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]

January 2, 2002

Abstract

A class of finite discrete dynamical systems, called **Sequential Dynamical Systems** (SDSs), was introduced in [BMR99, BR99] as a formal model for analyzing simulation systems. 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 state, (ii) $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$, with f_i denoting a function associated with node $v_i \in V$ and (iii) π is a permutation of (or total order on) the nodes in V. A configuration of an SDS is an n-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 basic problems and their generalizations 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 $\mathcal{S}$ has a transition from $\mathcal{C}'$ to $\mathcal{C}$. (If $\mathcal{C}$ has no predecessor, $\mathcal{C}$ is known as a **garden of Eden** configuration.) Our results provide separations between efficiently solvable and computationally intractable instances of the PRE problem. For example, we show that the PRE problem can be solved efficiently for SDSs with Boolean state values when the node functions are symmetric and the underlying graph is of bounded treewidth. In contrast, we show that allowing just one non-symmetric node function renders the problem **NP**-complete even when the underlying graph is a tree (which has a treewidth of 1). We also show that the PRE problem is efficiently solvable for SDSs whose state values are from a field and whose node functions are linear. Some of the polynomial algorithms also extend to the case where we want to find an ancestor configuration that precedes a given configuration by a logarithmic number of steps. 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.

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 such that $\mathcal{S}$ has a transition 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.

Submission to Track 1: Algorithms, Complexity and Models of Computation.

[1]basic and Applied Simulation Sciences, (-2), 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 (D-2) at the Los Alamos National Laboratory.

Title: Predecessor and Permutation Existence Problems for Sequential Dynamical Systems

Track: 1 (Algorithms, Complexity and Models of Computation)

Authors: Chris Barrett, Harry B. Hunt III, Madhav V. Marathe, S. S. Ravi, Daniel J. Rosenkrantz and Richard E. Stearns

Contact author: S. S. Ravi
Department of Computer Science, LI 67A
University at Albany - State University of New York
1400 Washington Avenue
Albany, NY 12222, USA

Email: `ravi@cs.albany.edu`
Phone: (518) 442-4278
Fax: (518) 442-5638

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 SDSs), proposed in [BR99, BMR99]. 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]. 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 state. $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$, with f_i denoting a 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-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 function. These updates are carried out in the order specified by π.

The research reported here is a part of a program[4] to provide a formal basis for the design and analysis of *large-scale computer simulations*, especially for socio-technical systems [BE+01]. Examples of such systems include various national infrastructures including transportation, power and communication. 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 can be seen to capture these features. For additional information regarding how SDSs can be used to model simulations, we refer the reader to [BB+99, BE+01].

Simulation of Mobile/Adhoc Communication Networks: One motivation to study SDS is derived from their applicability in specifying large scale simulation of mobile networks [BM+00] coined **AdhopNET**. For simplicity of description, the basic framework consists of three ingredients. Transceivers are the agents of the simulation and correspond to the nodes in the interaction graph. There is a directed edge from transceiver A to transceiver B if and only if transceiver B is within the effective broadcast range of transceiver A (i.e., B can receive signals from A within a certain acceptable level of signal-to-noise ratio). The local function at each node corresponds to the parametric representation of the protocols that we use to represent each agent. A number of additional features can be added to this basic model but we omit them for reasons of brevity and clarity. In this setting, we can show that two different hand-off protocols can be simulated by *essentially* changing the order of updates. In the base station initiated hand-off, the base station continually monitors the strength of the received signal and makes a decision to initiate a hand-off. Here, we first update the mobile units and then the base stations. In the *receiver* oriented hand-off protocols, a receiver initiates a hand-off by monitoring the strength of the beacons of the neighboring base stations. In this case the mobile units are updated after the base stations.[5] Again, the important point to note is that sequential processing of nodes is more realistic and useful for modeling real protocols as opposed to synchronous updates.

In [BH+01b, BH+01c], we studied the complexity of some configuration reachability problems for SDSs. Here, we focus on the complexity of two basic problems for SDSs, namely PREDECESSOR EXISTENCE and

[4] See `http://tsasa.lanl.gov` for additional details on this program.

[5] In the real system, there is a sequence of alternating updates but we omit this complication here.

PERMUTATION EXISTENCE. We now discuss these problems informally and defer the formal definitions to Section 2.3. Given an SDS $\mathcal{S} = (G, \mathcal{F}, \pi)$ and a configuration $\mathcal{C}$, the PREDECESSOR EXISTENCE problem (abbreviated as PRE) is to determine whether the configuration $\mathcal{C}$ has a predecessor; that is, whether there is a configuration $\mathcal{C}'$ such that $\mathcal{S}$ has a transition from $\mathcal{C}'$ to $\mathcal{C}$ in one step. Given a partially specified SDS $\mathcal{S}$ consisting of the underlying graph $G(V, E)$, the set $\mathcal{F}$ of functions associated with the nodes and two configurations $\mathcal{C}$ and $\mathcal{C}'$, the PERMUTATION EXISTENCE problem (abbreviated as PME) is to determine whether there is a permutation π of the nodes such that under permutation π, there is a transition from $\mathcal{C}'$ to $\mathcal{C}$ in one step. 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. Predecessor existence question is directly related to certain liveness properties of certain network protocols [GC86, Pe97]. Our conclusion is that these questions are, in general, computationally intractable. However, we identify a number of special classes of SDSs for which the questions can be answered efficiently. Several of our results are also applicable to cellular automata (CA) and graph automata (GA).

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

2 Definitions and Problem Formulations

2.1 Sequential Dynamical Systems

A **Sequential Dynamical System** (SDS) $\mathcal{S}$ over a given domain $\mathbb{D}$ of state values is a triple $(G, \mathcal{F}, \pi)$, whose components are as follows:

1. $G(V, E)$ is a finite 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, ..., n.

2. For each node i of G, $\mathcal{F}$ specifies a **local transition function**, denoted by f_i. This function maps $\mathbb{D}^{\delta_i+1}$ into $\mathbb{D}$, where δ_i is the degree of node i. Letting $N(i)$ denote the set consisting of node i itself and its neighbors, each parameter of f_i corresponds to a member of $N(i)$. Throughout the paper, we assume that the local transition function associated with each node can be evaluated in polynomial time.

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.

A **configuration** $\mathcal{C}$ of $\mathcal{S}$ can be interchangeably regarded as an n-vector $(c_1, c_2, \ldots, c_n)$, where each $c_i \in \mathbb{D}$, $1 \leq i \leq n$, or as a function $\mathcal{C} : V \rightarrow \mathbb{D}$. From the first perspective, c_i is the state value of node i in configuration $\mathcal{C}$, and from the second perspective, $\mathcal{C}(i)$ is the state value of node i in configuration $\mathcal{C}$.

Computationally, each step of an SDS (i.e., the transition from one configuration to another), involves n substeps, where the nodes are processed in the *sequential* order specified by permutation π. The "processing" of a node consists of computing the value of the node's local transition function and changing its state to the computed value. The following pseudocode shows the computations involved in one transition.

for $i = 1$ **to** n **do**

(i) 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 $\pi(i)$.) Let x denote the value computed.

(ii) Node $\pi(i)$ sets its state $s_{\pi(i)}$ to x.

end-for

We let $F_{\mathcal{S}}$ denote the **global transition function** associated with $\mathcal{S}$. This function can be viewed either as a function that maps $\mathbb{D}^n$ into $\mathbb{D}^n$ or as a function that maps $\mathbb{D}^V$ into $\mathbb{D}^V$. $F_{\mathcal{S}}$ represents the transitions between configurations, and can therefore be considered as defining the dynamic behavior of SDS $\mathcal{S}$.

Let $\mathcal{I}$ denote the designated configuration of $\mathcal{S}$ at time 0. Starting with $\mathcal{I}$, the configuration of $\mathcal{S}$ after t steps (for $t \geq 0$) is denoted by $\xi(\mathcal{S}, \mathcal{I}, t)$. Note that $\xi(\mathcal{S}, \mathcal{I}, 0) = \mathcal{I}$ and $\xi(\mathcal{S}, \mathcal{I}, t+1) = F_{\mathcal{S}}(\xi(\mathcal{S}, \mathcal{I}, t))$. Consequently, for all $t \geq 0$, $\xi(\mathcal{S}, \mathcal{I}, t) = {F_{\mathcal{S}}}^t(\mathcal{I})$.

Recall that a configuration $\mathcal{C}$ can be viewed as a function that maps V into $\mathbb{D}$. As a slight extension of this view, we use $\mathcal{C}(W)$ to denote the states of the nodes in $W \subseteq V$. $\mathcal{C}(W)$ is called a **subconfiguration** of $\mathcal{C}$.

The **phase space** $\mathcal{P}_{\mathcal{S}}$ of an SDS $\mathcal{S}$ is a directed graph defined as follows: There is a node in $\mathcal{P}_{\mathcal{S}}$ for each configuration of $\mathcal{S}$. There is a directed edge from a node representing configuration $\mathcal{C}$ to that representing configuration $\mathcal{C}'$ if $F_{\mathcal{S}}(\mathcal{C}) = \mathcal{C}'$. In such a case, we also say that configuration $\mathcal{C}$ is a **predecessor** of configuration $\mathcal{C}'$. Since SDSs are deterministic, each node in its phase space has an outdegree of 1. In general, the phase space $\mathcal{P}_{\mathcal{S}}$ may have an infinite number of nodes. When the domain $\mathbb{D}$ of state values is finite, the number of nodes in the phase space is $|\mathbb{D}|^n$.

A node in phase space may have multiple predecessors. This means that the time evolution map of an SDS is, in general, *not invertible* but is *contractive*. The existence of configurations with multiple predecessors also implies that certain configurations may have no predecessors. A configuration with no predecessors is referred to as a **garden of Eden** configuration. Such configurations can occur only as initial states and can never be generated during the time evolution of an SDS.

2.2 Variations of the Basic SDS Model

The above definition of an SDS imposes no restrictions on either the domain $\mathbb{D}$ of state values or the local transition functions, except that the ranges of the local transition functions must be subsets of $\mathbb{D}$. SDSs that model simulation systems can be obtained by appropriately restricting $\mathbb{D}$ and/or the local transition functions. We use the notation "(x,y)-SDS" to denote an SDS where 'x' specifies the restriction on the domain and 'y' specifies the restriction on the local transition functions. Some restrictions studied in this paper are discussed below.

References [BMR99, BMR00, MR99, Re00] studied SDSs with Boolean domains and symmetric Boolean local transition functions. (The definition of symmetric Boolean functions is given in Section 2.4.) We denote such an SDS by (BOOL, SYM)-SDS. Symmetric functions provide 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]. As will be seen in Section 4, symmetry plays an important role in separating efficiently solvable and computationally intractable versions of the PME problem for SDSs whose underlying graphs have bounded treewidth.

A restricted class of (BOOL, SYM)-SDSs is obtained by requiring the local transition functions to be *monotone* as well. It can be seen that a Boolean function is symmetric and monotone if and only if it is a k-simple threshold function (defined in Section 2.4) for some integer $k \geq 0$. Therefore, over the Boolean domain, the class of SDSs where each node function is symmetric and monotone coincides with the class

of SDSs where each node function is a k-simple-threshold function for some integer $k \geq 0$. We denote this restricted class by (BOOL, ST)-SDS. A class of symmetric but not monotone Boolean functions is that of k-*tally* functions (defined in Section 2.4) for any integer $k \geq 0$. The class of SDSs in which each node computes the k-tally function for some nonnegative integer k is denoted by (BOOL, TALLY)-SDS. SDSs in which the local transition functions are not necessarily symmetric are also considered in the companion papers [BH+01a, BH+01b, BH+01d].

Some of our results are for SDSs over the Boolean domain where the node functions are from an explicitly specified set of Boolean functions. For example, when each node function is AND, the corresponding class of SDSs is denoted by (BOOL, AND)-SDS. If the set of node functions has cardinality two or more, then standard set notation is used as the restriction on the local transition functions in the notation for such SDSs. For example, when each node function is from the set {XOR, XNOR}, the corresponding class of SDSs is denoted by (BOOL, {XOR, XNOR})-SDS.

We also consider SDSs over an algebraic field where the node functions are linear combinations of the inputs. The class of such SDSs is denoted by (FIELD, LINEAR)-SDS. To specify the node functions more precisely, consider each node v_i of a (FIELD, LINEAR)-SDS, and recall that $N(i) = \{v_i, v_{i_1}, v_{i_2}, \ldots, v_{i_r}\}$ denotes the neighbors of v_i, including v_i itself. Each local transition function f_i, $1 \leq i \leq n$, has the following form:

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

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 '$+$' (addition) and '$*$' (scalar multiplication) are the operators of the 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 underlying field is Boolean with XOR denoting addition and AND denoting scalar multiplication, each linear local transition function is either XOR or XNOR.

It is also of interest to consider dynamical system models obtained by modifying some components of an SDS. One such model is a **Synchronous Dynamical System** (SyDS), which is an SDS *without* the node permutation. In a SyDS, during each time step, all the nodes *synchronously* compute and update their state values. Thus, SyDSs are similar to classical CA with the difference that the connectivity between cells is specified by an arbitrary graph. The restrictions on domain and local transition functions for SDSs are applicable to SyDSs as well.

2.3 Problems Considered

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 two basic 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 $F_{\mathcal{S}}(\mathcal{C}') = \mathcal{C}$. (Note that $\mathcal{C}$ has a predecessor if and only if $\mathcal{C}$ is not a garden of Eden configuration.)

2. Given a partially specified SDS $\mathcal{S}$ consisting of graph $G(V,E)$, the set $\mathcal{F}$ of local transition 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. Recall that the states of nodes in W is the subconfiguration $(b_{i_1}, b_{i_2}, \ldots, b_{i_k})$. Using this notation, we can state the SDS analogs of some problems considered [6] by Green [Gr87] in the context of infinite CA. 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 vector $B = (b_{i_1}, b_{i_2}, \ldots, b_{i_k})$ that specifies state values for the nodes in W and an integer $t \geq 1$, the t-SUBCONFIGURATION 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 vector $B = (b_{i_1}, b_{i_2}, \ldots, b_{i_k})$ that specifies state values for the nodes in W and an integer $t \geq 1$, the t-SUBCONFIGURATION 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 $\mathcal{C}$; in other words, the question is 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 ${F_{\mathcal{S}}}^j(\mathcal{C})(v) = b_v(j),\ 1 \leq j \leq t$.

Another way of thinking about subconfigurations is to assume that a configuration specifies suitable state values for some nodes and "don't-care" values for other nodes. The corresponding subconfiguration is obtained by retaining only the non-don't-care 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 local transition 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 non-don't-care values. (In other words, $\mathcal{C}$ succinctly specifies a set of configurations, and $F_{\mathcal{S}}(\mathcal{C}')$ may be any one of these configurations.)

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

2.4 Other Relevant Definitions

Several special classes of Boolean functions are considered in this paper. We provide formal definitions of these classes below.

Definition 2.1 *A **symmetric** Boolean function is one whose 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.*

[6] 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.

Definition 2.2 *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.*

Definition 2.3 *The k-**tally** (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 both k-simple-threshold and k-tally functions are symmetric. The k-simple-threshold functions are also monotone. They are a special case of threshold functions [Ko70].

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 2.4 *(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 2.5 *[ALS91, Bo88] Let $G(V, E)$ be a graph. A* **tree-decomposition** *of G is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, where $\{X_i \mid i \in I\}$ is a family of subsets of V and $T = (I, F)$ is an undirected 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 its 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] 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 each of the following restricted classes of SDSs: (i) (BOOL, ST)-SDSs where each node computes the same k-simple-threshold function for any $k \geq 2$ (ii) (BOOL, TALLY)-SDSs in which each node computes the same k-tally function for any $k \geq 1$, (iii) (BOOL, {AND, OR})-SDSs and (iv) (BOOL, SYM)-SDSs whose underlying graphs are planar.

We present polynomial time algorithms for the PRE problem for (BOOL, AND)-SDSs and (BOOL, OR)-SDSs with no restrictions on the underlying graph. We also present polynomial time algorithms for the PRE problem for (BOOL, SYM)-SDSs whose underlying graphs have bounded treewidth. These algorithms can be extended to solve the t-PRE, t-SUB-PRE, t-SUB-RECUR and t-TEMP-SEQ-PRE problems in polynomial time when when t is polynomial in $|\mathcal{S}|$. In contrast, we show that even if one nonsymmetric local transition function is permitted, the PRE problem is **NP**-complete for SDSs whose domain is Boolean and whose underlying graphs are trees.

In addition, we present a polynomial time algorithm for the PRE problem for (FIELD, LINEAR)-SDSs. Further, when the underlying graph is both degree and bandwidth bounded, we show that 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 algorithm can also be extended to solve t-PRE, 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 $|\mathcal{S}|$. 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 polynomial time solvability results for the class of SDSs whose underlying graphs are both degree and treewidth bounded, extend Sutner's result since CA can be 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), 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 for (BOOL, ST)-SDSs. We also show that the GEN-PME problem is **NP**-complete for each of the following restricted classes of SDSs: (i) (BOOL, NOR)-SDSs ((BOOL, NAND)-SDSs) where the maximum node degree in the underlying graph is 3 and (ii) (BOOL, SYM)-SDSs whose underlying graphs are planar. We present polynomial time algorithms for the GEN-PME problem for (BOOL, OR)-SDSs and (BOOL, AND)-SDSs. We also present polynomial time algorithms for the PME problem (without don't care values) for (BOOL, NOR)-SDSs and (BOOL, NAND)-SDSs. These results show an interesting contrast between the complexity of GEN-PME and PME problems for (BOOL, NOR)-SDSs and (BOOL, NAND)-SDSs.

3.3 Applications

Our lower bounds for PREDECESSOR EXISTENCEproblem restrcited to (BOOL, ST)-SDSdirectly imply analogous lower bounds for Hopfield networks with sequential update. To our knowledge, such results have not been reported earlier.

Communicating finite State Machines (CFSMs) have been widely studied as models of concurrent processes. As a result, a number of models have been proposed in the literature. Since these models were proposed for different applications, they are not always equivalent. We refer the reader to [BZ83, GC86, Mi99, Pe97, Ra97, SH+96] for definitions, results, applications and the state of current research in this area. The basic setup consists of a collection of finite state machines. These machines communicate with each other via explicit channels [BZ83, Pe97, GC86] or via action symbols [Ra97, SH+96]. Our results apply to both these variants. To see this, we note the following:

1. Simple-threshold can be easily represented as finite state machines (FSMs) that essentially emulate a counter. The FSM corresponding to each node of an SDS consists of two parts, namely a control part and a part simulating the threshold function. (For some models, we can sometimes eliminate the control part.)

2. Sequential update of the nodes of an SDS can be simulated by using n distinct (one for each machine) action symbols that in effect imply that each FSM is updated in the order determined from the ordering used for the given SDS. When dealing with explicit channels, this can be done by initializing all the FIFO I/O channels and using the control part to make sure that each machine corresponding to a threshold function makes a transition only after all its inputs have been received. At that point, the transition simply consists of counting how many inputs are 1 and how many of them are 0. After this, the machine posts the result of evaluating the function on each of the output channels.

Given these observations, the remaining details are fairly straightforward. Our results show that the **NP**-hardness of PREDECESSOR EXISTENCEproblems for CFSMs holds for extremely simple individual automata. Specifically, the automata use a very simple model of concurrency and the underlying graphs are of bounded degree. This points out that a bounded amount of concurrency is sufficient to yield computational intractability results for problems for CFSMs.

3.4 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 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. (BOOL, ST)*-SDSs, where each node computes the same k-simple-threshold-function, for each $k \geq 2$.*
2. (BOOL, TALLY)*-SDSs, where each node computes the same k-tally function, for each $k \geq 1$.*
3. (BOOL, {AND, OR})*-SDSs.*
4. (BOOL, SYM)*-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 $\mathcal{S}$ 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, $1 \leq i \leq n$.) The edge set E consists of the following.

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

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

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

4. For each j, $1 \leq j \leq m$, 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, $1 \leq i \leq n$, edges $\{x_i, z_i\}$ and $\{\overline{x_i}, z_i\}$.

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

7. For each i, p, and q, such that $1 \leq i \leq n$ and $1 \leq p < q \leq k$, 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, $1 \leq i \leq n$ and $1 \leq p \leq k-1$, 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 $\mathcal{C}'$: 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 $\mathcal{C}'$, $\mathcal{S}$ reaches $\mathcal{C}$ in one step.

Now, suppose there is a configuration $\mathcal{C}'$ such that $\mathcal{S}$ can reach the configuration $\mathcal{C}$ in one step. We claim that $\mathcal{C}'$ must set the state of each node c_j and c'_j ($1 \leq j \leq m$) to 0. To see this, suppose $\mathcal{C}'$ 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 $\mathcal{C}$ 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 $\mathcal{C}'$ so that the final value of c_j becomes 1. In other words, the values assigned by $\mathcal{C}'$ 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. As will be seen, whenever the answer to a problem is "yes", our algorithms can also generate a feasible solution with the desired property.

4.2.1 Results for (FIELD, LINEAR)-SDSs

Theorem 4.2 *Let $\mathcal{S}$ be a* (FIELD, LINEAR)*-SDSs 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 (Equation (1)) is nonzero. For such a* (FIELD, LINEAR)*-SDSs the answer to the* PRE *problem is always "yes". Moreover, for a given final configuration $\mathcal{C}$, there is a* unique *predecessor configuration $\mathcal{C}'$, which can be found in time linear in the size of the underlying graph.*

Proof: Let $\mathcal{C} = (b_1, b_2, \ldots, b_n)$ 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 $\mathcal{C}'(v_j)$ has already been determined. Therefore, in the equation for determining the new value of v_i, the only unknown is $\mathcal{C}'(v_i)$. This is because the other values in the equation are from $\mathcal{C}$ for neighboring nodes before v_i in π, the already computed values from $\mathcal{C}'$ for neighboring nodes after v_i in π, and $\mathcal{C}(v_i)$ itself. Since the entry a_{ii} is not zero, this equation has a unique solution. ■

For (FIELD, LINEAR)-SDSs that do not satisfy the condition mentioned in Theorem 4.2 and for (FIELD, LINEAR)-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* (FIELD, LINEAR)*-SDSs and* (FIELD, LINEAR)*-SyDSs can be solved in polynomial time.*

Proof: First consider a (FIELD, LINEAR)-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 (FIELD, LINEAR)-SyDS, 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 (FIELD, LINEAR)-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 the linear functions over the field $\mathbb{F}_2$ under addition modulo 2. Thus, by Theorem 4.2, for any (BOOL, {XOR, XNOR})-SDS, the PRE problem has a unique solution which can be found efficiently.

4.2.2 (BOOL, SYM)-SDSs with Bounded Treewidth

Theorem 4.4 *Let $\mathcal{S}$ be a* (BOOL, SYM)*-SDS whose underlying graph has bounded treewidth. The* PRE *problem for $\mathcal{S}$ can be solved in polynomial time.*

Proof: Let k be the treewidth of the underlying graph $G(V, E)$. It is well known that a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of G can be constructed in time that is a polynomial in the size of G. Moreover, this can be done so that T is a binary tree; that is, each node of T has at most two children [Bo88].

For a given node i of the tree decomposition, we refer to the SDS nodes in X_i as **explicit nodes** of i. If a given explicit node of i is also an explicit node of the parent of i, we refer to this node as an **inherited node** of i; and if it does not occur in the parent of i, we refer to it as an **originating node** of i. We refer to the set of all explicit nodes occurring in the subtree of T rooted at i that are not explicit nodes of i as **hidden nodes** of i. (Thus, the hidden nodes of i are the union of the originating and hidden nodes of the children of i.)

Given the underlying graph $G(V, E)$ of $\mathcal{S}$, the permutation π of $\mathcal{S}$and two disjoint subsets Y and Z of V, we define the set $N(Y, Z)$ as follows.

> $N(Y, Z)$ is the set of nodes w G such that for each node $y \in Y$, $\{w, y\} \in E$, y precedes w in π, and there is no node $z \in Z$ such that $\{w, z\} \in E$ and z precedes w in π.

Intuitively, $N(Y, Z)$ is the set of nodes w of G, not in Y, such that in a transition of $\mathcal{S}$, the old value of w is an input parameter to the computation of the new value of every node in Y, but is not an input parameter to the computation of the new value of any of the nodes in Z.

Let $\mathcal{C}$ be the configuration specified in the given instance of the PRE problem for $\mathcal{S}$. Consider a given node i of the tree decomposition. Suppose α is a given assignment of state values to the explicit nodes of i and β is a given assignment of state values to the hidden nodes of i. We say that the combined assignment $\alpha \cup \beta$ is **viable** for i if for every hidden node w of i, the evaluation of the local transition function f_w gives the value $\mathcal{C}(w)$, using the value $\beta(w)$ for w, the value $\alpha \cup \beta(u)$ for every neighbor u of w that follows w in π, and the value $\mathcal{C}(u)$ for every neighbor u of w that precedes w in π. (Note that the definition of a tree decomposition ensures that every neighbor of hidden node w is either an explicit node

We say that the combined assignment $\alpha \cup \beta$ is **strongly viable** for i if the above condition holds for every node w that is either a hidden node or an originating node of i. (The definition of a tree decomposition ensures that every neighbor of an originating node of i is either an explicit node or a hidden node of i.)

For a given node i of the tree decomposition, and a given assignment β to the states of the hidden nodes of i, define a function $h_\beta : 2^{X_i} \longrightarrow \mathbb{N}$ from the nonempty subsets of X_i to the set $\mathbb{N}$ of natural numbers as follows. For a subset Y of X_i,

> $h_\beta(Y)$ is the number of hidden nodes w of i such that $w \in N(Y, X_i - Y)$ and $\beta(w) = 1$.

For a given node i of the tree decomposition, and a given assignment α to the states of the explicit nodes of i, define the set H_α to be the set of functions h from $2^{X_i} \longrightarrow \mathbb{N}$ such that there exists an assignment β to the states of the hidden nodes of i such that $\alpha \cup \beta$ is viable for i and h is h_β.

Since for any given β and Y, the maximum possible value of $h_\beta(Y)$ is the number of hidden nodes of i, and $|X_i| \leq k$ (where k is the treewidth), an upper bound on $|chal|$ is n^{2^k}.

Note: The function h_β could have been defined as a function whose domain is the nonempty subsets of X_i. However, including the empty set $\emptyset$ in the domain of h_β makes the final decision at the root node of the tree decomposition easier to describe.

We solve the PRE problem for $\mathcal{S}$ by using bottom-up dynamic programming on the decomposition tree. For each node i of T, we compute a table with an entry for each assignment α to the states of the explicit nodes of i. The value of the entry for each such assignment α is the set H_α. We refer to this table as J_i. Since

the treewidth k is a constant, the size of the table for each node of the decomposition tree is a polynomial in n, the number of nodes of the underlying graph $G(V, E)$.

For a leaf node i of the decomposition tree, every entry H_α in the table consists of the single function h that maps every subset X_i into zero. Thus, the table for each leaf node of the decomposition tree can be computed in polynomial time.

For a non leaf node i of the tree decomposition, the table for i can be computed in polynomial time from the tables of its children. To facilitate this computation, we utilize the following concepts and notation.

For a given node i of the tree decomposition, let $\widehat{X_i}$ denote the set of inherited nodes of i. (Thus, $\widehat{X_i} \subseteq X_i$.)

For a given node i of the tree decomposition, and a given assignment $\widehat{\beta}$ to the states of the originating nodes and the hidden nodes of i, define a function $\widehat{h}_{\widehat{\beta}} : 2^{\widehat{X}_i} \longrightarrow \mathbb{N}$ from the nonempty subsets of $\widehat{X_i}$ to the set $\mathbb{N}$ of natural numbers, as follows. For a subset Y of $\widehat{X_i}$,

> $\widehat{h}_{\widehat{\beta}}(Y)$ is the number of nodes w such that w is an originating or a hidden node of i, $w \in N(Y, X_i - Y)$ and $\widehat{\beta}(w) = 1$.

For a given node i of the tree decomposition, and a given assignment ψ to the states of the inherited nodes of i, define $\widehat{H}_\psi$ to be the set of functions $h\ 2^{\widehat{X}_i} \longrightarrow \mathbb{N}$ such that there exists an assignment $\widehat{\beta}$ to the states of the originating and hidden nodes of i such that $\psi \cup \widehat{\beta}$ is strongly viable for i and h is $\widehat{h}_{\widehat{\beta}}$.

For each node i of the tree decomposition, we define $\widehat{J}_i$ as a table with an entry for each assignment ψ to the states of the inherited nodes of i. The value of the entry for each such assignment is the set $\widehat{H}_\psi$.

For each node i of the tree decomposition, given table J_i, the table $\widehat{J}_i$ can be constructed in polynomial time. For each assignment α to X_i and function h in H_α, consider the assignment α to be the union of an assignment ψ to the states of the inherited nodes of i, and an assignment γ to the states of the originating nodes of i. In polynomial time, we can determine whether the combination of ψ, γ and h corresponds to a combined assignment that is strongly viable for i. If so, we combine γ and h to obtain a function $\widehat{h}$ that we union into the $\widehat{J}_i$ entry for ψ. By considering each such pair (α, h) in J_i, we can construct $\widehat{J}_i$ in polynomial time.

Consider a non leaf node i of the tree decomposition. Suppose that i has only one child, say the tree decomposition node i'. Given the table $\widehat{J}_{i'}$ for i', the table J_i can be constructed in polynomial time, as follows. Consider an assignment α to the states of the explicit nodes of i. Let ψ denote the projection of the assignment α on to $\widehat{X_i}$, the inherited nodes of i'. Then the entry for α in the table J_i for i is the set $\widehat{H}_\psi$ in the entry for ψ in the table for $\widehat{J}_{i'}$ for i', with the modification that every function $h : 2^{\widehat{X}_{i'}} \longrightarrow \mathbb{N}$ in the set $\widehat{H}_\psi$ is extended to be a function $g : 2^{\widehat{X}_i} \longrightarrow \mathbb{N}$ by setting $g(Y) = 0$ for every subset Y of X_i that contains at least one member of $X_i - X_{i'}$.

Now suppose that non leaf node i of the tree decomposition has two children, say nodes i' and i" of the tree decomposition. The tables $\widehat{J}_{i'}$ and $\widehat{J}_{i"}$ for tree nodes i' and i" are combined to produce table J_i for i as follows. Consider an assignment α to the states of the explicit nodes of i. Let ψ' and ψ" denote the projection of assignment α onto the inherited nodes of i' and i" respectively. Let every function h' in $\widehat{H}_{\psi'}$ for node i' be extended to a function $g' : 2^{\widehat{X}_i} \longrightarrow \mathbb{N}$, and every function h'' in $\widehat{H}_{\psi"}$ for node i" be extended to a function $g" : 2^{\widehat{X}_i} \longrightarrow \mathbb{N}$; both extensions are done as described above. Define the function $g' + g" : 2^{\widehat{X}_i} \longrightarrow \mathbb{N}$ as follows: $(g' + g")(Y) = g'(Y) + g"(Y)$. Thus, the entry for α in table J_i for i is the set of all functions $(g' + g")$ that can be constructed in the above manner from some h' in $\widehat{H}_{\psi'}$ from $\widehat{J}_{i'}$ and $h"$ in $\widehat{H}_{\psi"}$ from $\widehat{J}_{i"}$.

Let r be the root node of the tree decomposition. The root node has no inherited nodes, so $\widehat{X}_r = \emptyset$. Consequently, table $\widehat{J}_r$ for the root node r consists of the single entry: $(\emptyset, \widehat{H}_\emptyset)$. Set $\widehat{H}_\emptyset$ is nonempty if and only if there exists an assignment to the states of all the nodes of SDS $\mathcal{S}$ that is strongly viable for r, that is, if and only if configuration $\mathcal{C}$ has a predecessor. ■

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 Anil Kumar, Gabriel Istrate, 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.

[BE+01] C. Barrett, S. Eubank, M. Marathe, H. Mortveit and C. Reidys, invited paper to appear in *Proc. 1st International Conference on Grand Challenges in Simulations* held as a part of *Western Simulation Conference*, San Antonio Texas, 2002. Technical Report LA-UR-01-6623 Los Alamos National Laboratory.

[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.

[BH+01a] C. Barrett, H. Hunt III, M. Marathe, S. Ravi, D. Rosenkrantz, R. Stearns and P. Tosic. Gardens of Eden and fixed points in sequential dynamical systems. *Proc. of the International Conference on Discrete Models - Combinatorics, Computation and Geometry* (DM-CCG), Paris, July 2001.

[BH+01b] C. Barrett, H. Hunt III, M. Marathe, S. Ravi, D. Rosenkrantz and R. Stearns. Analysis Problems for Sequential Dynamical Systems and Communicating State Machines. *Proc. International Symposium on Mathematical Foundations of Computer Science* (MFCS'01), Czech Republic, August 2001, pp. 159–172.

[BH+01c] C. Barrett, H. Hunt III, M. Marathe, S. Ravi, D. Rosenkrantz and R. Stearns. Reachability Problems for Classes of Sequential Dynamical Systems, Submitted for publication, Sept. 2001.

[BH+01d] C. Barrett, H. Hunt III, M. Marathe, S. Ravi, D. Rosenkrantz and R. Stearns. Elements of a theory of computer simulation V: computational complexity and universality. Under preparation, May 2001.

[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.

[BM+00] C. Barrett, M. Marathe, H. Mortveit, C. Reidys, J. Smith and S.S. Ravi. "AdhopNET: Advanced Simulation-based Analysis of Ad-Hoc Networks", Los Alamos Unclassified Internal Report, 2000.

[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.

[BZ83] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2), Apr. 1983, pp. 323–342,

[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.

[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.

[GC86] M. Gouda, C. Chang. Proving Liveness for Networks of Communicating Finite State Machines. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 8(1): 154-182, pp. 1986.

[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.

[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*, 2002.)

[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.

[KMR01] S. Krumke, M. Marathe and S. Ravi. Models and Approximation Algorithms for Channel Assignment in Radio Networks. *Wireless Networks*, Vol. 7, Issue 6, Nov. 2001, pp. 567–574.

[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.

[Mi99] R. Milner. *Communicating and Mobile systems: the π-calculus.* Cambridge University Press, 1999.

[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.

[Pe97] W Peng. Deadlock Detection in Communicating Finite State Machines by Even Reachability Analysis. *Mobile Networks* (MONET), 2(3), 1997, pp. 251-257.

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

[Ra97] A. Rabinovich. Complexity of Equivalence Problems for Concurrent Systems of Finite Agents. *Information and Computation*, 127(2), 1997, pp. 164–185.

[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.

[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.

[SH+96] S. K. Shukla, H. B. Hunt III, D. J. Rosenkrantz and R. E. Stearns. On the Complexity of Relational Problems for Finite State Processes. *International Colloquium on Automata Programming and Languages (ICALP),* 1996, pp. 466-477.

[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.

Appendix

4.3 Predecessor Existence problems

4.3.1 Easiness results for general graphs

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 $\mathcal{C} = (b_1, b_2, \ldots, b_n)$ 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 begin with some preliminary definitions. Following Schaefer [Sc78] we say that 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**[7] 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 4.5 *[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. The result holds even when we are allowed to have constants as a part of the formula.* ■

Theorem 4.6 *When t is polynomial in $|\mathcal{S}|$, the t-*PRE*, 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:* (BOOL, OR)-*SDSs,*(BOOL, NOR)-*SDSs,* (BOOL, AND)-*SDSs,* (BOOL, NAND)-*SDSs,* (BOOL, XOR)-*SDSsand* (BOOL, XNOR)-*SDSs.* ■

Proof sketch: Consider any one of the problems for (BOOL, OR)-SDSs. We show that the transition of a single node v in one time unit can be represented by a weakly-negative formula $A_v(t, t+1)$. Then all the above problems can be simply encoded as $F = \bigwedge_{v,t} A_v(t, t+1)$. Some of the variables in the formula might be predetermined depending on the particular problem. Using that fact that F is weakly-negative formula combined with the result in Theorem 4.6 yields the claimed result. The results for (BOOL, AND)-SDSsyield weakly-positive formula while the problems for (BOOL, XOR)-SDSsyield affine formulas. thus results in Theorem 4.6 can be applied directly to complete the proof of the theorem. thus in the rest of the proof, we show how to represent a one step transition for (BOOL, OR)-SDSsas a weakly-negative formula.

Consider a given node w_{i_0} and its associated variable x_{i_0}. Suppose the local transition function f_i associated with w_{i_0} is OR. Let the neighbors of w_{i_0} be $w_{i_1}, \ldots w_{i_k}$ with the associated variables $x_{i_1}, \ldots x_{i_k}$. We will use $x^t_{i_j}$ $0 \leq j \leq k$ as a variable to denote the state of node v_{i_j} at time t. Without loss of generality assume that the nodes $w_{i_1}, \ldots w_{i_r}$ are updated before w_{i_0} and $w_{i_{r+1}}, \ldots w_{i_k}$ be updated after w_{i_0}. Then the

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

predecessor existence question can be equivalently expressed as a conjunction of Horn (weakly negative) clauses as follows:

$$x_{i_0}^{t+1} \equiv \left(x_{i_0}^t \vee x_{i_{r+1}}^t \vee \cdots \vee x_{i_k} \vee x_{i_1}^{t+1} \vee \cdots \vee x_{i_r}^{t+1}\right)$$

which is equivalent to

$$\left(x_{i_0}^{t+1} \rightarrow \left(x_{i_0}^t \vee x_{i_{r+1}}^t \vee \cdots \vee x_{i_k}^t \vee x_{i_1}^{t+1} \vee \cdots \vee x_{i_r}^{t+1}\right)\right) \wedge \left(\left(x_{i_0}^t \vee x_{i_{r+1}}^t \vee \cdots \vee x_{i_k}^t \vee x_{i_1}^{t+1} \vee \cdots \vee x_{i_r}^{t+1}\right) \rightarrow x_{i_0}^{t+1}\right)$$

The first implication is rewritten as

$$\mathcal{W}_1 \equiv \overline{x_{i_0}^{t+1}} \vee x_{i_0}^t \vee x_{i_{r+1}}^t \vee \cdots \vee x_{i_k}^t \vee x_{i_1}^{t+1} \vee \cdots \vee x_{i_r}^{t+1}.$$

The second implication is easily simplified using DeMorgan's law as

$$\mathcal{W}_2 \equiv \left(\overline{x_{i_0}^t} \vee x_{i_0}^{t+1}\right) \wedge \left(\overline{x_{i_{r+1}}^t} \vee x_{i_0}^{t+1}\right) \wedge \cdots \wedge \left(\overline{x_{i_k}^t} \vee x_{i_0}^{t+1}\right) \wedge \left(\overline{x_{i_1}^{t+1}} \vee x_{i_0}^{t+1}\right) \wedge \cdots \wedge \left(\overline{x_{i_r}^{t+1}} \vee x_{i_0}^{t+1}\right)$$

Both $\mathcal{W}_1$ and $\mathcal{W}_1$ are Horn (weakly negative) formulas Note that some of the $x_{i_j}^p$'s are constants since their values have been pre-determined.

(The proof for NOR is similar.) If the bit b_i corresponding to v_i in the final configuration $\mathcal{C}$ 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 $\mathcal{C}$ 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. ■

4.3.2 Treewidth Bounded and Degree 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.

Given an undirected graph $G(V, E)$, the **square** of G, denoted by $G^2(V, E^2)$, is an undirected graph in which there is an edge between nodes u and v if and only if there is a path consisting of at most two edges between the same pair of nodes in G. The following result from the literature (see for example [KMR01]) about the treewidth square graphs is used in obtaining some of the polynomial time algorithms in this paper.

Proposition 4.1 *Let G be a graph with maximum node degree Δ and treewidth k. The treewidth of G^2 is at most $\Delta(k+1) - 1$.* ■

Corollary 4.1 *Let G be a graph whose maximum node degree and treewidth are both constants. The treewidth of G^2 is also a constant.* ■

Definition 4.1 *[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 4.7 *[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 4.8 *Let $\mathcal{S}$ be an SDS where each local transition function is Boolean and the underlying graph has bounded treewidth and bounded degree. The* PRE *problem for $\mathcal{S}$ can be solved in polynomial time.*

Proof: Given SDS $\mathcal{S}$ 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 4.7, the SAT problem for the resulting set of terms can be solved in polynomial time. Thus, the PRE problem for SDS $\mathcal{S}$ 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 4.8 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 4.8 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 $|\mathcal{S}|$.

4. Finally note that combining the ideas used to prove Theorem 4.6 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 $|\mathcal{S}|$.

5 The PERMUTATION EXISTENCE Problem

5.1 NP-Completeness Results

Theorem 5.1 *The* PME *problem is* **NP**-*complete for* (BOOL, ST)-*SDSs.*

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. (BOOL, NOR)-*SDSs (*(BOOL, NAND)-*SDSs) whose underlying graphs have a maximum node degree of 3.*

2. (BOOL, SYM)-*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' ($1 \leq i \leq n$), the associated Boolean function is the NOR 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 initial configuration $\mathcal{C}'$ assigns the value 0 to each node. The final configuration $\mathcal{C}$ requires each node c_j ($1 \leq j \leq m$) to have the value 1; the values for every other node is "don't care".

This completes the specification of of the partial SDS $\mathcal{S}$. We now show that there is a permutation π such that $\mathcal{S}$ can reach one of the final configurations specified by $\mathcal{C}$ 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 $\mathcal{S}$. The first $2n$ entries of π specify the order for the nodes x_i and x_i' ($1 \leq i \leq n$). 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 $\mathcal{C}$. Thus, π is a valid solution to the constructed generalized PME instance.

Now, suppose there is a permutation π for $\mathcal{S}$ such that $\mathcal{S}$, starting from $\mathcal{C}'$, reaches one of the final configurations specified by $\mathcal{C}$ in one step. We carry out this step of $\mathcal{S}$ 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* (BOOL, OR)-*SDSs and* (BOOL, AND)-*SDSs.*

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

Consider a (BOOL, OR)-SDS $\mathcal{S}$. 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 (BOOL, NOR)-SDSs and (BOOL, 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 (BOOL, NOR)-SDSs and (BOOL, NAND)-SDSs.

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