Predecessor existence problems for finite discrete dynamical systems✩

Chris Barretta,b, Harry B. Hunt IIIc, Madhav V. Marathea,b, S.S. Ravi,*, Daniel J. Rosenkrantz, Richard E. Stearnsc, Mayur Thakurd

a Network Dynamics and Simulation Science Laboratory, Virginia Bioinformatics Institute, Blacksburg, VA 24061, United States
b Department of Computer Science, Virginia Tech, Blacksburg, VA 24061, United States
c Department of Computer Science, University at Albany - SUNY, Albany, NY 12222, United States
d Department of Computer Science, University of Missouri-Rolla, Rolla, MO 65409, United States

Received 24 March 2006; received in revised form 31 March 2007; accepted 10 April 2007

Communicated by A. Condon

Abstract

We study the predecessor existence problem for finite discrete dynamical systems. Given a finite discrete dynamical system S and a configuration C, the PREDECESSOR EXISTENCE (or PRE) problem is to determine whether there is a configuration C′ such that S has a transition from C′ to C. In addition to the decision version, we also study the following variants: the #-PREDECESSOR EXISTENCE (or #PRE) problem – counting the number of predecessors, the UNIQUE-PREDECESSOR EXISTENCE (or UPRE) problem – deciding whether there is a unique predecessor and the AMBIGUOUS-PREDECESSOR EXISTENCE (or APRE) problem – given a configuration C and a predecessor C′ of C, deciding whether there is a different predecessor C′′ of C.

General techniques are presented for simultaneously characterizing the computational complexity of the PRE problem and its three variants. Our hardness results are based on the concept of simultaneous reductions: single transformations that can be used to simultaneously prove the hardness of the different variants of the PRE problem for their respective complexity classes. Our easiness results are based on dynamic programming and they extend the previous results on PRE problem for one-dimensional cellular automata. The hardness results together with the easiness results provide a tight separation between easy and hard instances.

Further, the results imply similar bounds for other classes of finite discrete dynamical systems including discrete Hopfield and recurrent neural networks, concurrent state machines, systolic networks and one- and two-dimensional cellular automata. Our results extend the earlier results of Green, Sutner and Orponen on the complexity of the predecessor existence problem and its variants.

Keywords: Discrete dynamical systems; Cellular automata; Predecessor existence; Data flow analysis; Software and hardware verification; Computational complexity


* Corresponding author. Tel.: +1 518 442 4278; fax: +1 518 442 5638.

E-mail addresses: cbarrett@vbi.vt.edu (C. Barrett), hunt@cs.albany.edu (H.B. Hunt III), mmarathe@vbi.vt.edu (M.V. Marathe), ravi@cs.albany.edu (S.S. Ravi), djr@cs.albany.edu (D.J. Rosenkrantz), res@cs.albany.edu (R.E. Stearns), thakurk@umr.edu (M. Thakur).
1. Introduction and motivation

We study finite discrete dynamical systems and focus our attention on a particular class of such systems called Sequential Dynamical Systems. Each Sequential Dynamical System (SDS) $S$ is specified as a triple $S = (G, F, \pi)$. Here, $G(V, E)$ 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 $v_i$. $\pi$ 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 $\pi$. SDSs can be equivalently viewed as systolic networks [39] in which individual nodes (processors) are updated sequentially rather than synchronously. SDSs in which nodes are updated synchronously will be called Synchronous Dynamical Systems (SyDS). SDSs and SyDSs are closely related to classical finite cellular automata (CA) [72,23,27,61], graph automata [51], systolic networks [39], finite discrete-time recurrent neural networks [20,21,52,62] and concurrent finite state machines [24,54]. Although our focus is primarily on SDSs, our methods and results also apply to all of the above models; we will discuss this in later sections.

Here, we study the computational complexity of the Predecessor Existence problem (sometimes called the preimage existence problem) for finite discrete dynamical systems. Given an SDS $S = (G, \mathcal{F}, \pi)$ and a configuration $C$, the Predecessor Existence problem (abbreviated as PRE) is to determine whether the configuration $C$ has a predecessor; that is, whether there is a configuration $C'$ such that $S$ has a transition from $C'$ to $C$ in one step. This is a classical problem studied by the dynamical systems community in the context of CA [63,26]. In addition to the above decision problem, we study three other variants of the problem:

- **#-Predecessor Existence (or #PRE):** Counting the number of predecessors of a given configuration $C$.
- **Unique-Predecessor Existence (or UPre):** Deciding whether a given configuration has a unique predecessor.
- **Ambiguous-Predecessor Existence (or APre):** Given one predecessor $C'$ for a configuration $C$, deciding whether there is a different predecessor $C''$ for $C$.

Each of these variants belongs to a different computational complexity class. Previously these problems have been studied separately in the literature; our goal is to develop unified proof techniques to simultaneously characterize the computational complexity of these problems.

Our work was motivated by two separate programs: (i) to provide a formal basis for the design and analysis of large-scale socio-technical simulations$^1$ [6,7,18] and (ii) interest in the use of systolic arrays, reconfigurable computing and nano-scale computing architectures for parallel data processing [16,39,46,44]. The predecessor existence problem and its variants also arise naturally in three other applications: (i) reverse engineering finite discrete dynamical systems from time series data [42], (ii) modeling problems such as the spread of influence in social networks [36] and (iii) testing liveness properties of certain network protocols [24,54]. See Section 3.2 for additional discussion.

The remainder of the paper is organized as follows. Section 2 provides formal definitions of the problems considered in the paper. Section 3 summarizes our results and discusses related results from the literature. Section 4 presents our results for SDSs and SyDSs with symmetric local transition functions. Section 5 considers the PRE problem when the local transition functions are threshold functions. In Section 6 we show how our results provide a first step towards establishing a dichotomy result for the PRE problem in a manner similar to the dichotomy result for the Boolean Satisfiability problem [60]. Implications of our results to problems for other computational models such as CFSMs are presented in Section 7. Finally, Section 8 offers some concluding remarks.

2. Preliminaries

2.1. Formal definition of an SDS

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

---

$^1$ See http://www.vbi.vt.edu/ndssl for additional details on this program.
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 $\delta_i$ 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, $\pi$ is a permutation of {1, 2, …, $n$} specifying the order in which nodes update their states using their local transition functions. Alternatively, $\pi$ 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 $\pi$. 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.

```pseudocode
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
```

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 $\mathcal{C}'$. In general, a configuration in phase space may have multiple predecessors.

### 2.2. Variations of the basic SDS model

The definition of an SDS imposes no restrictions on the domain $\mathbb{D}$ of state values, the local transition functions or the underlying graph. SDSs that model simulation systems can be obtained by appropriately restricting these components.

We use the notation “($x,y,z$)-SDS” to denote an SDS where ‘$x$’ specifies the restriction on the domain, ‘$y$’ specifies the restriction on the local transition functions and ‘$z$’ specifies the restriction on the underlying graph. Tables 1–3 give the abbreviations that we use for specifying the restrictions on the three components. We also use the keyword
Table 2
Notation for restrictions on the local transition functions of SDSs and SyDSs

<table>
<thead>
<tr>
<th>Notation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYM</td>
<td>Each local transition function is symmetric.</td>
</tr>
<tr>
<td>(g, r) - SYM</td>
<td>Each local transition function is (g, r)-symmetric.</td>
</tr>
<tr>
<td>TOTALISTIC</td>
<td>Each local transition function is totalistic.</td>
</tr>
<tr>
<td>THRESH</td>
<td>Each local transition function is a simple threshold function.</td>
</tr>
<tr>
<td>1-THRESH</td>
<td>Each local transition function is a 1-simple threshold function.</td>
</tr>
<tr>
<td>WTHRESH</td>
<td>Each local transition function is a weighted threshold function.</td>
</tr>
<tr>
<td>POLY-WTHRESH</td>
<td>Each local transition function is a weighted threshold function where the weights are polynomial in the size of the graph.</td>
</tr>
<tr>
<td>BIJUNCTIVE</td>
<td>Each local transition function is bijunctive.</td>
</tr>
<tr>
<td>0-VALID</td>
<td>Each local transition function is 0-valid (i.e., has the value 1 when all the inputs are 0).</td>
</tr>
<tr>
<td>1-VALID</td>
<td>Each local transition function is 1-valid (i.e., has the value 1 when all the inputs are 1).</td>
</tr>
<tr>
<td>WEAKLY-POSITIVE</td>
<td>Each local transition function is weakly positive.</td>
</tr>
<tr>
<td>WEAKLY-NEGATIVE</td>
<td>Each local transition function is weakly negative.</td>
</tr>
<tr>
<td>OR</td>
<td>Each local transition function is the (Boolean) Or function. (A similar notation is used when each local transition function is And, Nor, etc.)</td>
</tr>
<tr>
<td>{AND, OR}</td>
<td>Each local transition function is from the set {AND, OR}. (A similar notation is used when each local transition function is from another set of functions.)</td>
</tr>
<tr>
<td>S</td>
<td>Each local transition function is from a (given) finite set S of finite arity Boolean relations. Each local transition function is a linear combination of its inputs, using the addition and multiplication operations of the algebraic field which forms the domain of the SDS or the SyDS.</td>
</tr>
<tr>
<td>LINEAR</td>
<td>Each local transition function is a linear combination of its inputs, using the addition and multiplication operations of the algebraic field which forms the domain of the SDS or the SyDS.</td>
</tr>
</tbody>
</table>

Table 3
Notation for restrictions on the underlying graphs of SDSs and SyDSs

<table>
<thead>
<tr>
<th>Notation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREE</td>
<td>The underlying graph is a tree.</td>
</tr>
<tr>
<td>STAR</td>
<td>The underlying graph is a star.</td>
</tr>
<tr>
<td>PLANAR</td>
<td>The underlying graph is planar.</td>
</tr>
<tr>
<td>GRID</td>
<td>The underlying graph is a (rectangular) grid.</td>
</tr>
<tr>
<td>TW-BOUNDED</td>
<td>The underlying graph is treewidth bounded.</td>
</tr>
<tr>
<td>DEG-TW-BOUNDED</td>
<td>The underlying graph is both degree and treewidth bounded.</td>
</tr>
</tbody>
</table>

NONE to indicate that there is no restriction on the corresponding component. Thus, for example, (BOOL, SYM, PLANAR)-SDS specifies the class of SDSs where the domain is Boolean, each (Boolean) local transition function is symmetric and the underlying graph is planar. Likewise, (FIELD, LINEAR, NONE)-SDS specifies the class of SDSs where the domain is an algebraic field, each local transition function is a linear combination of the inputs (using the addition and multiplication operators of the field) and there are no restrictions on the underlying graph.

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, local transition functions and the underlying graph are also applicable to SyDSs. So, we specify restricted versions of SyDSs in the same manner as SDSs.

2.3. Definitions of some complexity classes

In this section we review the definitions some complexity classes used in this paper. Additional information regarding these and other complexity classes can be found in [25,29,68,70].

A search problem $II$ consists of a set $D_{II}$ of objects called instances. For each instance $I \in D_{II}$, the set of solutions is denoted by $S_{II}[I]$. An algorithm is said to solve a search problem $II$ if, given $I \in D_{II}$ as input, the algorithm outputs $\textbf{no}$ if $S_{II}[I] = \emptyset$ and outputs an $s \in S_{II}[I]$ otherwise. The enumeration problem associated with a
search problem $\Pi$ is the problem of determining, given an $I \in D_\Pi$, the value of $|S_\Pi[I]|$. The class $\#P$ consists of all enumeration problems associated with search problems $\Pi$ such that all of the following conditions hold. (1) There is a nondeterministic algorithm $A$ for solving $\Pi$. (2) For all $I \in D_\Pi$, the number of distinct accepting sequences for $I$ by $A$ equals $|S_\Pi[I]|$. (3) For all $I \in D_\Pi$, the length of the longest accepting computation of $A$ on $I$ is bounded by a polynomial in the length of $I$. A reduction $f : D_\Pi \rightarrow D_{\Pi'}$ is parsimonious if and only if for all $I \in D_\Pi$, $|S_{\Pi'}[f(I)]| = |S_\Pi[I]|$. A reduction $f$ is weakly parsimonious if and only if for all $I \in D_\Pi$, $|S_{\Pi'}[f(I)]| = g(I)|S_\Pi[I]|$, where $g(I)$ is a polynomial time computable function. An enumeration problem is said to be $\#P$-hard if each problem in $\#P$ is polynomial time parsimoniously or weakly parsimoniously Turing reducible to it. If in addition, the enumeration problem is in $\#P$, the problem is said to be $\#P$-complete. The complexity class $D^P$ [70,29], is defined by

$$D^P = \{L_1 - L_2 \mid L_1, L_2 \in NP\}.$$ 

This class is appropriate for studying whether a search problem has a unique solution. Intuitively, a problem is in $D^P$ if it can be solved by asking one question in $NP$ and one question in $Co-NP$. In subsequent sections, we will show that unique versions of the PRE problem are complete$^2$ for the class $D^P$.

A number of other standard definitions (e.g. Boolean constraint satisfaction problems, graph theoretic notions such as treewidth) used in the paper are provided in the Appendices A and B.

2.4. Local replacement based ultra-efficient SIMULTANEOUS reductions

The lower bound results obtained in this paper use local replacement based simultaneous reductions; we briefly discuss this concept below. Reductions by local replacement have been used extensively in the literature (e.g., see [25]). The first step in formalizing this concept is to separate the concept of replacement from that of reduction. Reductions by local replacement construct target instances from source instances by replacing each object (e.g. clause/variable in a formula) by a collection of objects (e.g. set of nodes and associated local functions of the SDS) in the target instance. When a replacement preserves the property (semantics) we are interested in, we call it a reduction.

We show that most of our reductions by simple local replacement are ultra-efficient, meaning that:

they are simultaneously $O(n \log n)$ time-, linear size-, and $O(\log n)$ intermediate space bounded on multi-tape deterministic Turing machines.

Moreover, we show that most of the reductions by simple local replacement given here are also simultaneously parsimonious, decision preserving, and planarity of instance preserving. Using known results regarding the source instance, we thus get that a single local transformation can be used to simultaneously characterize the decision, counting, ambiguous and unique versions of the predecessor existence problem.

We call a multi-purpose reduction that simultaneously relates the sequential and parallel complexities of decision, counting, ambiguous and unique problems, for arbitrary and planar instances, a SIMULTANEOUS reduction.

Here we will concentrate on reductions that simultaneously preserve the decision complexity and are parsimonious. By a slight abuse of notation, we will use the phrase “$A$ is local replacement based (decision, parsimonious)-reducible to $B$” to mean the following: there is a local replacement based transformation from instances of $A$ to instances of $B$ that is a reduction from $A$ to $B$ as well as a parsimonious reduction from $\#A$ to $\#B$. The following lemma is a direct corollary of the known results on the complexity of 3SAT, its variants discussed in [25,29,68,70,69] and the above discussion.

Lemma 2.1. 1. If $A$ is local replacement based (decision, parsimonious)-reducible to $B$ and $B$ is local replacement based (decision, parsimonious)-reducible to $C$, then $A$ is local replacement based (decision, parsimonious)-reducible to $C$.

2. If $3SAT$ is local replacement based (decision, parsimonious)-reducible to PRE for $(x,y,z)$-SDSs and $(x,y,z)$-SyDSs, then PRE is NP-complete, $\#PRE$ is $\#P$-complete, UPRE is $D^P$-complete and APRE is NP-complete for $(x,y,z)$-SDSs and $(x,y,z)$-SyDSs.

---

$^2$Completeness for $D^P$ is proven using randomized polynomial time reductions [70]: Problem $A$ is reducible to problem $B$ by a randomized polynomial time reduction if there is a randomized polynomial time Turing machine $T$ and a polynomial $p$ such that (i) $\forall x \ [x \notin A \rightarrow T(x) \notin B]$, and (ii) $\forall x \ [x \in A \rightarrow T(x) \in B \text{ with probability at least } 1/p(|x|)]$. 

Proof. Proof of Part 1 is straightforward and is omitted. For Part 2, we modify the proof of the \(D^P\)-completeness of \(\text{UNIQUE SAT}\) in [70], so that whenever their reduction outputs a Boolean formula \(f\), we output the instance \(I_f\) of \((x,y,z)\)-\(\text{SDS}\) (or \((x,y,z)\)-\(\text{SyDS}\)), obtained by the parsimonious reduction from \(\text{SAT}\) to \(3\text{SAT}\) in [29] composed with the parsimonious reduction from \(3\text{SAT}\) to \((x,y,z)\)-\(\text{SDS}\) (or \((x,y,z)\)-\(\text{SyDS}\)) assumed above. The remaining details follow by noting that the reduction is parsimonious. Now consider the hardness of \(\text{AP RE}\). The proof is similar so we just give a sketch. Starting with an instance of \(\text{AMBIGUOUS-3SAT}\) consisting of a 3CNF formula \(f\) and a satisfying assignment \(v\), we construct an instance of \((x,y,z)\)-\(\text{SDS}\) (or \((x,y,z)\)-\(\text{SyDS}\)) consisting of \(S\) and a configuration \(C\) using the parsimonious reduction assumed in the theorem. Corresponding to the satisfying assignment \(v\) we obtain a predecessor \(C'\) of \(C\) that can be constructed in polynomial time. Now \(f\) has an additional satisfying assignment \(w\) iff \(C\) has another predecessor \(C_1\) distinct from \(C'\). ■

Definitions of the restricted forms of the \(\text{SAT}\) problem used in the following theorem can be found in the Appendices A and B.

**Theorem 2.2.** There is a local replacement based ultra-efficient (decision, parsimonious) reduction from \(3\text{SAT}\) to \(\text{PLANAR-3SAT, PLANAR MONOTONE 3SAT and PLANAR EXACTLY 1-IN-3 3SAT}\). Thus

- \(\text{PLANAR-3SAT, PLANAR-EX3SAT, PLANAR MONOTONE 3SAT and PLANAR EXACTLY 1-IN-3 3SAT are NP-complete.}\)
- \(\text{AMBIGUOUS PLANAR-3SAT, AMBIGUOUS PLANAR-EX3SAT, AMBIGUOUS PLANAR MONOTONE 3SAT and AMBIGUOUS PLANAR EXACTLY 1-IN-3 3SAT are NP-complete.}\)
- \(\text{UNIQUE PLANAR-3SAT, UNIQUE PLANAR-EX3SAT, UNIQUE PLANAR MONOTONE 3SAT and UNIQUE PLANAR EXACTLY 1-IN-3 3SAT are \(D^P\)-complete under randomized polynomial time reductions.}\)
- \(\text{#-PLANAR-3SAT, #-PLANAR-EX3SAT, #-PLANAR MONOTONE 3SAT and #-PLANAR EXACTLY 1-IN-3 3SAT are \#P-complete.}\)

Proof. The proofs of all the above results except those for \(\text{PLANAR MONOTONE 3SAT}\) can be found in [29]. We sketch a planarity preserving parsimonious reduction from \(3\text{SAT}\) to \(\text{MONOTONE 3SAT}\). Consider for example the clause \((x \lor \neg y \lor z)\). We replace this clause by the set of clauses \((x \lor w \lor z) \land (w \lor y) \lor (w \lor \neg y)\), where \(w\) is a new variable. The last two clauses simply enforce the condition that \(w \equiv \neg y\). The reduction is easily seen to be parsimonious and planarity preserving.

In all the above cases, excepting \(\text{PLANAR EX3SAT}\) and \(\text{PLANAR EXACTLY 1-3SAT}\), we can assume without loss of generality that each variable appears in no more than 3 clauses. This can be accomplished by replacing each variable by copies of variables in the following manner: for each variable \(x\) that appears in \(r\) clauses, create \(r\) copies \(x^1,\ldots,x^r\) of \(x\) and add the clauses \((x^i \lor x^{i+1}) \land (x^i \lor x^{i+1})\), \(1 \leq i \leq r - 1\). We start with such instances of \(3\text{SAT}\). The reductions in [29] preserve the number of occurrences of variables. ■

2.5. Special classes of local transition functions

Several special classes of local transition functions are considered in this paper. We provide formal definitions of these classes below. Many of these definitions can be found in [37].

Given two Boolean \(q\)-vectors \(X = \langle x_1, x_2, \ldots, x_q \rangle\) and \(Y = \langle y_1, y_2, \ldots, y_q \rangle\), define the relation \(\leq\) as follows: \(X \leq Y\) if \(x_i \leq y_i\), \(1 \leq i \leq q\). A \(q\)-input Boolean function \(f\) is monotone if \(X \leq Y\) implies that \(f(X) \leq f(Y)\).

For each integer \(k \geq 0\), 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. The \(k\)-inverted-threshold function is the Boolean complement of the \(k\)-simple-threshold function. Thus, the \(k\)-inverted-threshold function has the value 1 if \(\text{fewer than} k\) of its inputs are 1; otherwise, the value of the function is zero.

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. A Boolean function \(f\) is \(r\)-symmetric if the inputs to \(f\) can be partitioned into \(r\) classes, such that the value of \(f\) depends only on how many of the inputs in each of the \(r\) classes are 1. An SDS is \(r\)-symmetric if each of its local transition functions is \(r'\)-symmetric for some \(r' \leq r\).

Weighted threshold functions, a generalization of threshold functions, are defined as follows. Consider a Boolean function \(f\) with \(n\) Boolean inputs denoted by \(q_1, q_2, \ldots, q_n\). Suppose the weight associated with input \(q_i\) is \(w_i\). The
function $f$ is a **weighted threshold function** with threshold $\alpha$ if the value of $f$ is 1 when $\sum_{i=1}^{n} q_i w_i \geq \alpha$ and 0 otherwise.

A Boolean function $f$ is $(W, r)$-**symmetric** if it is possible to assign an integer (possibly negative) weight to each of its inputs, and partition its inputs into $r$ classes, such that each weight is at most $W$, and the value of $f$ depends only on the total weight for each class, of those inputs in the class that have value 1. An SDS is $(W, r)$-symmetric if each of its local transition functions is $(W, r')$-symmetric for some $r' \leq r$. Let $g$ be a function from $\mathbb{N}$ to $\mathbb{N}$. An SDS is $(g, r)$-symmetric if it is $(g(n), r)$-symmetric, where $n$ is the number of nodes in the underlying graph of the SDS.

Let $\mathbb{D}$ denote the domain of state values. A local transition function $f_i : \mathbb{D}^{\delta_i+1} \rightarrow \mathbb{D}$ at a node $i$, is a **totalistic function** if for each input $(x_1, \ldots, x_{\delta_i+1})$, the value of $f_i$ depends only on the sum $\sum_{1 \leq j \leq \delta_i} x_j$. Here, $x_j$ denotes the state of node $j$. In other words, a totalistic local transition function at a node maps the sum of state value of the node and those of its neighbors to a value in the domain of the corresponding SDS. It is easy to see that, over the Boolean domain, the class of symmetric functions coincides with the class of totalistic functions. For domains of larger cardinality, every totalistic function is a symmetric function; however, the converse is not necessarily true. It should be noted that totalistic functions may be nonlinear. CAs with totalistic local transition functions have been studied extensively in the past (see for example, [2,23]).

3. **Summary of results and related work**

3.1. **Summary of contributions**

We comprehensively and simultaneously characterize the computational complexity of PREDECESSOR EXISTENCE, #PREDECESSOR EXISTENCE, AMBIGUOUS-PREDECESSOR EXISTENCE and UNIQUE-PREDECESSOR EXISTENCE problems for SDSs and SyDSs. Our work is motivated by the earlier work of (i) Sutner [63,64] and Green [26] who considered the PRE problem and its generalizations in the context of 1D- and 2D-CA and (ii) Floreen and Orponen [20,21,52] who considered complexity theoretic questions related to phase space properties of discrete finite recurrent neural networks. Our work extends the above mentioned results in [63,26,52] on the complexity of the PREDECESSOR EXISTENCE problem for 1D- and 2D-CA. For instance, we show that PRE for 2D-CA is NP-complete, even when restricted to Boolean symmetric local transition functions. To our knowledge, unified methods for characterizing the computational complexity of the PRE problem and its variants have not been studied comprehensively prior to this work. The paper makes the following general contributions.

1. **Comprehensive characterization of computational complexity.** Our results can be categorized along four different axes shown in Fig. 1. These axes reflect (i) the expressiveness of the local functions (ii) structure of the underlying graphs, (iii) the update order and (iv) the problem class. In case of hardness results, our transformations simultaneously yield #P-completeness of #PREDECESSOR EXISTENCE problem, $\mathbb{D}^P$-completeness of UNIQUE-PREDECESSOR EXISTENCE problem and NP-completeness of the AMBIGUOUS-PREDECESSOR EXISTENCE problem. The restrictions on graphs and functions studied are shown in Fig. 2.

Our results, characterizing the complexity of the PRE, #PRE, UPRE and APRE problems, provide in many cases a tight separation between easy and hard instances. In addition, the results yield a trade-off between the connectivity of the underlying graph and the class of local functions used in determining the complexity of the problem. As an example, Fig. 3 summarizes some of the results that illustrate the tight upper and lower bounds obtained in this paper. We note the trade-off between graph theoretic structure and expressiveness of local transition functions in the hardness results shown in Fig. 3. Thus, the upper and lower bounds are close to being tight: any generalization in terms of allowing more expressive functions or more general graphs is not possible unless $P = \mathbb{NP}$. For example, consider SDSs (SyDSs) with weighted threshold functions. The complexity of the problem varies with degree, treewidth and the size of edge weights. Our results summarized in Fig. 3 tightly capture the variation. These results should be considered while keeping in mind the function and graph hierarchies shown in Fig. 2.

In an attempt to obtain tight bounds, this paper initiates the study of discrete dynamical systems on graphs of bounded treewidth. 1D-CA with radius $R$ can be equivalently viewed as SyDSs on graphs with treewidth $R$. Similarly, 2D-CA with radius $R$ can be viewed as SyDSs on graphs of treewidth $O(\sqrt{Rn})$. Thus, SyDSs on treewidth bounded graphs can be viewed as generalized CAs. We show that the predecessor existence problems and their variants are efficiently solvable for SDSs and SyDSs for a large class of local transition functions when the underlying graphs are of bounded treewidth. Intuitively, treewidth bounded graphs are generalizations of trees and bandwidth bounded
Fig. 1. Figure showing the four axes along which the PRE problem is categorized. The graph classes considered include arbitrary (unbounded treewidth) graphs, planar graphs, grids (treewidth $\Theta(\sqrt{n})$), graphs whose treewidth is bounded by a constant and trees. The problems considered include PRE, #PRE, APRE, and UPRE. Both sequential and synchronous update orders are considered. Finally, the classes of local transition functions considered include arbitrary, symmetric, simple threshold, weighted threshold, affine (linear), bijunctive, weakly positive (negative) and 0-valid (1-valid).

Fig. 2. Figure showing how the various graphs and local transition functions studied in this paper are related. An directed arrow from A to B represents an “is a” relation.

The PRE problem is \textit{NP}-complete, APRE problem is \textit{NP}-complete, #PRE problem is \textit{#P}-complete and UPRE problem is \textit{D}^\text{P}-complete for the following classes of finite discrete dynamical systems.

- (BOOL, SYM, GRID)-SDSs (SyDSs) (i.e., for graphs of treewidth $\Theta(\sqrt{n})$) (Theorem 4.1).
- (BOOL, THRESH, PLANAR)-SDSs (SyDSs), even when each node computes the same Boolean $k$-simple-threshold function (which is symmetric and monotone), for $k = 2, 3$ (remark following Theorem 5.3).
- (BOOL, WTHRESH, TW-BOUNDED)-SDSs, even when the bound on the treewidth is $2$ (Theorem 5.8).
- (BOOL, NONE, STAR)-SDSs (Proposition B.1).

In contrast, the PRE, #PRE, UPRE and APRE problems can be solved in polynomial time for the following classes of SDSs (SyDSs).

- (FIELD, LINEAR, NONE)-SDSs (SyDSs) (Theorem 6.5).
- (BOOL, SYM, TW-BOUNDED)-SDSs (SyDSs) (Theorem 4.3) and (BOOL, POLY-WTHRESH, TW-BOUNDED)-SDSs (SyDSs) (Corollary 4.4).
- (BOOL, NONE, DEG-TW-BOUNDED)-SDSs (SyDSs) (Corollary 4.5).
- (BOOL, WTHRESH, TREE)-SDSs (SyDSs): In this case, while PRE is in \textit{P} (Theorem 5.5), #PRE is \textit{#P}-complete (Proposition 5.7).

Fig. 3. Example of tight results obtained in this paper on the computational complexity of PRE problem and its variants. Note the interplay between the graph structure and function complexity. These results also imply analogous results for discrete Hopfield networks, concurrent transition systems and other related models.

graphs that still are amenable to dynamic programming based solution methods. Statistical physicists have long studied Ising, Potts and other percolation models on trees (e.g. Cayley trees).
2. **Local replacement based efficient SIMULTANEOUS reductions.** In contrast to the earlier work in computational complexity theory, we present unified proof techniques to *simultaneously* characterize the computational complexity of all the four variants of the PRE problem. The single transformations simultaneously preserve various graph theoretic properties of the underlying instances; more importantly, they are decision preserving and number preserving (parsimonious). The existence of such multi-purpose reductions, for computational problems, is probably not surprising. Nevertheless, results such as those presented here, restricted to decision or to counting problems, are rare in the literature. Indeed, there has been little prior work even on the identification of reductions, for particular problems, that are SIMULTANEOUS in the general sense used here. Simultaneous reductions to characterize the computational complexity of problems arising in the study of discrete dynamical systems have not been considered prior to this work. Apart from the immediate benefit (i.e., unified proofs of lower bounds) that such reductions provide, they also have important implications to computational complexity theory. We briefly discuss a few of these implications below.

1. As mentioned earlier, most of our reductions are ultra-efficient. This, among other things, implies tight bounds on the deterministic time complexities of these problems; in particular, the results imply that the PRE problem and its variants have the same power index [34,33,32,58] as the corresponding Boolean satisfiability problems. Recently a number of authors have investigated the deterministic time complexity of NP-hard combinatorial problems; see [34,58] and the references therein.

2. Second, using general *lifting theorems* analogous to the ones given in [45,31], local replacement based SIMULTANEOUS reductions immediately imply appropriate lower bounds on the complexity of the PRE problem and its variants when the underlying graphs (and the functions) are specified succinctly using hierarchical or periodic specifications. For example, we get that PRE problem is PSPACE-hard for (BOOL, SYM, PLANAR)-SDSs when the underlying instances are specified hierarchically using the hierarchical specification scheme proposed in [30] or using the one-dimensional finite periodic specification scheme proposed in [45,31,30].

3. Finally, local replacement based SIMULTANEOUS reductions provide additional examples of the special kinds of morphisms discussed in [22,45,31]. One motivation for this is provided by the recent results of Freedman [22] on finding methods for separating P and NP. As discussed in [45,31], this requires the development of reductions that tightly relate the complexity of problems as one changes the specification scheme used to describe the instances.

3. **Towards dichotomy theorems.** Here we initiate a new method for classifying the complexity of the PRE problem and its variants. Classifying CA has been a topic of extensive research after the early work by Wolfram [72]; see [27,72,23,61] for more on this subject. Informally speaking, Wolfram’s classification roughly corresponds to types of attractors of dynamical systems. Later, Culik, Pachl and Yu [13,23,61] made this classification more formal. Another elegant result of Sutner [64] in this direction shows that the PRE problem for SyDS is solvable in polynomial time when each local function computes addition over a group but is NP-complete when addition is computed over an arbitrary monoid. In addition, the classification of such systems depends on the specific question that one is trying to solve. Classification of SDSs (SyDSs) can be done on the basis of any of the attributes of such systems: (i) the underlying graph or (ii) the local transition functions. The lower bound results mentioned above combined with the polynomial time results provide a weak classification based on the underlying graph. We also initiate a study to classify the computational complexity of the PREDECESSOR EXISTENCE problem for SDS and SyDS based on local transition functions. Our approach was inspired by the work of Schaefer [60] who classified the computational complexity of Boolean constraint satisfaction problems based on the relations expressed by individual clauses. In particular, he shows that depending on the type of constraint, each such problem is either polynomial time solvable or is NP-complete. Recently there has been renewed interest in obtaining similar dichotomy results for counting and optimization problems and for succinctly specified satisfiability problems; see [10,12,31]. The SIMULTANEOUS reductions used in this paper, allow us to make progress in this direction in a unified way. The results are outlined in Section 6.

---

3 Recently, SIMULTANEOUS reductions for several variant decision and optimization Boolean satisfiability problems have been proposed in [12,31,32,38].

4 The reduction showing the hardness of PRE problems for grids is technically not ultra-efficient. Nevertheless, we can show that this has to be the case by providing tighter upper bounds on the deterministic time complexity of the problems.
4. Applications to other discrete dynamical system models. SDSs and SyDSs are closely related to other well known models of discrete dynamical systems. These include (a) Classical CA (see for example, [72]) — a widely studied class of discrete dynamical systems in physics and complex systems, (b) Discrete recurrent Hopfield networks [20,21,52,62,53] which are used in machine learning and pattern recognition, (c) Concurrent and communicating finite state machines [24,28,57,59] which are used to model and verify distributed systems, and (d) Systolic arrays proposed by Kung et al. and their recent extension to spatial computing proposed by DeHon for massively parallel data processing [14,39,44,16]. The hardness results for SDSs and SyDSs immediately imply analogous results for each of the above models; these applications are outlined in Section 7. For example, our polynomial time results for (BOOL, SYM, TW-BOUNDED)-SDSs imply analogous results for Hopfield networks when restricted to treewidth bounded graphs.

3.2. Related work

Computational aspects of CA have been studied by a number of researchers; see for example [43,17,23,56,72,27,26,63]. Much of this work addresses decidability of properties for infinite CA. Barrett, Mortveit and Reidys [8,47,55] and Laubenbacher and Pareigis [41,40] investigate the mathematical properties of sequential dynamical systems. Questions concerning the existence of fixed points and garden of Eden configurations in SDSs are addressed in [5,21,53,65,66]. Experimental results from the use of a SAT solver to analyze the properties of a CA are reported in [15]. See a recent comprehensive survey by Sarkar [61] for additional information.

The PRE problem was shown to be NP-complete for finite 2D-CA by Sutner [63] and Green [26]. Sutner also showed that the PRE problem for finite 1D-CA with a fixed neighborhood radius can be solved in polynomial time. As mentioned earlier, Green [26] studied generalized versions of the PRE problem for infinite CA. The problems were so formulated that his hardness results are also applicable to finite one-dimensional CA.

The results presented here are applicable to automated behavior analysis of large-scale distributed systems. The problem of finding predecessors and its variants (e.g. counting the number of predecessors) are closely related to the problem of detecting unreachable states in distributed systems [11]: a state is unreachable iff it has no predecessors (preimages). Our lower bound results show that deciding whether a given state is unreachable is intractable even for very simple classes of concurrent transition systems. In contrast, the results for treewidth bounded graphs show how graph theoretic structure can be exploited to obtain polynomial time results for detecting unreachable states. Moreover, the algorithm for treewidth bounded graphs allows us to specify any finite set of linear constraints on the states of the predecessors or on the target states. For example, when the underlying graph has bounded treewidth and the local transition functions are symmetric, our results lead to a polynomial algorithm for the following problem: Suppose $A$ is the set of configurations of an SDS $S$ such that for every configuration in $A$, at most five nodes have the state value 1. Count the number of predecessors of the configurations in $A$.

The predecessor existence problems for dynamical systems studied in this paper can also serve as a way of modeling problems such as the spread of influence in social networks [36]. In such problems, the general goal is to identify a suitable initial subset of individuals to publicize an idea so that the number of individuals to whom the idea will propagate over time through the underlying social network is maximized. Such an approach is also useful in marketing products (e.g. identifying individuals to whom free samples should be sent) and in studying the spread of infectious diseases (e.g. identifying a subset of individuals who should be vaccinated so as to minimize the rate at which a disease spreads). When a given social network is thought of as the underlying graph of an SDS (or a SyDS), the target of a marketing campaign can be modeled by a collection of configurations in the phase space of the SDS. (For example, if the marketing goal is to reach at least $N$ individuals, then each configuration in which $N$ or more state values are 1 is a suitable target for the campaign.) The initial group of individuals to be chosen can be then be thought of as finding not an immediate predecessor, but an ancestor that is $t$-generations early, for some suitable $t$.

Recently, there has also been substantial interest in the use of agent based models for understanding problems in social science [3,48,49,19]. Several researchers (see e.g. [3,4,1,50]) have demonstrated that network topology, agent behavior and update order (schedule or activation regime) can have substantial impact on the overall dynamic behavior of multiagent systems. In particular, Axtell [3] points out the need for understanding the intrinsic computational intractability of basic social processes for making further progress in modeling such processes. Our results shed some light in this direction by identifying the effects of local functions, update schedules and network structure on the computational intractability of some basic problems.
Additional terminology: The remainder of this paper presents proofs of our results for the PRE problem and its variants. We now mention some terminology used in these proofs. Suppose $C$ and $C'$ are configurations such that $C'$ is a predecessor of $C$. In our proofs, we use the terms “initial value” and “final value” of a node $v$ to mean the state of $v$ in $C'$ and $C$ respectively.

4. Results for systems with symmetric functions

4.1. Hardness results for grids

**Theorem 4.1.** There is a local replacement based (decision, parsimonious)-reduction from $3$SAT to PRE problem for $(\text{BOOL}, \text{SYM}, \text{GRID})$-SDSs and $(\text{BOOL}, \text{SYM}, \text{GRID})$-SyDSs. Thus $\text{PRE}$ is $\mathcal{NP}$-complete, $\#\text{PRE}$ is $\mathcal{#P}$-complete, $\text{APRE}$ is $\mathcal{NP}$-complete and $\text{UPRE}$ is $\mathcal{D}^P$-complete for $(\text{BOOL}, \text{SYM}, \text{GRID})$-SDSs and $(\text{BOOL}, \text{SYM}, \text{GRID})$-SyDSs.

**Proof.** We present a reduction from the restricted version of $\text{PLANAR}$-$3$SAT ($\text{RP3}$SAT) problem defined in the Appendices A and B. In this restricted version, each clause has exactly three literals and each variable occurs in at most three clauses. We will first present the proof details for $(\text{BOOL}, \text{SYM}, \text{GRID})$-SDSs and then indicate the necessary modifications for $(\text{BOOL}, \text{SYM}, \text{GRID})$-SyDSs. Throughout this proof, the reader should bear in mind that for each node $v$, the neighborhood of $v$ includes $v$ itself.

Let $I$ denote the given instance of the RP3SAT problem with variable set $X = \{x_1, x_2, \ldots, x_n\}$ and clause set $C = \{C_1, C_2, \ldots, C_m\}$. Since the bipartite graph $G_I$ corresponding to instance $I$ (also called the factor graph of $I$ [35]) has maximum node degree three, $G_I$ can be embedded on a grid in polynomial time [67]. The nodes in the resulting grid can be partitioned into the following four categories:

1. **Variable nodes:** There is one grid node corresponding to each Boolean variable in the instance $I$.
2. **Clause nodes:** There is one grid node corresponding to each clause in the instance $I$.
3. **Routing nodes:** For each occurrence of a literal in a clause, there is a path consisting of routing nodes. The path goes from the variable node to the clause node.
4. **Neutral nodes:** These are grid nodes that are not covered by any of (i), (ii) and (iii) above.

An example of an RP3SAT problem instance and an initial grid embedding for the instance are shown in Fig. 4. (In that figure, we have numbered the rows and columns of the grid for convenience in describing some modifications to the embedding.)

We may assume without loss of generality that the grid embedding satisfies the following properties.

**Property I:** If any two non-neutral nodes are adjacent, they are part of the same path (consisting of a variable node, a sequence of routing nodes and a clause node) between a variable node and a clause node. This condition can be enforced as follows. Assume that two non-neutral nodes belonging to two different paths are adjacent along a vertical edge. (The following construction can be readily modified if the adjacency is along a horizontal edge.) Let $r$ and $r + 1$ denote the number of the rows in which the two non-neutral nodes are located. Introduce a new row of grid nodes between rows $r$ and $r + 1$; for convenience, let $r'$ denote the new row. A node $v$ in row $r'$ is chosen as a routing node if it is placed on a vertical edge which is part of a variable to clause path; otherwise, node $v$ is made a neutral node.

To illustrate this construction, consider the initial embedding shown in Fig. 4. In that figure, let $w_1$ denote the routing node at row 1, column 5. Note that $w_1$, which is part of the $x_4$ to $C_2$ path, is adjacent along a vertical edge to the variable node $x_3$. To remedy this situation, we introduce a new row of nodes (denoted by 1') between rows 1 and 2, as shown in Fig. 5.

**Property II:** Each path from a variable node to a clause node has at least one routing node. This condition can also be enforced by the introduction of a new row or column of nodes as mentioned in Property I above.

To illustrate this, consider the embedding shown in Fig. 5. Note that the path from the variable node $x_3$ to clause node $C_2$ consists of just a single edge without any routing nodes. To remedy this situation, we introduce a new column of nodes (denoted by 5') between columns 5 and 6, as shown in Fig. 6.

**Property III:** Every non-neutral node has four neighbors. In the given grid embedding, each non-neutral node with three or fewer neighbors is on the boundary of the grid. Thus, Property III can be enforced by adding one or more of the following: a new row of nodes at the top and/or the bottom, a new column of nodes at the left and/or right. In this construction, the nodes in all the new rows/columns are neutral nodes.
Fig. 4. Example showing an initial grid embedding of an RP3SAT instance. Dark edges of the grid indicate paths from variable nodes to clause nodes. (The rows and columns of the grid have been numbered for convenience in describing some transformations of the initial embedding.)

Fig. 5. Modified grid embedding obtained from Fig. 4 to satisfy Property I.

As an example, in the embedding shown in Fig. 6, there are non-neutral nodes along all the four boundaries of the grid. The reader can readily visualize the addition of two new rows (one at the top and another at the bottom) of neutral nodes and two new columns (one to the left and one to the right) of neutral nodes to Fig. 6 so that Property III is satisfied.

**Property IV:** Each clause node $c$ has a neutral neighbor $v$ that itself has exactly one non-neutral neighbor (namely $c$). Once properties I through III are enforced, each clause node $c$ has exactly 4 neighbors, 3 of which are routing nodes and one a neutral node $v$. We can enforce the current property by adding one or more of the following: a new row of nodes at the top and/or the bottom, a new column of nodes at the left and/or right.
As an example, in the embedding shown in Fig. 6, the clause node $C_1$ has a neutral neighbor $v$ (at location $(3, 3)$) that has more than one (in fact, three—$C_1$ and two routing nodes at $(4, 3)$ and $(3, 2)$) non-neutral neighbors. To enforce property IV for $C_1$, we can insert a new row between rows 3 and 4 and a new column between columns 2 and 3.

Properties I and III together imply that each routing node has exactly 2 neighbors that are neutral nodes. This fact will be used later in the proof.

Let $U$ be the set of neutral nodes adjacent to some clause node. By Property IV, there is a one-to-one, onto mapping between the set of clause nodes and $U$.

Given the instance $I$ of RP3SAT, the instance $I'$ of PRE problem for a (BOOL, SYM, GRID)-SDS is constructed as follows. The underlying graph $G$ of the SDS $S$ is the grid satisfying the above three properties. The local transition functions are as follows.

1. For each clause node $v$, the local transition function $f_v$ is the 3-simple-threshold function.
2. For each variable node $v$, the local transition function $f_v$ is the OR function.
3. For each neutral node $v$, the local transition function $f_v$ is the AND function.
4. For each routing node $v$, the local transition function $f_v$ is chosen as follows.
   
   (i) If $v$ is not adjacent to a variable node, the function $f_v$ is 1 when exactly 3 or exactly 5 of its inputs are 1; for all other input combinations, $f_v$ is 0.
   
   (ii) If $v$ is adjacent to a variable node and the route corresponds to the occurrence of a positive literal, then the function $f_v$ is 1 when exactly 3 or exactly 5 of its inputs are 1; for all other input combinations, $f_v$ is 0.
   
   (iii) If $v$ is adjacent to a variable node and the route corresponds to the occurrence of a negative literal, then the function $f_v$ is 1 when exactly 4 of its inputs are 1; for all other input combinations, $f_v$ is 0.

Note that each local transition function is symmetric.

To describe the permutation, we use the following notation. Let $\mathcal{C}$ denote an arbitrary ordering of the clause nodes. Suppose there are $t$ clause to variable paths in the given embedding. For path $i$, let $\rho_i$ denote an ordering of routing nodes in the path listed in order from the clause node to the variable node, $1 \leq i \leq t$. Let $V$ and $N$ denote respectively an arbitrary ordering of the variable nodes and neutral nodes. Let $U$ denote an arbitrary ordering of the set of neutral nodes adjacent to some clause node. The permutation $\pi$ is given by

$$\pi = (U, \mathcal{C}, \rho_1, \rho_2, \ldots, \rho_t, V, N - U).$$

The required final configuration $\mathcal{C}$ has the value 1 for all the nodes. This completes the specification of the PRE instance $I'$. The following lemma is a consequence of the above construction.
Lemma 4.2. Let $C'$ be a predecessor of $C$.

1. For every neutral node $v$, $C'(v) = 1$.
2. For every clause node $c$, $C'(c) = 1$.
3. Consider any routing path that corresponds to the positive occurrence of a variable in a clause. Let $w$ denote the node corresponding to the variable. For each routing node $v$ along the path, $C'(v) = C'(w)$.
4. Consider any routing path that corresponds to the negative occurrence of a variable in a clause. Let $w$ denote the node corresponding to the variable. For each routing node $v$ along the path, $C'(v) = C'(w)$, where $C'(w)$ denotes the complement of $C'(w)$.

Proof of Lemma 4.2. Part 1 holds because the final value of each node is 1, and for each neutral node $v$, the local transition function $f_v$ is the AND function.

To see that Part 2 holds, note that corresponding to each clause node $c$, there is a node $v \in \mathbb{U}$. The neighbors of $v$ are $c$ and some neutral nodes. Since $v$ is the first to be evaluated among all its neighbors, if $C'(c) = 0$, then $C'(v) = 0$, which is a contradiction. Thus, $C'(c) = 1$.

To show that Part 3 holds, let $p$ be a routing path from clause node $c$ to variable node $w$. Let $w$ have a positive occurrence in $c$. Let nodes $v_1, v_2, \ldots, v_k$ be routing nodes (in order from $c$ to $w$) in $p$. As noted earlier, each of these routing nodes is adjacent to exactly two neutral nodes. It follows from Parts 1 and 2 above that when $v_1$ is being updated, the values of $c$ and the two neutral nodes adjacent to $v_1$ are all 1. The local transition function at $v_1$ outputs 1 exactly if 3 or 5 of its inputs are 1. Thus, if $C(v_1) = 1$, then $C'(v_1) = C'(v_2)$. We can similarly argue that $C'(v_1) = C'(v_2) = \cdots = C'(v_k) = C'(w)$. However, as noted earlier, each routing path contains exactly $4$ or $6$ nodes, so $C'(v_k)$ is only evaluated if $4$ of the inputs to $v_k$ are 1. Thus, exactly one of $v_k$ and $w$ must be 1. In other words, $C'(v_k) = \overline{C'(w)}$. □

We can now show that the RP3SAT instance $I$ has a solution if and only if the PRE instance $I'$ has a solution.

Suppose the RP3SAT instance $I$ has a satisfying assignment. For each variable $x$, let $\tau(x)$ denote the truth value in the given assignment. Construct the following configuration $C'$ for the SDS $S$.

(a) For each variable node $v$, let $C'(v)$ be the value assigned by $\tau$ to the variable corresponding to $v$.
(b) For each clause node $c$, let $C'(c) = 1$.
(c) For each neutral node $y$, let $C'(y) = 1$.

Using Lemma 4.2, it can be verified that $C'$ constructed above is a predecessor of $C$.

Now, suppose the PRE instance has a solution, and $C'$ denotes a predecessor of $C$. Construct the following truth assignment. For each variable $x$, let $v_x$ denote the corresponding node of SDS $S$. Set $x$ to $C'(v_x)$. Again using Lemma 4.2, it can be verified that the resulting assignment satisfies all the clauses of the RP3SAT instance $I$. This completes the proof of Theorem 4.1 for SDSs.

To see that the construction is parsimonious, we note the following. The construction forces the initial values of all the clause nodes and neutral nodes to 1. Further, for each clause to variable path, all the routing nodes in the path are forced to have the same initial value. (This value is equal to the truth value assigned to the variable node or its complement depending on whether the path corresponds to a positive or negative occurrence of the variable in the clause.) Thus, there is a one-to-one correspondence between the set of satisfying assignments to the given RP3SAT instance and the set of predecessors in the constructed PRE instance.

We now present the modifications needed to prove the result for SyDSs. Define a pure neutral node to be a neutral node all of whose neighbors are neutral nodes. The following additional assumptions regarding the grid layout of the bipartite graph $G_I$ of the RP3SAT instance $I$ are used. (These properties can be enforced in a manner similar to that described in the proof for SDSs.)
Property I: Every neutral node is either a pure neutral node or adjacent to a pure neutral node.

Property II: Every path between a variable node and a clause node contains at least two routing nodes.

The local transition functions for each clause node and variable node are as in the construction for SDSs. For other nodes, the local transition functions are chosen as follows.

(1) For each node \( v \) which is either a pure neutral node or a neutral node adjacent to a clause node, the local transition function \( f_v \) is the AND function. (The former forces the initial values of all neutral nodes to be 1 and the latter forces the initial values of all clause nodes to be 1.) For all other neutral nodes \( v \), the local transition function is the constant function \( f_v = 1 \).

(2) For each routing node \( v \), the local transition function \( f_v \) is chosen as follows.
   
   (i) If \( v \) is adjacent to a clause node, the function \( f_v \) is 1 when exactly 3 or exactly 5 of its inputs are 1; for all other input combinations, \( f_v \) is 0. (This forces the routing node to have the same initial value as its neighbor that is also a routing node.)
   
   (ii) If \( v \) is adjacent to two other routing nodes, the function \( f_v \) is 1 when exactly 2 or exactly 5 of its inputs are 1; for all other input combinations, \( f_v \) is 0. (This forces the routing node to have the same initial value as its two neighboring routing nodes.)

   (iii) If \( v \) is adjacent to a variable node and the route corresponds to the occurrence of a positive literal, then the function \( f_v \) is 1 when exactly 2 or exactly 5 of its inputs are 1; for all other input combinations, \( f_v \) is 0. (This forces the routing node to have the same initial value as the variable node.)

   (iv) If \( v \) is adjacent to a variable node and the route corresponds to the occurrence of a negative literal, then the function \( f_v \) is 1 when exactly 3 or exactly 4 of its inputs are 1; for all other input combinations, \( f_v \) is 0. (This forces the routing node to have the initial value that is the complement of the value assigned to the variable node.)

The proof that the resulting PRE instance has solution if and only if the RP3SAT instance has a solution is similar to that for the case of SDSs. Further, the construction is also parsimonious. □

4.2. Polynomial time algorithms for graphs of bounded treewidth

This section presents polynomial time algorithms for the PRE problem and its extensions for (BOOL, SYM, TW-BOUNDED)-SDSs. Several extensions of these polynomial time algorithms are also presented. In contrast, it can be shown that the PRE problem is NP-complete even when at most one of the local transition functions is not symmetric and the underlying graph is a star, which has a treewidth of 1. (This NP-completeness result is included as Proposition B.1 in the Appendix B.)

Theorem 4.3. The PRE, APRE, #PRE and UPRE problems for (BOOL, SYM, TW-BOUNDED)-SDSs (SyDSs) are in P.

Proof. For brevity, we will give the algorithms for SDSs. The algorithms for SyDSs are similar.

We begin with the PRE problem. Let \( S \) be a (BOOL, SYM, TW-BOUNDED)-SDS, whose underlying graph \( G(V, E) \) has a treewidth of \( k \). It is well known that a tree decomposition \( (\{X_i \mid i \in I\}, \mathcal{T} = (I, F)) \) 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 [9].

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 \( i \).) If any node \( i \) of \( T \) with fewer than two children contains no originating node, then \( T \) can be modified by combining node \( i \) with its parent in \( T \). Thus, without loss of generality, we can assume that the number of nodes of \( T \) with fewer than two children is at most \( n \), the number of nodes in \( G \). Moreover, since this bound applies to the number of leaves in \( T \), the number of nodes of \( T \) with two children is also less than \( n \).

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

We say that the combined assignment $\alpha \cup \beta$ is \textit{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 $\beta$ to the states of the hidden nodes of $i$, define a function $h_\beta : X_i \rightarrow \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers, as follows. For $y \in X_i$,

$$h_\beta(y) \text{ is the number of hidden nodes } w \text{ of } i \text{ such that } \{w, y\} \in E, y \text{ precedes } w \text{ in } \pi, \text{ and } \beta(w) = 1.$$ 

The value $h_\beta(y)$ is the number of hidden nodes $w$ of $i$ whose initial value is an input parameter to the computation of the final value of $y$, and $\beta(w) = 1$.

For a given node $i$ of the tree decomposition, and a given assignment $\alpha$ to the states of the explicit nodes of $i$, define the set $H_\beta$ to be the set of functions $h$ from $X_i$ to $\mathbb{N}$ such that there exists an assignment $\beta$ to the states of the hidden nodes of $i$ satisfying the following two conditions: $\alpha \cup \beta$ is viable for $i$ and $h$ is $h_\beta$.

Let $d$ be the maximum node degree of $G$. For any given $\beta$ and $y \in X_i$, the maximum possible value of $h_\beta(y)$ is $d$. The maximum possible value of $|X_i|$ is $k + 1$ (where $k$ is the treewidth). So, function $h_\beta$ can be represented as a table with at most $k + 1$ entries, where each entry is an integer value in the range 0 through $d$. Hence, an upper bound on $|H_\beta|$ is $(d + 1)^{(k + 1)}$.

We solve the PRE problem for $S$ 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 $\alpha$ to the states of the explicit nodes of $i$. The value of the entry for each such assignment $\alpha$ is the set $H_\alpha$. We refer to the entire table for node $i$ 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 $H_\alpha$ in table $J_i$ consists of the single function $h$ that maps every $y \in X_i$ to zero.

For a nonleaf node $i$ of the tree decomposition, the table for $i$ can be computed 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 $X_i$ denote the set of inherited nodes of $i$. (Thus, $X_i \subseteq X_i$.)

For a given node $i$ of the tree decomposition, and a given assignment $\beta$ to the states of the originating nodes and the hidden nodes of $i$, define a function $h_\beta : X_i \rightarrow \mathbb{N}$ as follows. For $y \in X_i$,

$$h_\beta(y) \text{ is the number of nodes } w \text{ such that } w \text{ is an originating or a hidden node of } i, \{w, y\} \in E, y \text{ precedes } w \text{ in } \pi, \text{ and } \beta(w) = 1.$$ 

For a given node $i$ of the tree decomposition, and a given assignment $\psi$ to the states of the inherited nodes of $i$, define $H_\psi$ to be the set of functions $h$ from $X_i$ to $\mathbb{N}$ such that there exists an assignment $\beta$ to the states of the originating and hidden nodes of $i$ satisfying the following two conditions: $\psi \cup \beta$ is strongly viable for $i$ and $h$ is $h_\beta$.

For each node $i$ of the tree decomposition, we define $J_i$ as a table with an entry for each assignment $\psi$ to the states of the inherited nodes of $i$. The value of the entry for each such assignment $\psi$ is the set $H_\psi$.

For each node $i$ of the tree decomposition, given table $J_i$, the table $J_i$ can be constructed as follows. Each assignment $\alpha$ to $X_i$ can be considered to be the union of an assignment $\psi$ to the states of the inherited nodes of $i$, and an assignment $\gamma$ to the states of the originating nodes of $i$. Because each local transition function of $S$ is symmetric, each assignment $\beta$ to the hidden nodes of $i$ can be characterized by function $h_\beta$. For an assignment $\alpha$ to $X_i$ and each function $h$ in $H_\alpha$, we can determine whether $\alpha$ and $h$ correspond to a combined assignment that is strongly viable for $i$. If so, we combine $\gamma$ and $h$ to obtain a function $\hat{h}$ that we union into the $J_i$ entry for $\psi$. By considering each such pair $(\alpha, h)$, we can construct $J_i$.

Consider a nonleaf node $i$ of the tree decomposition. Suppose that $i$ has only one child, say the tree decomposition node $i'$. Given the table $J_i$ for $i'$, the table $J_i$ can be constructed as follows. Consider an assignment $\alpha$ to the states of the explicit nodes of $i$. Let $\psi$ denote the projection of the assignment $\alpha$ onto $X_i$, the inherited nodes of $i'$. Then
the entry for \( \alpha \) in the table \( J_i \) for \( i \) is the set \( \hat{H}_\psi \) in the entry for \( \psi \) in the table for \( \hat{J}_i \) for \( i' \), with the modification that every function \( h \) from \( X_{i'} \) to \( \mathbb{N} \) in the set \( \hat{H}_\psi \) is extended to be a function \( g \) from \( X_i \) to \( \mathbb{N} \) by setting \( g(y) = 0 \) for every \( y \) in \( X_i - X_{i'} \).

Now suppose that nonleaf node \( i \) of the tree decomposition has two children, say nodes \( i' \) and \( i'' \) of the tree decomposition. The tables \( \hat{J}_i \) and \( J_{i'} \) for tree nodes \( i' \) and \( i'' \) are combined to produce table \( J_i \) for \( i \) as follows. Consider an assignment \( \alpha \) to the states of the explicit nodes of \( i \). Let \( \psi' \) and \( \psi'' \) denote the projection of assignment \( \alpha \) onto the inherited nodes of \( i' \) and \( i'' \) respectively. Let every function \( h' \) in \( \hat{H}_{\psi'} \) for node \( i' \) be extended to a function \( g' : X_i \rightarrow \mathbb{N} \), and every function \( h'' \) in \( \hat{H}_{\psi''} \) for node \( i'' \) be extended to a function \( g'' : X_i \rightarrow \mathbb{N} \); both extensions are done as described above. Define the function \( g' + g'' : X_i \rightarrow \mathbb{N} \) as follows: \((g' + g'')(y) = g'(y) + g''(y)\). Thus, the entry for \( \alpha \) 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 \( \hat{H}_{\psi'} \) from \( \hat{J}_i \) and \( h'' \) in \( \hat{H}_{\psi''} \) from \( \hat{J}_i'' \).

Let \( r \) be the root node of the tree decomposition. The root node has no inherited nodes, so \( \hat{X}_r = \emptyset \). Consequently, table \( \hat{J}_r \) for the root node \( r \) consists of a single entry. This entry is for the empty assignment, which we denote as \( \emptyset \). This entry \( \hat{H}_{\emptyset} \) contains at most one function. (If \( \hat{H}_{\emptyset} \) is nonempty, it contains the unique function with the empty domain.) Set \( \hat{H}_C \) is nonempty if and only if there exists an assignment to the states of all the nodes of SDS \( S \) that is strongly viable for \( r \), that is, if and only if configuration \( C \) has a predecessor.

Now we consider the time complexity of the above algorithm. Each table \( J_i \) and \( \hat{J}_i \) has at most \( 2^{k+1} \) entries (corresponding to at most \( 2^{k+1} \) assignments \( \alpha \)). Each of these entries consists of a set \( H_\alpha \), containing at most \((d+1)^{k+1} \) values of \( h_\beta \), each of which has at most \( 2^{k+1} \) entries. Since the treewidth \( k \) is bounded, for any given value of \( k \), the size of each table \( J_i \) and \( \hat{J}_i \) is polynomial in \( n \), the number of nodes in \( G \). As mentioned earlier, the number of nodes in tree \( T \) is linear in \( n \). For each node \( i \) of \( T \), computing \( \hat{J}_i \) from \( J_i \) can be done in polynomial time. Also, computing the table \( J_i \) for each node \( i \) from the tables of its children (if any) in \( T \) can be done in polynomial time. Thus, the algorithm runs in polynomial time.

We now briefly indicate the necessary modifications to the above algorithm to obtain a polynomial time algorithm for the #PRE problem for (BOOL, SYM, TW-BOUNDED)-SDSs. Once we have a polynomial time algorithm for the #PRE problem, it is clear that APRE and UPRE problems for such SDSs can also be solved in polynomial time.

Our bottom-up dynamic programming algorithm for the #PRE problem also maintains a table \( J_i \) at each node \( i \) of the tree decomposition. Each row of \( J_i \) corresponds to an assignment \( \alpha \) of values to the explicit nodes of \( i \). In solving the PRE problem, the table entry corresponding to assignment \( \alpha \) is the set \( H_\alpha \) of functions, where each function \( h \in H_\alpha \) characterizes a set of assignments to the hidden nodes of \( i \). To solve the #PRE problem, the entry \( H_\alpha \) of table \( J_i \) corresponding to assignment \( \alpha \) is collection of ordered pairs \((h, v)\), where \( h \) is as before and \( v \in \mathbb{N} \) is the number of assignments to the hidden nodes of \( i \) which are characterized by the function \( h \). With this choice, the bottom-up dynamic programming proceeds as before, except that suitable arithmetic operations are used to compute the \( v \) values at node \( i \) from the \( v \) values of the children of \( i \). We note that \( n \) bits are adequate to store each \( v \) value. So, the arithmetic operations needed to compute the \( v \) values as the algorithm proceeds up the tree can be carried out in polynomial time.

### 4.2.1. Extensions of the algorithm

We now generalize Theorem 4.3 to wider classes of Boolean functions. We first consider \( r \)-symmetric functions. Note that every symmetric function is 1-symmetric. Also, if the maximum node degree of the underlying graph of an SDS is \( d \), then the SDS is \((d + 1)\)-symmetric.

For the PRE problem, we generalize the algorithm given in the proof of Theorem 4.3 to \( r \)-symmetric functions as follows. (In the following, we use some of the definitions and notation introduced in the proof of Theorem 4.3.) For a given node \( y \) of \( G \), let \( I_y \) denote the set of at most \( r \) classes of inputs to the \( r \)-symmetric local transition function for \( y \). For a given node \( i \) of the tree decomposition, let \( Z_i \) be a bag (i.e., multiset) containing an element for each class of each node \( y \) in \( I_y \). Thus, if a given node \( y \in I_y \) contains \( q \) classes \( v^1_y, v^2_y, \ldots, v^q_y \), then bag \( Z_i \) contains \( q \) corresponding elements, which can be denoted by \((y, 1), (y, 2), \ldots, (y, q)\). For a given assignment \( \beta \) to the states of the hidden nodes of \( i \), we define a function \( h_\beta : Z_i \rightarrow \mathbb{N} \) as follows. For \( y \in X_i \) and class \( v^k_y \) for node \( y \),

\[
h_\beta((y, z)) = \text{number of hidden nodes } w \text{ of } i \text{ such that } \{w, y\} \in E, w \text{ is a member of class } z \text{ of node } y, y \text{ precedes } w \text{ in } \pi, \text{ and } \beta(w) = 1.
\]
We then define $J_i$ and $\hat{J}_i$ for node $i$ as the corresponding generalization. Each function $h_\beta$ can be represented as a table with at most $r(k + 1)$ entries, where each entry is an integer in the range $0$ through $d$. Since $r$ and $k$ are fixed, it can be seen that the bottom-up dynamic programming algorithm runs in polynomial time.

Next, we sketch the extension of the algorithm to handle $(g, d)$-symmetric SDSs, where $g(n)$ is a polynomial in $n$. For each node $i$ of the tree decomposition, the number of table entries is the same as before. The difference is that each value in the table $h_\beta$ is a sum of weights, rather than a count, and so has a maximum value of $(d + 1)g(n)$. Thus, we can use the dynamic programming approach as before and obtain the following corollary.

**Corollary 4.4.** The PRE, APRE, #PRE and UPRE problems for (BOOL, (g, d)-SYM, TW-BOUNDED)-SDSs (SyDSs) are in $P$, for any function $g(n)$ which is a polynomial in $n$, the number of nodes of the underlying graph. $\blacksquare$

Consider the class of SDSs (SyDSs) over the Boolean domain where the underlying graph has bounded treewidth and bounded degree. If the maximum node degree is $d$ (a constant), each local transition function is trivially $(d + 1)$-symmetric. Further, for such SDSs (SyDSs), each local transition function which is a weighted threshold function can be efficiently replaced by an equivalent Boolean function without edge weights; this is because the number of Boolean functions with at most $d + 1$ inputs is bounded when $d$ is bounded. Therefore, for such SDSs (SyDSs), the PRE problem and its variants can be solved in polynomial time with no restrictions on local transition functions. This is stated in the following corollary.

**Corollary 4.5.** The PRE, APRE, #PRE and UPRE problems for (BOOL, NONE, DEG-TW-BOUNDED)-SDSs (SyDSs) are in $P$. $\blacksquare$

Note that Theorem 4.3 holds for underlying graphs of unbounded degree when the local transition functions are symmetric. In contrast, Corollary 4.5 holds for arbitrary local transition functions as long as each function has only a constant number of inputs.

It can also be seen that all the polynomial time results indicated above hold when the underlying domain is finite (i.e., of fixed size) instead of being Boolean. Note that 1D-CAs with bounded radius correspond to SyDSs over a finite domain with underlying graphs which are both treewidth and degree bounded. Thus, the extension of Corollary 4.5 to finite domains shows that the PRE problem and its variants can be solved in polynomial time for 1D-CAs with bounded radius. Sutner [63] showed that the PRE problem for 1D-CAs is in $P$. No results were known for APRE, #PRE and UPRE problems for 1D-CAs. Corollary 4.5 shows that these problems also admit polynomial time algorithms for 1D-CAs.

**Adding linear constraints.** Finally, we consider another useful extension, namely to handle linear constraints on the state values of the predecessor configuration. As an example, consider the PRE problem for (BOOL, SYM, TW-BOUNDED)-SDSs (SyDSs) with the additional constraint that the state values of a predecessor configuration should sum to at least $\alpha$. The algorithm considered in Theorem 4.3 can be extended to handle any fixed number of such linear constraints. Formally, let $C = (y_1, \ldots, y_n)$ be a configuration. A linear constraint is specified by $\sum_{i=1}^{n} y_i \leq \alpha$ or $\sum_{i=1}^{n} y_i \geq \alpha$, where $\alpha$ is from the same domain over which the symmetric functions are evaluated. To see how such constraints can be handled, we need the following facts, which are easy to verify.

**Lemma 4.6.** 1. A linear constraint can be expressed as a symmetric function.
2. Let $f_1$ and $f_2$ be linear constraints over the same set of variables. Then there is a single symmetric function $F$ that expresses the two linear constraints simultaneously. $\blacksquare$

In view of Lemma 4.6, we will assume that we are given set of $k$ (a constant) symmetric functions each over different subset of variables. These symmetric functions model the constraints on the states of the individual nodes in the predecessor configuration. Our solution consists of a polynomial time structure preserving transformation from the predecessor existence problem with linear constraints to the problem of solving the predecessor existence problem for a new SDS (SyDS) without any linear constraints. The new SDS (SyDS) has additional nodes that will be used to encode these constraints implicitly. We note that the algorithm given in Theorem 4.3 can be modified to account for the linear constraints; we have chosen this method since it illustrates the use of local replacement based SIMULTANEOUS reductions for obtaining easiness results.

**Theorem 4.7.** The PRE, APRE, #PRE and UPRE problems subject to a fixed set of linear constraints for (BOOL, SYM, TW-BOUNDED)-SDSs (SyDSs) are in $P$. 

Proof. We will outline our extension for SyDSs. The extension for SDS is similar and is omitted.

Given our original SyDS $S$, and a set $\{f_1, \ldots, f_k\}$ of symmetric functions encoding the $k$ constraints (one for each distinct subset of nodes), we will construct a new SyDS $S_1$ without any explicit linear constraints, as follows.

1. **Graph:** $S_1$ contains $k$ additional nodes $v_{n+1}, \ldots, v_{n+k}$, one for each symmetric function $f_i$. If $f_i$ is a symmetric function of variables $x_1, \ldots, x_p$, then the node associated with $f_i$ is adjacent to nodes representing $x_1, \ldots, x_p$. For each node $v_{n+i}$, there is a "mate" node $m_i$, that is adjacent only to $v_{n+i}$.

2. **Local functions.** The local function at node $v_{n+i}$ associated with constraint $f_i$ computes the symmetric function $g_{v_{n+i}}$ (defined below). The new local function at an original node $v$ computes the same symmetric function, except that it takes $p_v$ extra inputs that denote the $p_v$ constraints in which the value of $v$ appears. Each mate node computes the simple 1-threshold function. We define these local functions formally below. In describing these functions, we use $f(j)$ to denote the value of a function $f$ when $j$ of its inputs are 1.

$$
\forall i, 1 \leq i \leq k, g_{v_{n+i}}(j) = 1 \quad \text{if} \quad f_i(j) = 1 \\
= 0 \quad \text{otherwise.}
$$

$$
\forall i, 1 \leq i \leq n, g_v(j) = 1 \quad \text{if} \quad f_v(j) = 1 \\
= 0 \quad \text{otherwise.}
$$

$$
\forall i, 1 \leq i \leq k, g_{m_i}(j) = 1 \quad \text{if} \quad (v_{n+i} \lor m_i) = 1 \\
= 0 \quad \text{otherwise.}
$$

3. **Final configurations.** Let the required final configuration for $S$ be $C_S = (\alpha_1, \ldots, \alpha_n)$. Then the required final configuration $C_{S_1}$ for $S_1$ is given by $(\alpha_1, \ldots, \alpha_n, \alpha_{n1}, \beta_1, \ldots, \alpha_{nk}, \beta_k)$, where $\alpha_{ni} = 1$ denotes the value at the node $v_{n+i}$ corresponding to the symmetric constraint $f_i$ and $\beta_i = 0$ corresponds to the value at the mate node $m_i$.

Given a SyDS $S$, let $tw(S)$ denote the treewidth of the graph associated with $S$. The following lemma (which can be proven in a straightforward manner) shows that the transformation described above yields a new SyDS whose treewidth is larger than that of $S$ only by an additive constant.

**Lemma 4.8.** $tw(S_1) \leq tw(S) + k$ and a tree decomposition of the graph associated with $S_1$ can be obtained in linear time, given a tree decomposition of $S$. $\square$

We now argue that a solution to the PRE problem for $S_1$ allows us to obtain a solution to the PRE problem with linear constraints for $S$. To see this, let $C'_{S_1} = (\alpha'_1, \ldots, \alpha'_n, \alpha'_{n+1}, \beta'_1, \ldots, \alpha'_{nk}, \beta'_k)$ be a predecessor of $C_{S_1}$ = $(\alpha_1, \ldots, \alpha_n, \alpha_{n1}, \beta_1, \ldots, \alpha_{nk}, \beta_k)$. We claim that $C'_S = (\alpha'_1, \ldots, \alpha'_n)$ is the predecessor of $C_S = (\alpha_1, \ldots, \alpha_n)$ that satisfies the constraints $f_i, 1 \leq i \leq k$.

By our construction, we have

$$
\forall i \leq i \leq k (\beta_i = 0) \implies (\beta'_i = 0) \land (\alpha'_{n+i} = 0).
$$

Thus $\forall i \leq i \leq k (\alpha_{n+i} = 1)$ implies that the node $v_{n+i}$ evaluates to 1. In turn, this implies, by the definition of the local transition function at $v_{n+i}$ and the fact that $\alpha'_{n+i} = 0$ from above, that the constraint $f_i$ is satisfied. The remaining details are easy to verify. We also note that the reduction preserves the number of solutions and hence $\#\text{PRE}$ and PRE problems can also be solved using this transformation.

We now briefly discuss how to handle the APRE problem. The basic idea is to use the above transformation, but with some important modifications. Given a predecessor $C''$ of $C$, one can encode the condition that $C''$ is different from $C'$ by an $n$-fold disjunction saying that the two $n$ tuples differ in at least one place. Such a clause is unfortunately not a symmetric function since it can have positive and negative literals. But this difficulty can be handled as follows. We construct two nodes NL and UNL; NL computes the simple 1-threshold function for the negated literals and UNL computes the simple 1-threshold function for unnegated literals. These two nodes are joined to the original node set as though they encode two distinct linear constraints. We then have another node $R$ that also computes a simple 1-threshold function of the node values at NL and UNL. The remaining details are straightforward and thus omitted. $\blacksquare$
5. Results for systems with threshold functions

5.1. Results for simple threshold functions

In this section, we consider SDSs and SyDSs in which local transition functions are simple threshold functions. The main result of this section is that the variants of the \textsc{PRE} problem are complete for the appropriate complexity classes even when all the nodes compute the same \textit{k}-simple-threshold function, for any \textit{k} \geq 2.

For SDSs and SyDSs in which each local transition function is the \textit{0}-simple-threshold function, it can be verified that the only configuration that has a predecessor is the one in which all nodes have the state value 1. Obviously, the number of predecessors of that configuration is \(2^n\), where \(n\) is the number of nodes in the underlying graph. The following result concerns SDSs in which each node computes the \textit{1}-simple-threshold function (i.e., the OR function). Note that (\textsc{Bool}, \textit{1-Threshold}, \textsc{None})-SDSs (SyDSs) are the same as (\textsc{Bool}, \textsc{OR}, \textsc{None})-SDSs (SyDSs).

**Proposition 5.1.** The \textsc{PRE} and \textsc{APRE} problems are in \textsc{P} for (\textsc{Bool}, \textit{1-Threshold}, \textsc{None})-SDSs (SyDSs). In contrast, there is a parsimonious planarity preserving reduction from \textsc{Planar Monotone 2SAT} to \textsc{PRE} restricted to (\textsc{Bool}, \textit{1-Threshold}, \textsc{Planar})-SDSs (SyDSs). Hence, \#\textsc{PRE} is \#\textsc{P}-complete, and \#\textsc{APRE} is \textsc{D}^\textsc{P}-complete, even when restricted to (\textsc{Bool}, \textit{1-Threshold}, \textsc{Planar})-SDSs (SyDSs).

**Proof.** We first consider the \textsc{PRE} problem. Let a (\textsc{Bool}, \textit{1-Threshold}, \textsc{None})-SDS \(S\) and a final configuration \(C\) be given. Construct a configuration \(C'\) as follows. For each node \(x\), if there exists a node \(y\) such that \(x\) precedes \(y\) in the permutation, \(x\) is a neighbor of \(y\) or \(x\) and \(y\) are the same node, and \(C(y) = 0\), then set \(C'(x) = 0\). Otherwise, set \(C'(x) = 1\). It can be seen that the answer to the \textsc{PRE} problem is yes if and only if the SDS has a transition from \(C\) to \(C'\). The construction of \(C'\), and determination of the next configuration of the SDS after \(C'\) can be done in time linear in the size of the graph.

Next we consider \textsc{APRE} problem for SyDSs. The proof of SDS follows along similar lines. Let \(C\) be the final configuration and \(C''\) be a given predecessor of \(C\). Our goal is to find a configuration \(C'\) that is a predecessor of \(C\) but differs from \(C''\) in at least one position. For each node \(x\) such that \(C(x) = 0\), we must have that \(\forall y \in N(x) \cup \{x\}, C'(y) = 0\). In the algorithm for \textsc{PRE}, we assigned 1 to the remaining nodes. If the resulting configuration differs from \(C''\) then we are done; otherwise, find a node (if it exists) that is not the only remaining neighbor of the nodes with final value 1. If such a node does not exist, the answer to the \textsc{APRE} problem is NO; otherwise (i.e., such a node exists), simply set the value of this node to 0 and all other nodes to value 1 as before.

To show that the counting version is \#\textsc{P}-hard, we will present a parsimonious reduction from a known \#\textsc{P}-complete problem, namely the problem of counting the number of satisfying assignments to a planar monotone 2CNF formula [68].

Given a planar monotone 2CNF formula, consisting of the set \(X = \{x_1, x_2, \ldots, x_n\}\) of variables and the set \(C = \{c_1, c_2, \ldots, c_m\}\) of clauses, we create the following SDS \(S\). For each variable \(x_i\), the underlying graph \(G(V, E)\) of the SDS has one node, also denoted by \(x_i\) (1 \(\leq i \leq n\)). For each clause \(c_j\), there are two nodes, denoted by \(c_j\) and \(d_j\), along with the edge \([c_j, d_j]\) (1 \(\leq j \leq m\)). Further, there are two edges joining node \(c_j\) to the nodes corresponding to the two variables whose literals appear in clause \(c_j\) (1 \(\leq j \leq m\)). The local transition function at each node is the \textsc{OR} function. For the SDS case, the permutation \(\pi\) is the following:

\[
\pi = \langle d_1, d_2, \ldots, d_m, c_1, c_2, \ldots, c_m, x_1, x_2, \ldots, x_n \rangle.
\]

The final configuration \(C\) is chosen as follows: \(\forall j \ C(d_j) = 0\) and all other nodes are set to 1. It is easy to observe that each predecessor \(C'\) must satisfy the following conditions: \(\forall j \ C'(d_j) = C'(c_j) = 0\). Using this observation, it is straightforward to verify that there is a one-to-one correspondence between the number of predecessors and the number of satisfying assignments to the given planar monotone 2CNF formula.

By very similar arguments, we can show that \textsc{PRE} problems for (\textsc{Bool}, \textsc{AND}, \textsc{None})-SDSs (SyDSs), (\textsc{Bool}, \textsc{NAND}, \textsc{None})-SDSs (SyDSs), and (\textsc{Bool}, \textsc{NOR}, \textsc{None})-SDSs (SyDSs) are in \textsc{P}. Note that (\textsc{Bool}, \textsc{AND}, \textsc{None})-SDSs (SyDSs) are a special class of (\textsc{Bool}, \textsc{Threshold}, \textsc{None})-SDSs (SyDSs) wherein each node \(v\) computes the simple \textit{k}-threshold function, with \(k = 1 + \text{degree}(v)\). The following theorem shows that the above results are essentially tight.
Theorem 5.2. There is a local replacement based planarity preserving (decision, parsimonious)-reduction from 3SAT to Pre problem for (BOOL, {AND, OR}, NONE)-SDSs (SySDSs). Thus Pre is NP-complete, #Pre is #P-complete, APRE is NP-complete and UPRE is D^P-complete for (BOOL, {AND, OR}, PLANAR)-SDSs (SySDSs).

Proof. We present a reduction from GOLD’S MONOTONE 3SAT, which is the restricted version of SAT in which each clause contains exactly three unnegated literals or exactly three negated literals. In [29], we proved a local replacement based planarity preserving (decision, parsimonious)-reduction from 3SAT to GOLD’S MONOTONE 3SAT. This observation in conjunction with the following reduction will establish the above theorem.

The reduction from GOLD’S MONOTONE 3SAT leads to the graph G(V, E), where V = {x1, x2, ..., xn, c1, c2, ..., cm, a, b, d}. The edges in E are as follows.

1. For each i, 1 ≤ i ≤ n, the edge {a, xi}.
2. For each i, j, 1 ≤ i ≤ n, 1 ≤ j ≤ m, the edge {xi, cj} whenever the literal xi or ¬xi appears in clause cj.
3. For each j, 1 ≤ j ≤ m, the edge {b, cj} if cj has all positive literals.
4. For each j, 1 ≤ j ≤ m, the edge {d, cj} if cj has all negative literals.

The node functions are as follows. For a, b, each node xi and each clause cj with only positive literals, the function is OR; for d and each clause cj with only negative literals, the function is AND. The permutation π is (a, b, d, c1, c2, ..., cm, x1, x2, ..., xn). The required final configuration C has C(a) = 1, C(b) = 0, C(d) = 1, C(cj) = 1 if cj has all positive literals, C(cj) = 0 if cj has all negative literals and C(xi) = 1, for 1 ≤ i ≤ n.

We now briefly indicate why the Pre instance has a solution if and only if the instance of GOLD’S MONOTONE 3SAT has a solution. The initial value C(b) = 0 forces the initial value of each clause containing only positive literals to be 0. The initial value 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. The parsimoniousness of the reduction is straightforward to verify.

Theorem 5.3. There is a local replacement based (decision, parsimonious)-reduction from 3SAT to Pre problem for (BOOL, THRESH, NONE)-SDSs and (BOOL, THRESH, NONE)-SySDSs, where each node computes the same k-simple-threshold function, for each k ≥ 2. Thus Pre is NP-complete, #Pre is #P-complete, APRE is NP-complete and UPRE is D^P-complete for (BOOL, THRESH, NONE)-SDSs (SySDSs).

Proof. Given an instance of 3SAT, consisting of the set X = {x1, x2, ..., xn} of variables and the set C = {c1, c2, ..., cm} of clauses, we create the following SDS S. First, the underlying graph G(V, E) has the following vertices and edges. V = V1 ∪ V2 ∪ V3, where V1 = {a^0, a^1, ..., a^{k-1}}, V2 = {xi, ¬xi, yi, zi | for each variable xi} and V3 = {c_j, d_j | for each clause c_j}. The set E has the following edges.

1. For each i, 1 ≤ i ≤ k, the node set {a^0_i, a^1_i, ..., a^{k-1}_i} is connected together as a k-clique.
2. ∀ i, 1 ≤ i ≤ n, edges {xi, yi}, {xi, zi} and {xi, ¬xi}.
3. ∀ j, 1 ≤ j ≤ m, edge {cj, d_j}, and an edge from cj to the node for each of the three literals occurring in clause cj.
4. ∀ p, i, 1 ≤ p ≤ k − 2 and 1 ≤ i ≤ n, the edges {a^p_i, yi} and {a^p_i, zi}.
5. ∀ p, i, 1 ≤ p ≤ k and 1 ≤ i ≤ n, edges {a^0_p, xi} and {a^0_p, ¬xi}.
6. ∀ p, i, 1 ≤ p ≤ k − 1 and 1 ≤ j ≤ m, edges {a^p_p, d_j} and {a^p_p, cj}.

Fig. 7 shows an example of the construction for k = 3. The figure shows the subgraph for each variable xi (box labelled I), the subgraph for each clause cj (box labelled II) and the edges between these subgraphs and the control subgraph (box labelled III). (To minimize clutter, edges between each clause node and the nodes corresponding to the literals in the clause are not shown in the figure.)

The local transition function at each node is the k-simple-threshold function. For the SDS case, the permutation π is given by

\[ π = (a^1_1, ..., a^{k-1}_1, a^0_1, ..., a^1_k, ..., a^{k-1}_k, a^0_k, d_1, ..., d_m, c_1, ..., c_m, y_1, ..., y_n, \]
\[ z_1, ..., z_n, x_1, ..., x_n, \neg x_1, ..., \neg x_n). \]
The required final configuration $C$ is given as follows: $\forall i, 1 \leq i \leq n, C(y_i) = 0$ and $\forall j, 1 \leq j \leq m, C(d_j) = 0$. For any other node $v$, $C(v) = 1$.

We now argue that there is a solution to the PRE instance if and only if there is a solution to the 3SAT instance. The argument holds for both the SDS and SyDS cases. Given a satisfying assignment to the 3SAT instance, a configuration $C'$ which is a predecessor of $C$ can be obtained as follows. Set $C'(a^p q^q) = 1$ (for each $p$ and $q$), $C'(c_j) = C'(d_j) = 0$ (for each $j$), $C'(y_i) = 0$ and $C'(z_i) = 1$ (for each $i$), and $C'(x_i)$ and $C'(\overline{x}_i)$ to the values given by the satisfying assignment to the 3SAT instance. It can be verified that $C'$ is a predecessor of $C$.

For the converse, suppose $C'$ is a predecessor. It can be seen that the following are true for any valid $C'$: $\forall p, q, C'(a^p q^q) = 1; \forall j, C'(d_j) = 0$ and $C'(c_j) = 0$. Moreover, for each $i$, the required final values of $y_i$ and $z_i$ ensure that $C'(x_i)$ and $C'(\overline{x}_i)$ are complementary values. Using the initial and final values of $c_j$ and the fact that each node computes the $k$-simple-threshold function it can be seen that $\forall i, C'(x_i)$ represents a satisfying assignment to the 3SAT instance. Furthermore, since the initial values of all nodes except $x_i$ and $\overline{x}_i$ are determined by the construction, it is easy to verify that there is a one-to-one correspondence between the set of satisfying assignments to the 3SAT instance and the set of predecessors. ■

**Remark.** For $k = 2, 3$, the reduction presented in the proof of Theorem 5.3 can be made planarity preserving by using a separate control subgraph for each variable and clause component (boxes I and II in Fig. 7). Hence, for these values of $k$, the theorem holds even when restricted to SDSs (SyDSs) whose underlying graphs are planar.

### 5.2. Predecessor existence with weighted threshold functions

Here we consider the PRE problem where each edge of an SDS (or SyDS) has a weight (positive, negative or zero) and each local transition function is a weighted threshold function. Recall from Section 2.5 that a Boolean weighted threshold function $f$ with threshold $\alpha$ has $n$ Boolean inputs denoted by $q_1, q_2, \ldots, q_n$, where each input $q_i$ has an associated weight $w_i, 1 \leq i \leq n$. The value of $f$ is $1$ if $\sum_{i=1}^{n} q_i w_i \geq \alpha$ and $0$ otherwise.

In considering SDSs in which each local transition function is a weighted threshold function, we assume that there is a self-loop around each node, and that the edge corresponding to the self-loop is also given a weight. Due to the local nature of these self-loops, they are not considered in specifying the structure of the underlying graph. Thus, a tree in this context is a normal tree with a self-loop for each node.

We use $(BOOL, WTHRESH)$-SDS ($(BOOL, WTHRESH)$-SyDS) to denote the class of SDSs (SyDSs) in which each local transition function is a weighted threshold function. The main results shown in this section are as follows.

1. The PRE problem can be solved in polynomial time for $(BOOL, WTHRESH, TREE)$-SDSs (SyDSs).
2. In contrast, the PRE problem is NP-complete for $(BOOL, WTHRESH, TW-BOUNDED)$-SDSs (SyDSs), even when the underlying graph has a treewidth of two.
We also show that #PRE problem is #P-complete for (BOOL, WTHRESH, TREE)-SDSs (SyDSs). Thus, for this class of SDSs (SyDSs), the decision and counting versions of the PRE problem behave differently with respect to complexity.

5.2.1. A polynomial time algorithm for PRE for trees

In this section, we develop a dynamic programming algorithm for the PRE problem for (BOOL, WTHRESH, TREE)-SDSs (SyDSs). In fact, our algorithm works for a class of local transition functions that are more general than weighted threshold functions. A definition of this class of functions is given below.

**Definition 5.4.** Suppose $x$ is an input to a Boolean function $f$ and $\Gamma$ is the set of all the other inputs to $f$. The function $f$ is **positive monotone** with respect to $x$, if for any combination $\mathcal{T}(\Gamma)$ of values for the inputs in $\Gamma$, $f(\mathcal{T}(\Gamma), x = 0) \leq f(\mathcal{T}(\Gamma), x = 1)$. The function $f$ is **negative monotone** with respect to $x$, if for any combination $\mathcal{T}(\Gamma)$ of values for the inputs in $\Gamma$, $f(\mathcal{T}(\Gamma), x = 0) \geq f(\mathcal{T}(\Gamma), x = 1)$.

When a local transition function $f_v$ at node $v$ is a weighted threshold function, it can be seen that $f_v$ is positive (negative) monotone with respect to input $x$ if the weight of the edge $(x, v)$ is positive (negative). Thus, positive and negative monotone functions are more general than weighted threshold functions. We show that the PRE problem can be solved in polynomial time for SDSs and SyDSs when the underlying graph is a tree and each local transition function is positive or negative monotone with respect to each input. Thus, our result holds for (BOOL, WTHRESH, TREE)-SDSs (SyDSs) even when the edge weights are negative. In contrast, it is shown in Section 5.2.2 that for graphs of treewidth 2, the PRE problem is NP-complete even when all the edge weights are nonnegative.

For simplicity, we will present our algorithm for the decision version of the PRE problem. It is straightforward to modify the algorithm to find a predecessor configuration, when such a configuration exists. We will present the details of the algorithm for an SDS and then indicate the necessary modifications for a SyDS.

To develop the bottom-up dynamic programming algorithm, we introduce the following notation for each node $v$ of the tree. If $C$ denotes the given final configuration, we use $C(v)$ to denote the value of node $v$ in $C$. We assume that we know, for each input, whether the local transition function $f_v$ at node $v$ is positive or negative monotone with respect to that input. Also, $p[v]$ denotes the parent of $v$ and $A_v$ denotes the subtree rooted at $v$. The **proper descendants** of $v$, denoted by $PD(v)$, are all the nodes in $A_v$, except $v$ itself. Let $R$ denote the root of the tree. At each node $v$, the dynamic programming algorithm maintains an array $B_v$ of four binary values. (For convenience, we use $(0, 0), (0, 1), (1, 0)$ and $(1, 1)$ as the indices of the four entries of $B_v$.) The significance of the entries in $B_v$ is given below.

For $(x, y) \in \{0, 1\} \times \{0, 1\}$, $B_v[x, y] = 1$ if and only if there is an assignment of initial values to the nodes in $PD(v)$ such that the assignment together with the assignment of Boolean values $x$ to $v$ and $y$ to $p[v]$ results in the correct value in the final configuration for each node in $A_v$.

**Note:** Since the root $R$ does not have a parent, the array $B_R$ at $R$ stores only two binary values, denoted by $B_R[0]$ and $B_R[1]$.

**Step 0 — Initialization of the tables at leaf nodes:** For each leaf $v$, the table $B_v$ can be computed as follows. (Since the degree of $v$ is 1, the corresponding local transition function $f_v$ has only two inputs.) For any pair $(x, y)$ of Boolean values, where $x$ corresponds to the value of $v$ and $y$ corresponds to the value of $p[v]$, the value $B_v[x, y]$ is given by

$$B_v[x, y] = (f_v(x, y) \equiv C(v)) \text{ if } v \text{ precedes } p[v] \text{ in the permutation and }$$

$$= (f_v(x, C(p[v])) \equiv C(v)) \text{ if } v \text{ follows } p[v] \text{ in the permutation.}$$

**Note:** The expression $(f_v(x, y) \equiv C(v))$ is 1 (true) if the Boolean values $f_v(x, y)$ and $C(v)$ are equal and 0 (false) otherwise.

**Step 1 — Computation of the table at a nonleaf node:** Now consider the computation of the entries in the table for a nonleaf node $v$. Let $w_1, w_2, \ldots, w_k$ denote the $k$ children of $v$. Assume that the table for each of the children has already been computed. The computation of the entries in $B_v$ is done in two stages.

**Step 1.1 — Preliminary checks:** These checks determine whether a node $w$ can force a value for some of the entries in its parent’s table.

1. If there is a child $w_j$ of $v$ such that $B_{w_j}[0, 0] = 0$ and $B_{w_j}[1, 0] = 0$, then set $B_v[0, 0] = 0$ and $B_v[0, 1] = 0$. 
2. If there is a child \( w_j \) of \( v \) such that \( B_{w_j}[0, 1] = 0 \) and \( B_{w_j}[1, 1] = 0 \), then set \( B_v[1, 0] = 0 \) and \( B_v[1, 1] = 0 \).

**Step 1.2 — Subsequent computation:** The goal of Steps 1.2 and 1.3 is to find the value for \( B_v[x, y] \) for combinations of \((x, y)\) values which were not determined in Step 1.1 above. The following computation is carried out for each child \( w_j \) of \( v \). Note that since Step 1.1 does not apply, at least one of \( B_{w_j}[0, x] \) and \( B_{w_j}[1, x] \) has the value 1. The computation identifies a Boolean value \( h_j \) for each child \( w_j \) of \( v \). If \( v \) follows \( w_j \) in the permutation, then this value \( h_j \) must be \( C(w_j) \). However, suppose \( v \) precedes \( w_j \) in the permutation. If \( B_{w_j}[0, x] = 1 \), then \( h_j \) can be 0, and if \( B_{w_j}[1, x] = 1 \), then \( h_j \) can be 1. If there is only one choice for \( h_j \), then Steps 1.2(a) and 1.2(b) choose that value for \( h_j \). If two choices are possible for \( h_j \), then Steps 1.2(a) and 1.2(b) select the value based on \( C(v) \) and the monotonicity of \( f_v \) with respect to \( w_j \).

This value \( h_j \) will be used in evaluating the local transition function \( f_v \) at \( v \), and the value of \( f_v \) will be used in Step 1.3 below to compute the table entries for \( B_v \). (For each child \( w_j \) of \( v \), exactly one of the steps (a), (b) and (c) described below will be carried out.)

(a) This step is done if \( v \) precedes \( w_j \) in the permutation and \( f_v \) is **positive monotone** with respect to \( w_j \).

\[
\text{if } (C(v) = 0) \quad \text{then } h_j = B_{w_j}[0, x] \quad \text{else } h_j = B_{w_j}[1, x].
\]

(b) This step is done if \( v \) precedes \( w_j \) in the permutation and \( f_v \) is **negative monotone** with respect to \( w_j \).

\[
\text{if } (C(v) = 0) \quad \text{then } h_j = B_{w_j}[1, x] \quad \text{else } h_j = B_{w_j}[0, x].
\]

(c) This step is done if \( v \) follows \( w_j \) in the permutation: Set \( h_j = C(w_j) \).

**Step 1.3 — Final computation:** Assume that the value \( h_j \) for each child \( w_j \) of \( v \) has been computed using the computation shown in Step 1.2. Now, \( B_v[x, y] \) (for those combinations of \((x, y)\) values which were not covered by Step 1.1) can be computed as follows.

\[
B_v[x, y] = (f_v(x, h_1, \ldots, h_k, y) = C(v)) \quad \text{if } v \text{ precedes } p[v] \text{ in the permutation and}
\]

\[
= (f_v(x, h_1, \ldots, h_k, C(p[v])) = C(v)) \text{if } v \text{ follows } p[v] \text{ in the permutation.}
\]

**Step 2 — Determining predecessor existence:** The table \( B_R \) at the root \( R \) has only two entries, namely \( B_R[0] \) and \( B_R[1] \). If the value of either of these entries is 1, then the \( \text{PRE} \) problem has a solution; otherwise, there is no solution.

**Modifications for a SyDS:** Having provided the details of the algorithm for an SDS, we indicate the modifications needed for the case of a SyDS.

**Step 0:** For each leaf \( v \) and for any pair \((x, y)\) of Boolean values, the value \( B_v[x, y] \) is given by

\[
B_v[x, y] = (f_v(x, y) = C(v)).
\]

**Step 1:** Step 1.1 remains the same. Parts (a) and (b) of Step 1.2 apply without the conditions regarding the relative positions of \( v \) and \( w_j \) in the permutation. Part (c) of Step 1.2 cannot arise. In Step 1.3, \( B_v[x, y] \) is given by

\[
B_v[x, y] = (f_v(h_1, \ldots, h_k, y) = C(v)).
\]

**Step 2:** Remains the same as that for an SDS.

Note that the above dynamic programming algorithm uses only \( O(n) \) evaluations of local transition functions. Since we assume that each local transition function can be evaluated in polynomial time, it follows that the algorithm runs in polynomial time. The following theorem summarizes the result.

**Theorem 5.5.** The \( \text{PRE} \) problem for \((B, W, \text{TREE})\-SDSs \) (SyDSs) is in \( \mathbb{P} \). The result holds even when each local transition function is **positive monotone** or **negative monotone** with respect to each input.  

We now address the complexity of the \#\text{PRE} problem for \((B, W, \text{TREE})\-SDSs \) (SyDSs). For this, we use the following problem, which is very similar to the \text{SUBSET SUM} problem [25].

**Subset sum with lower bound (SSLB)**

**Instance:** A set \( A = \{a_1, a_2, \ldots, a_n\} \) of \( n \) nonnegative integers and another nonnegative integer \( B \).

**Question:** Is there a subset of \( A \) whose sum is at least \( B \)?

The SSLB problem can be seen to be in \( \mathbb{P} \). However, as shown below, its counting version denoted by \#SSLB, is \#\( \mathbb{P} \)-complete.

**Proposition 5.6.** Problem \#SSLB is \#\( \mathbb{P} \)-complete.
Proof. It is easy to see that SSLB is in \#P. To show \#P-hardness, we use a Turing reduction from the counting version of the SUBSET SUM problem, denoted by \#SS. In \#SS, we are given a set \( A = \{a_1, a_2, \ldots, a_n\} \) of \( n \) nonnegative integers and another nonnegative integer \( B \). The goal is to find the number of subsets of \( A \) whose sum is equal to \( B \). The \#SS problem is known to be \#P-complete.

Suppose the \#SSLB problem is in \( P \). Consider the given set of integers \( A \), and using the assumed polynomial algorithm for \#SSLB, compute the values \( N_B \) and \( N_{B+1} \), which represent respectively the number of subsets of \( A \) whose sum is at least \( B \) and \( B + 1 \) respectively. It can be verified that the solution to the \#SSLB problem is given by \( N_B - N_{B+1} \). Thus, we have a polynomial time algorithm for \#SS, contradicting the assumption that \#SS is \#P-complete. Proposition 5.6 follows.

Our next result shows that the \#PRE problem is \#P-complete for (BOOL, WTHRESH, TREE)-SDSs (SyDSs). For brevity, we prove the result for SDSs; the proof for SyDSs is similar.

**Proposition 5.7.** The \#PRE problem for (BOOL, WTHRESH, TREE)-SDSs is \#P-complete, even when all the edge weights are nonnegative.

Proof. We use a reduction from the #SSLB problem. Given an instance of the #SSLB problem consisting of set \( A = \{a_1, a_2, \ldots, a_n\} \) of \( n \) nonnegative integers and another nonnegative integer \( B \), construct the following (BOOL, WTHRESH, TREE)-SDS \( S \). Except for the self-loop around each node, the underlying graph of \( S \) is a star graph with \( n + 2 \) nodes. The edges join the center node \( v_0 \) (of degree \( n + 1 \)) to each of the leaf nodes \( v_0', v_1, v_2, \ldots, v_n \). The weight of edge \( \{v_0, v_0'\} \) is chosen as 1. The weight of the self-loop around \( v_0' \) is also set to 1. For \( 1 \leq i \leq n \), the weight of edge \( \{v_0, v_i\} \) is chosen as \( a_i \). The weight on the self-loop for each of the nodes \( v_0, v_1, \ldots, v_n \) is 0. The local transition function \( f_0' \) at \( v_0' \) is the weighted threshold function with threshold 1. Let \( f_i \) denote the local transition function at \( v_i, 0 \leq i \leq n \). The local transition function \( f_0 \) at \( v_0 \) is the weighted threshold function with threshold \( B \). For \( 1 \leq i \leq n \), the local transition function \( f_i \) is the weighted threshold function with threshold 0. The node permutation is \( \{v_0', v_0, v_1, v_2, \ldots, v_n\} \) and the final configuration \( C \) has the value 0 for node \( v_0' \) and 1 for every other node. This completes the construction.

We note that the weight of the self-loop around \( v_0' \) is 1, the weight of the edge \( \{v_0', v_0\} \) is 1 and the local transition function at \( v_0' \) has a weighted threshold of 1. Using these facts, it can be seen that initial values of both \( v_0' \) and \( v_0 \) must be 0; otherwise, the final value of \( v_0' \) cannot be 0. Also, since \( v_0' \) precedes \( v_0 \) in the permutation and the final value of \( v_0' \) is 0, the weight of the edge \( \{v_0', v_0\} \) has no influence on the local transition function \( f_0 \) at \( v_0 \). Because of this, it is straightforward to verify that there is a one-to-one correspondence between the set of predecessors of \( C \) and the set of solutions to the \#SSLB problem.

### 5.2.2. Hardness result for graphs of treewidth two

**Theorem 5.8.** There is a local replacement based (decision, parsimonious)-reduction from PARTITION to \#PRE problem for (BOOL, WTHRESH, TW-BOUNDED)-SDSs (SyDSs). Thus \#PRE is NP-complete, \#PRE is \#P-complete, \#PRE is \#P-complete, \#PRE is \#P-complete, \#PRE is \#P-complete and \#PRE is \#P-complete for (BOOL, WTHRESH, TW-BOUNDED)-SDSs (SyDSs) even when the bound on the treewidth of the underlying graph is 2 and all the edge weights are nonnegative.

Proof. The membership of \#PRE in \#NP and that of the counting problem in \#P is obvious. We complete the proof of the theorem by giving a local replacement based parsimonious reduction from PARTITION to \#PRE. Recall that in the PARTITION problem, we are given a collection \( A = \{a_1, a_2, \ldots, a_n\} \) of integers and the question is whether there is a subset \( A' \subseteq A \) such that \( \sum_{a_i \in A'} a_i = \left(\sum_{i=1}^{n} a_i \right)/2 \).

The reduction is the same for both SDSs and SyDSs. Except for the self-loop at each node, the underlying graph is a complete bipartite graph with \( n + 1 \) nodes (denoted by \( v_0, v_1, v_2, \ldots, v_n \)) on one side of the bipartition and two nodes (denoted by \( u_1 \) and \( u_2 \)) on the other side. It can be seen that the underlying graph has a treewidth of 2. The weight of each of the two edges \( \{v_0, u_1\} \) and \( \{v_0, u_2\} \) is set to 1. For each \( v_i, 1 \leq i \leq n \), the weight of each of the two edges \( \{v_i, u_1\} \) and \( \{v_i, u_2\} \) is set to \( a_i \). The weight of the self-loop around \( v_0 \) is set to 1. For every other node, the weight of the self-loop is set to 0. The local transition function \( f_0 \) at \( v_0 \) is a weighted threshold function with threshold \( 1 \). For \( 1 \leq i \leq n \), the local transition function at \( v_i \) is a weighted threshold function with threshold \( 0 \). The local transition functions at \( u_1 \) and \( u_2 \) are weighted threshold functions with thresholds \( (\sum_{i=1}^{n} a_i)/2 \) and \( 1 + (\sum_{i=1}^{n} a_i)/2 \) respectively. The required final configuration has the value 0 for nodes \( u_2 \) and \( v_0 \) and 1 for all other nodes. For the SDS version of the problem, the permutation is chosen as \( \{v_0, u_1, u_2, v_1, \ldots, v_n\} \).
We note that the weight of the self-loop around \( v_0 \) is 1, the weights of the edges \( \{v_0, w_1\} \) and \( \{v_0, w_2\} \) are both 1 and the local transition function at \( v_0 \) has a threshold of 1. Using these facts, it can be seen that initial values of \( v_0 \), \( w_1 \) and \( w_2 \) must be 0; otherwise, the final value of \( v_0 \) cannot be 0. Further, since the initial and final values of \( v_0 \) are 0, the weights of the edges \( \{v_0, w_1\} \) and \( \{v_0, w_2\} \) have no influence on the local transition functions at \( w_1 \) and \( w_2 \).

Using the fact that the required final values of \( w_1 \) and \( w_2 \) are 1 and 0 respectively, it can be verified that the PRE instance has a solution if and only if the PARTITION instance has a solution. Moreover, since the initial values of \( v_0 \), \( w_1 \) and \( w_2 \) must be 0, it is also seen that the above reduction from PARTITION is parsimonious. Thus, the results for #PRE, APRE and UPRE follow from the corresponding results for PARTITION.  

The #P-completeness of #PRE for (BOOL, WTHRESH, TW-BOUNDED)-SDSs also follows from Proposition 5.7. We included this result in the statement of Theorem 5.8 to emphasize the usefulness of simultaneous reductions.

5.2.3. Observations regarding results for weighted threshold functions

We make some observations to highlight the tightness of the results presented above for SDSs (SyDSs) with weighted threshold functions. The results for weighted threshold functions are especially important in the context of discrete neural networks [52,53].

Theorem 5.5 shows that the PRE problem can be solved in polynomial time when the underlying graph is a tree (which has a treewidth of 1). Proposition 5.7 shows that the #PRE problem is #P-complete even when the underlying graph is a tree. In contrast to Theorem 5.5, Theorem 5.8 shows that all variants of the PRE problem are computationally intractable when the underlying graph has a treewidth of 2.

The construction used in the proof of Theorem 5.8 produces two nodes of degree \( n \), while all other nodes have the degree 2. This construction cannot be improved to make all the nodes to be of bounded degree. This is because when both degree and treewidth of the underlying graph are bounded, the PRE problem and its variants are in \( P \) with no restriction on the local transition functions (Corollary 4.5). Also, the reduction used to prove Theorem 5.8 uses edge weights whose values may be exponential in \( n \). The reduction cannot be improved by making all edge weights to be bounded by a polynomial in \( n \). This is because when the treewidth is bounded by a constant and the weights are bounded by a polynomial \( g(n) \), each weighted threshold function is a \((g, d + 1)\)-symmetric function, where \( d \) is the maximum node degree in the underlying graph. Corollary 4.4 points out that for such SDSs (SyDSs), all the variants of the PRE problem can be solved in polynomial time.

6. Towards a dichotomy theorem

Following the set-up for SAT(S) (see Section A.1 in the Appendix A), we define the PRE(S) as the PRE problem for SDSs (SyDSs) in which each local transition function is chosen from the given set \( S \). Throughout this section, we will assume that the underlying graphs for SDSs (SyDSs) are of bounded degree. This immediately implies that the local functions are of finite arity.

Example 6.1. Consider the Boolean domain \( \{0, 1\} \) and suppose the set of relations (i.e., constraints) is given by \( S = \{\text{XOR}(\alpha, \beta, \gamma, \delta), \text{XNOR}(\alpha, \beta, \gamma)\} \). Now consider an SDS in which each node is of degree 2 or 3. Each degree two node executes XNOR, while each degree three node executes XOR.

6.1. Weakly positive/negative and bijunctive relations

Theorem 6.2. There is a local replacement based (decision, parsimonious)-reduction from 3SAT to PRE problem restricted to instances in which the local transition functions are simultaneously bijunctive, weakly positive and weakly negative. Thus, the PRE problem is \( \text{NP}\)-complete. #PRE is \#P-complete, APRE is \#P-complete and UPRE is \( D^{NP} \)-complete for (BOOL, BIJUNCTIVE, PLANAR)-SDSs (SyDSs), (BOOL, WEAKLY-POSITIVE, PLANAR)-SDSs (SyDSs), and (BOOL, WEAKLY-NEGATIVE, PLANAR)-SDSs (SyDSs).

Proof. Given an instance of PLANAR 3SAT, consisting of the set \( X = \{x_1, x_2, \ldots, x_n\} \) of variables and the set \( C = \{c_1, c_2, \ldots, c_m\} \) of clauses, we create the following SDS \( S \). For each variable \( x_i \in X \), \( S \) has one node (denoted by \( x_i \)), 1 ≤ \( i \) ≤ \( n \). For each clause \( c_j \in C \), \( S \) has two nodes (denoted by \( c_j \) and \( c_j' \)), 1 ≤ \( j \) ≤ \( m \). There is an edge between \( c_j \) and \( c_j' \) for each \( j \), 1 ≤ \( j \) ≤ \( m \). Further, if the clause \( c_j \) contains literals corresponding to variables \( x_{i_1} \), \( x_{i_2} \)
and $x_j$, node $c_j$ is joined to the nodes corresponding to the variables. Given that the reduction is from an instance of Planar 3SAT, it can be seen that the resulting graph of the SDS $S$ is planar.

The local transition functions are chosen as follows. For simplicity, we use the name of a node to also denote the state value of the node. (The usage will be clear from the context.) For each variable node $x_i$, $1 \leq i \leq n$, let $c_{i_1}, c_{i_2}, \ldots, c_{i_k}$ be the clauses in which $x_i$ appears. The local transition function at $x_i$ is given by $x_i \land c_{i_1} \cdots \land c_{i_k}$. For each clause node $c_j$, $1 \leq j \leq m$, suppose the literals appearing in clause $c_j$ are $l_{j_1}, l_{j_2}$ and $l_{j_3}$. The local transition function for node $c_j$ is given by $c_j \lor l_{j_1} \lor l_{j_2} \lor l_{j_3}$. The local transition function for each node $c_j$, $1 \leq j \leq m$, is given by $c_j \lor c_j$. Thus, for each node, the local transition function is the AND of two or more literals. Consequently, each local transition function is simultaneously bijective, weakly positive and weakly negative.

The permutation $\pi$ for the SDS $S$ is set as

$$\pi = (c_1', c_2', \ldots, c_m', c_1, c_2, \ldots, c_m, x_1, x_2, \ldots, x_n).$$

The required final configuration $C$ is set as follows: $\forall j \forall C(c_j) = 1$, $\forall j \forall C(c_j) = 0$, and $\forall j \exists C(x_i) = 0$.

Suppose there is a satisfying assignment to the Planar 3SAT instance. We claim that the configuration $C'$ obtained by setting each node $x_i$ to the value given by the satisfying assignment and setting the other nodes to 1 is a predecessor of $C$. To see this, first consider each node $c_j'$, $1 \leq j \leq m$. The two inputs to the local transition function at $c_j'$ are both 1; thus, the value of the function is 1. Now consider each node $c_j$, $1 \leq j \leq m$. Three of the inputs to the local transition function at $c_j$ are the complements of the literals $l_{j_1}, l_{j_2}$ and $l_{j_3}$. Since we have a satisfying assignment, the complement of at least one of these literals has the value 0. Thus, the local transition function at $c_j$ evaluates to 0, as required. Finally, consider each node $x_i$, $1 \leq i \leq n$. For each such node, there is at least one clause node $c_j$ such that $x_i$ is adjacent to $c_j$. Since the final value of $c_j$ is 0 and the local transition function at $x_i$ is the AND function, the local transition function at $x_i$ evaluates to 0. In other words, the SDS reaches the configuration $C$ from $C'$ in one step.

Suppose there is a configuration $C'$ such that the SDS reaches $C$ from $C'$ in one step. We claim that the truth assignment that sets each variable $x_i$ to the Boolean value $C'(x_i)$ is a satisfying assignment. To see this, first observe that

$$\forall j, 1 \leq j \leq m, C(c_j') = 1 \implies (C'(c_j) = 1) \land (C'(c_j) = 1).$$

Now by noting that the local transition function at $c_j$ is an AND function

$$((C(c_j) = 0) \land (C'(c_j) = 1) \land (C(c_j) = 1)) \implies (l_{j_1} \lor l_{j_2} \lor l_{j_3} = 1).$$

Thus, clause $c_j$ is satisfied by the chosen assignment. This completes the proof of NP-hardness. It can easily be verified that there is a one-to-one correspondence between the set of satisfying assignments to the Planar 3SAT instance and the set of predecessors for the configuration $C$.

The proof for SyDSs uses the same reduction as above except that for each variable node $x_i$, $1 \leq i \leq n$, the local transition function is given by $x_i \land c_{i_1} \land \cdots \land c_{i_k}$. This ensures that the final value of node $x_i$ is 0 regardless of its initial value since as argued above, the initial value of each node $c_j$ must be chosen as $1$. Even with this modification, it can be seen that the construction is parsimonious.

### 6.2. 0-valid and 1-valid relations

The remark at the end of Section 5 pointed out that the PRE problem and its variants are hard for (BOOL, THRESH, PLANAR)-SDSs (SyDSs), even when each node computes the same local transition function. Since for any $k \geq 1$, each simple $k$-threshold function is 1-valid, it follows that the problems remain hard for (BOOL, 1-VALID, PLANAR)-SDSs (SyDSs).

The following theorem shows that the PRE problem and its variants remain hard for (BOOL, 0-VALID, PLANAR)-SDSs (SyDSs). (It should be noted that for any $k \geq 1$, $k$-inverted threshold function is 0-valid.)

**Theorem 6.3.** There is a local replacement based (decision, parsimonious)-reduction from 3SAT to the PRE problem for (BOOL, 0-VALID, PLANAR)-SDSs (SyDSs), even when each node computes the same 0-valid function. Thus, PRE is NP-complete, #PRE is #P-complete, APre is NP-complete and UPre is $\mathsf{D}^P$-complete for (BOOL, 0-VALID, PLANAR)-SDSs (SyDSs).
We will present the proof for (BOOL, 0-VALID, PLANAR)-SDSs and then indicate the necessary modifications to obtain the result for (BOOL, 0-VALID, PLANAR)-SyDSs.

**Proof.** We will present the proof for (BOOL, 0-VALID, PLANAR)-SDSs and then indicate the necessary modifications to obtain the result for (BOOL, 0-VALID, PLANAR)-SyDSs.

Given an instance of PLANAR-3SAT, consisting of the set \( X = \{x_1, x_2, \ldots, x_n\} \) of variables and the set \( C = \{c_1, c_2, \ldots, c_m\} \) of clauses, we create the following SDS \( S \). For each variable \( x_i \in X \), \( S \) has five nodes denoted by \( x_i, \overline{x_i}, y_i, z_i \) and \( w_i \), \( 1 \leq i \leq n \). For each clause \( c_j \in C \), \( S \) has three nodes, denoted by \( a_j, b_j \) and \( c_j \), \( 1 \leq j \leq m \). The edges in the underlying graph are as follows: (i) for each \( j \), the edges \( \{a_j, b_j\} \) and \( \{b_j, c_j\} \), \( 1 \leq j \leq m \); (ii) for each \( i \), the edges \( \{x_i, \overline{x_i}\}, \{y_i, x_i\}, \{y_i, \overline{x_i}\}, \{z_i, x_i\}, \{z_i, \overline{x_i}\}, \{w_i, x_i\} \) and \( \{w_i, \overline{x_i}\} \), \( 1 \leq i \leq n \); (iii) for each node \( c_j \), edges from \( c_j \) to the nodes corresponding to the three literals occurring in clause \( c_j \). (Fig. 8(a) shows the edges introduced in (i) and (ii) above.)

Given that the reduction is from an instance of Planar 3SAT, it can be seen that the resulting graph of the SDS \( S \) is planar.

The local transition function at each node is the 2-inverted-threshold function. The permutation \( \pi \) for the SDS \( S \) is

\[
\pi = (a_1, b_1, c_1, \ldots, a_m, b_m, c_m, y_1, z_1, w_1, \ldots, y_n, z_n, w_n, x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}).
\]

The required final configuration \( C \) is given as follows: the value of each node \( \forall j \ C(a_j) = 0, C(b_j) = 1, C(c_j) = 0 \) and \( \forall i \ C(y_i) = 1, C(z_i) = 0, C(w_i) = 1, C(x_i) = 0, C(\overline{x_i}) = 0 \).

Suppose there is a satisfying assignment to the PLANAR 3SAT instance. It can be verified that the following configuration \( C' \) is a predecessor of \( C \): for \( 1 \leq j \leq m \), set \( C'(a_j) = 1, C'(b_j) = 1 \) and \( C'(c_j) = 0 \); for \( 1 \leq i \leq n \), set \( C'(y_i) = 1, C'(z_i) = 0, C'(w_i) = 1 \) and set \( C'(x_i) \) and \( C'(\overline{x_i}) \) to the values given by the satisfying assignment. (Note that, for each \( i \), \( 1 \leq i \leq n \), the final values of \( x_i \) and \( \overline{x_i} \) are 0 since both of these nodes have edges to nodes \( y_i \) and \( w_i \) whose final values are 1.)

Now, suppose there is a configuration \( C' \) such that the SDS reaches \( C \) from \( C' \) in one step. It can be seen using the chosen permutation that \( C' \) must set \( a_j \) and \( b_j \) to 1 and \( c_j \) to 0, for each \( j \); otherwise, \( a_j \) and \( b_j \) cannot reach their specified final values. Now, for each \( c_j \) to reach the value 0 in \( C \), at least one of the nodes corresponding to the literal occurring in \( c_j \) must be set to 1. Further, for each \( i \), the chosen final values of \( y_i \) and \( z_i \) ensure that \( x_i \) and \( \overline{x_i} \) must be set to complementary values in \( C' \). Using this fact, it can also be seen that, for each \( i \), \( y_i \) and \( w_i \) must be set to 1 and \( z_i \) must be set to 0 in \( C' \). Thus, the assignment of values to \( x_i \) and \( \overline{x_i} \) provide a satisfying assignment to the given instance of 3SAT. This completes the proof of NP-hardness. Further, since the initial values of all nodes except \( x_i \) and \( \overline{x_i} \), \( 1 \leq i \leq n \), are determined by the construction, it can be seen that there is a one-to-one correspondence between the set of satisfying assignments to the Planar 3SAT instance and the set of predecessors for the configuration \( C \); that is, the reduction is parsimonious.

The proof for SyDSs uses a slightly different construction. The subgraph constructed for each variable \( x_i \) and each clause \( c_j \) are shown in Fig. 8(b). As in the construction for SDS, each clause node \( c_j \) is joined to the node corresponding to the three literals in \( c_j \) and the local transition function at each node is the 2-inverted-threshold...
function. The required final configuration $C$ is chosen as follows: $\forall i \in \text{C}(y_i) = 1$ and $\forall j \in \text{C}(b_j) = 1$. For every other node $v$, $\text{C}(v) = 0$. The following facts can be verified: (a) for each $i$, $1 \leq i \leq n$, the initial values of $y_i$ and $z_i$ must 0 and 1 respectively and those of $x_i$ and $\overline{x}_i$ must be complementary; (b) for each $j$, $1 \leq i \leq m$, the initial values of $a_j$ and $d_j$ must 1 and those of $b_j$ and $c_j$ should be 0. Using these facts, it can be seen that there is a one-to-one correspondence between the predecessors of $C$ and the set of satisfying assignments to the planar 3SAT instance. In other words, the reduction for SyDSs is also parsimonious.

### 6.3. Linear relations

Next, we 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, NONE)-SDS. To specify the node functions more precisely, consider each node $v_i$ of a (FIELD, LINEAR, NONE)-SDS, and let $N(i) = \{v_1, v_{i_1}, v_{i_2}, \ldots, v_r\}$ denote the neighbors of $v_i$, including $v_i$ itself. Each local transition function $f_i$, $1 \leq i \leq n$, has the following form:

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

Here, $\alpha_i$ 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 [71]) that solving a set linear equations over the field can be done in polynomial time. 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.

**Theorem 6.4.** Let $S$ be a (FIELD, LINEAR, NONE)-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 $f_i$ (Eq. (1)) is nonzero. For such an 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 $S$, we associate a variable $x_i$ with each node $v_i$ of $S$ and construct a system of linear equations over the algebraic field corresponding to $S$. 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 $S$. 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 $v_{i_1}, v_{i_2}, \ldots, v_i$ 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. Using Eq. (1), the linear equation for $v_i$, where the arithmetic operations are carried out over the field corresponding to $S$, is the following:

$$\alpha_i + a_{ii}x_i + \sum_{q=1}^r a_{iq}b_{iq} + \sum_{q=1}^p a_{iq}x_{iq} = b_i.$$  

(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 $S$ 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 $\pi$. 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 $v_j$ that follow $v_i$ in the permutation, the unique value $C'(v_j)$ has already been determined. In the equation for determining the new value of $v_i$, the only unknown is $C'(v_j)$. This is because the other values in the equation are from $C$ for neighboring nodes before $v_i$ in $\pi$, the already computed values from $C'$ for neighboring nodes after $v_i$ in $\pi$, and $C(v_i)$ itself. Since the entry $a_{ii}$ is not zero, this equation has a unique solution.

For (FIELD, LINEAR, NONE)-SDSs that do not satisfy the condition mentioned in Theorem 6.4 and for (FIELD, LINEAR, NONE)-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 6.5. The problems PRE, #PRE, UPRE and APRE for (FIELD, LINEAR, NONE)-SDSs and (FIELD, LINEAR, NONE)-SyDSs are in P.

Proof. First consider a (FIELD, LINEAR, NONE)-SDS. Using the steps mentioned in the proof of Theorem 6.4, 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 [71], it follows that the PRE problem for linear SDSs can be solved in polynomial time.

For (FIELD, LINEAR, NONE)-SyDS, since the nodes update their states synchronously, the form of linear equations is slightly different from the one given in Eq. (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, NONE)-SyDS is as follows.

\[
a_i + \sum_{v_q \in N'(i)} a_{iq}x_q = b_i. \tag{3}
\]

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 \(\mathbb{F}_2 = \{0,1\}\) under addition modulo 2. Thus, by Theorem 6.4, for any (BOOL, {XOR, XNOR})-SDS, the PRE problem has a unique solution which can be found efficiently. Additionally, by directly applying the ideas in Creignou and Hermann [10] one can count the number of solutions.

Reversibility. An SDS (SyDS) is said to be reversible if its global transition function \(F_S\) is reversible. For a finite reversible SDS (SyDS), each configuration \(C\) has a unique predecessor; in other words, for such SDSs (SySDs), the phase space consists of disjoint cycles. Theorem 6.4 provides a sufficient condition for (FIELD, LINEAR, NONE)-SDSs (SyDSs) to be reversible. Theorems 6.4 and 6.5 allow us to efficiently decide whether a given (FIELD, LINEAR, NONE)-SDS (SyDS) is reversible.

6.4. Putting it all together

The theorems in Sections 6.1–6.3 provide a first step towards possible dichotomy theorems for PREDECESSOR EXISTENCE and #PREDECESSOR EXISTENCE problems. It is instructive to compare the results obtained thus far with the elegant dichotomy theorems of Schaefer [60] for Boolean constraint satisfaction problems (SAT(S)) and of Creignou and Hermann [10] for #SAT(S).

Theorem 6.6 (Schaefer [60]). Let \(S\) be a finite set of finite arity Boolean relations. Let conditions (a) through (e) be defined as follows: (a) Every relation in \(S\) is 0-valid, (b) Every relation in \(S\) is 1-valid, (c) Every relation in \(S\) is weakly positive, (d) Every relation in \(S\) is weakly negative, (e) Every relation in \(S\) is affine, (f) Every relation in \(S\) is bijunctive. If \(S\) satisfies one of the conditions (a), (b), (c), (d), (e) or (f), then SAT(S) is in P; otherwise, SAT(S) is NP-complete.

Theorem 6.7 (Creignou and Hermann [10]). Let \(S\) be a finite set of finite arity Boolean relations. If every relation in \(S\) is affine, then #SAT(S) is in P, otherwise #SAT(S) is #P-complete.

The results obtained in Sections 6.1–6.3 can be summarized as follows.

Lemma 6.8. 1. If every relation in \(S\) is affine, then PRE, #PRE, APRE and UPRE are in P for (BOOL, S, NONE)-SDSs (SySDs).
2. There exists an \(S\) in which every relation is 0-valid such that PRE is NP-complete, #PRE is #P-complete, APRE is NP-complete and UPRE is D^P-complete for (BOOL, S, PLANAR)-SDSs (SySDs).
3. There exists an \(S\) in which every relation is 1-valid such that PRE is NP-complete, #PRE is #P-complete, APRE is NP-complete and UPRE is D^P-complete for (BOOL, S, PLANAR)-SDSs (SySDs).
4. There exists an \(S\) with the property that each relation in \(S\) is simultaneously bijunctive, weakly positive and weakly negative such that PRE is NP-complete, #PRE is #P-complete, APRE is NP-complete and UPRE is D^P-complete for (BOOL, S, PLANAR)-SDSs (SySDs).
7. Implications for other computational models

We briefly mention the implications of our results to other computational models of discrete dynamical systems.

Cellular automata and systolic networks: As mentioned earlier, treewidth bounded regular SDSs and SyDSs can be viewed as simple extensions of 1D-CA. For this reason, we refer to SyDSs (and SDSs) whose underlying graphs are regular and treewidth bounded as generalized 1D-CA. Theorem 4.1 shows that PREDECESSOR EXISTENCE problem is \( \text{NP}- \)hard for 2D-CAs with symmetric Boolean local transition functions. In contrast, Corollary 4.5 shows that PREDECESSOR EXISTENCE, AMBIGUOUS-PREDECESSOR EXISTENCE, \( \# - \)PREDECESSOR EXISTENCE and UNIQUE-PREDECESSOR EXISTENCE problems can be solved in polynomial time for 1D-CA regardless of the local transition functions so far as the radius is bounded and the domain is finite. Analogous results hold when nodes are updated using a specific permutation. Noting the equivalence between symmetric functions and totalistic functions when restricted to the Boolean domain, it follows that the PREDECESSOR EXISTENCE problem is \( \text{NP}- \)hard for 2D-CAs with totalistic local transition functions. In contrast, the PREDECESSOR EXISTENCE problem restricted to instances in which local transition functions are linear, can solved in polynomial time (Section 6.3). Thus, the complexity of PREDECESSOR EXISTENCE problem changes substantially with slightly more “powerful” local transition functions. A similar behavior has been observed for REACHABILITY problems for 2D-CAs [23].

Discrete Hopfield networks: A discrete Hopfield network consists of an undirected graph with a state value from the domain \( \{0, 1\} \) for each node, a threshold for each node and weights on edges. The next state of a node \( v \) is determined by a function of its current state, the states of the neighbors which have an edge from \( v \), the weights of those edges, and the threshold of \( v \). Note that AND and OR local transition functions can be viewed as simple types of threshold functions. Thus, our lower bound results show that PREDECESSOR EXISTENCE problem is \( \text{NP}- \)hard for discrete Hopfield networks with unit edge weights that have just two types of simple local transition functions. The graph topology plays a crucial role in this result. As shown in Theorem 4.3, for graphs of bounded treewidth, PREDECESSOR EXISTENCE problem can be solved in polynomial time when each node computes simple threshold function. Similarly, edge weights have significant effect: when large edge weights are allowed, PREDECESSOR EXISTENCE problem is \( \text{NP}- \)hard even when restricted to graphs of treewidth 2 (Theorem 5.8).

Communicating and concurrent finite state machines: The lower bounds presented here imply similar lower bounds for predecessor (preimage) existence questions for communicating finite state machines (CFSM). As mentioned earlier, verifying whether communication protocols have potential livelocks can be combinatorially modeled as predecessor existence questions for communicating finite state machines. Our results show that predecessor existence questions for very simple CFSMs are intractable. In contrast, we believe our polynomial time algorithms for treewidth bounded graphs are potentially applicable for verifying properties about protocols. Empirical observations about some of the CFSMs used for modeling protocols suggests that the resulting graphs are likely to have bounded treewidth.

8. Concluding remarks and open questions

We have comprehensively characterized the computational complexity of PREDECESSOR EXISTENCE problem and its three variants for discrete dynamical systems. Our results provide several ways of delineating hard and easy instances of the \( \text{Pre} \) problem for SDSs and SyDSs. Variants of these problems have been considered in the literature in the context of data flow analysis, program and hardware verification, and modeling and simulation.

We conclude with some directions for future research. One direction is to identify other restrictions on the graph structure or on the local transition functions that render the \( \text{Pre} \) problem efficiently solvable. The most important open question is whether there are precise dichotomy theorems for PREDECESSOR EXISTENCE, \( \# - \)PREDECESSOR EXISTENCE, UNIQUE-PREDECESSOR EXISTENCE and AMBIGUOUS-PREDECESSOR EXISTENCE problems similar to those for \( \text{SAT}() \) and \( \# \text{SAT}() \). The existence of such dichotomy results will provide a very different way to classify SDSs (and CAs) as compared to the earlier work of Wolfram [72,23] and Culik et al. [23]. The results obtained in this paper are a step in this direction. Finally, it is of interest to further study SIMULTANEOUS reductions and their implications on computational complexity theory.
Acknowledgments

We are grateful to the referees for their careful review of the manuscript. Their suggestions helped us to significantly improve the presentation and to correct some of the constructions so that they are parsimonious. Some of the research was funded by LDRD-DR research project Scalable Reconfigurable Computing at the Los Alamos National Laboratory. We thank Cris Moore (University of New Mexico), Anil Kumar (VBI and Virginia Tech), Gabriel Istrate (Los Alamos National Laboratory), Henning Mortveit (VBI and Virginia Tech), Pekka Orponen (Helsinki University of Technology) and Christian Reidys (Los Alamos National Laboratory) for discussions on topics related to this paper. We also thank Pekka Orponen for providing us with copies of his survey papers.

Part of the work of Chris Barrett and Madhav V. Marathe was done while the authors were at Los Alamos National Laboratory, Los Alamos, NM.

Harry B. Hunt III, S.S. Ravi, Daniel J. Rosenkrantz and Richard E. Stearns were supported by a grant from Los Alamos National Laboratory and by NSF Grants CCR-01-05536 and CCR-97-34936.

Appendix A. Other definitions

A.1. Boolean constraint satisfaction problems

In this paper, hardness results for several variants of the $\text{PRE}$ problem are established using reductions from appropriate 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 $BG$ (also called the factor graph [35]) 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 $\text{NP}$-complete [25, 29].

Definition A.1. (a) 3SAT is the restricted version of SAT where each clause contains no more than three literals.

EX3SAT is a restriction of 3SAT in each clause contains exactly 3 literals. PLANAR-3SAT is a restriction of 3SAT to instances whose associated bipartite graphs are planar. In a further restricted version of Planar 3SAT (denoted by RP3SAT), each clause has exactly three literals and each variable appears in at most three clauses.

(b) PLANAR MONOTONE 3SAT (also called GOLD’S MONOTONE 3SAT) is the restricted version of 3SAT in which each clause contains at most three unnegated literals or at most three negated literals and the associated bipartite graph is planar.

(c) 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.

Our results point out some interesting parallels between the complexity of the $\text{PRE}$ problem and Boolean constraint satisfaction problems (also called generalized Boolean satisfiability problems). Details regarding these are given in Section 6. Here, we recall some terminology from [60] concerning the generalized satisfiability problem.

A Boolean formula $\Phi$ is weakly positive if it is logically equivalent to some $\text{CNF}$ formula having at most one negated variable in each clause. A Boolean formula $\Phi$ is weakly negative if it is logically equivalent to some $\text{CNF}$ formula having at most one unnegated variable in each clause. (Such a clause is also known as a HORN clause.) A logical relation $R(x_1, x_2, \ldots)$ is bijunctive if $R$ is logically equivalent to some $\text{CNF}$ formula having at most two literals in each conjunct. A logical relation $R$ is 0-valid if $(0, \ldots, 0) \in R$; $R$ is 1-valid if $(1, \ldots, 1) \in R$. 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 $\mathbb{Z}_2$.

Let $D$ be an arbitrary (not necessarily finite) nonempty set and let $S$ be a finite set of finite-arity relations on $D$. An $S$-clause ($S$-term) is a relation in $S$ applied to variables and constants in $D$. An $S$-formula is a finite nonempty conjunction of $S$-clauses. The problem of determining the satisfiability of finite conjunctions of relations in $S$ is denoted by $\text{SAT}(S)$.
A.2. Graph theoretic definitions

The following definitions of tree decomposition and treewidth are from [9].

Definition A.2. Given an undirected graph $G(V, E)$, a tree decomposition of $G$ is a pair $((X_i | i \in I), T = (I, F))$, where $\{X_i | i \in I\}$ 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 $X_i \cap X_k \subseteq X_j$.

The treewidth of a tree decomposition $((X_i | i \in I), T)$ 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 $k$.

We note here that a number of graph classes are known to have bounded treewidth. They include trees, $k$-outerplanar graphs, $k$-bandwidth bounded graphs (both for constant $k$), series parallel graphs, Halin graphs, chordal graphs of bounded clique size, etc. A number of problems that are $\text{NP}$-hard on general graphs can be solved efficiently when restricted to the class of treewidth bounded graphs. A considerable amount of work has been done in this area (see [9, 45] and the references therein). Grid graphs are a prototypical example of class of graphs that do not have bounded treewidth. In general, a grid graph with $n$ nodes has a treewidth of $\Theta(\sqrt{n})$.

Appendix B. Hardness of PRE for star graphs

We include below a statement and proof sketch to show hardness results for the PRE problem and its variants when the underlying graphs is a star (which has a treewidth of 1) and just one of the local transition function is nonsymmetric.

Proposition B.1. There is a local replacement based (decision, parsimonious)-reduction from 3SAT to PRE problem for (BOOL, NONE, STAR)-SDSs (SyDSs), where at most one of the local transition functions is nonsymmetric. Thus PRE is $\text{NP}$-complete, $\#\text{PRE}$ is $\text{#P}$-complete, $\text{APRE}$ is $\text{NP}$-complete and $\text{UPRE}$ is $\text{D}^\text{P}$-complete for such SDSs (SyDSs).

Proof sketch. For the sake of brevity, we sketch the reduction for SDSs. The reduction for SyDSs is similar.

Given an instance of 3-SAT with $n$ variables $x_1, x_2, \ldots, x_n$ and the 3SAT formula $F$, construct an SDS $S$ as follows. The underlying graph is a star with $n + 1$ nodes. Let $v_0$ denote the center of the star (i.e., node of degree $n$) and $v_1, v_2, \ldots, v_n$ denote the $n$ other nodes, each of degree 1. Let $s_i$ denote the state value of node $v_i, 0 \leq i \leq n$. For $v_0$, the local transition function is given by $s_0 \land F$. For $v_1, 1 \leq i \leq n$, the local transition function is the 0-simple-threshold function. Note that all the local transition functions, except for the one at $v_0$, are symmetric. The node permutation is given by $\langle v_0, v_1, \ldots, v_n \rangle$ and the final configuration $C$ has the value 1 for all the nodes. It is straightforward to verify that $C$ has a predecessor if and only if $F$ has a satisfying assignment. Moreover, this transformation is also parsimonious.

References


[54] W. Peng, Deadlock detection in communicating finite state machines by even reachability analysis, Mobile Networks and Applications 2 (3) (1997) 251–257.


