STOC'78), 1978, pp. 216–226.

[SH96] R. E. Stearns and H. B. Hunt III. An Algebraic Model for Combinatorial Problems. *SIAM Journal on Computing*, Vol. 25, No. 2, April 1996, pp. 448–476.

[Sm71] A. Smith. Simple computation-universal cellular spaces. *J. ACM,* 18(3), pp. 339-353, 1971.

[Su95] K. Sutner. On the computational complexity of finite cellular automata. *Journal of Computer and System Sciences,* 50(1), pp. 87-97, February 1995.

[Su90] K. Sutner. De Bruijn graphs and linear cellular automata. *Complex Systems,* 5(1), pp. 19-30, 1990.

[Su89] K. Sutner. Classifying circular cellular automata. *Physica D,* 45(1-3), pp. 386-395, 1989.

[SDB97] C. Schittenkopf, G. Deco and W. Brauer. Finite automata-models for the investigation of dynamical systems. *Information Processing Letters,* 63(3), pp. 137-141, August 1997.

[Von93] J. von zur Gathen. Parallel Linear Algebra. Chapter 13 in *Synthesis of Parallel Algorithms,* Edited by J. H. Reif, Morgan Kaufmann Publishers, San Mateo, CA, 1993, pp. 573–617.

[Wo86] S. Wolfram, Ed. *Theory and applications of cellular automata.* World Scientific, 1987.