ON THE AUTOMATIC COMPUTATION OF NETWORK INVARIANTS

by

Felice Balarin and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M94/18

31 March 1994

COVER PAGE

ON THE AUTOMATIC COMPUTATION OF NETWORK INVARIANTS

by

Felice Balarin and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M94/18

31 March 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ON THE AUTOMATIC COMPUTATION OF NETWORK INVARIANTS

by

Felice Balarin and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M94/18

31 March 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

On the Automatic Computation of Network Invariants

Felice Balarin* Alberto L. Sangiovanni-Vincentelli
Department of Electrical Engineering and Computer Science
University of California, Berkeley, CA 94720

Abstract

We study ***network invariants***, abstractions of systems consisting of arbitrary many identical components. In particular, we study a case when an instance of some fixed size serves as an invariant. We show that the existence of such an invariant is undecidable in general, present a procedure that will find it, if one exists, and finally give conditions under which such an invariant does not exist. These conditions can be checked in finite time, and if satisfied, they can be used in further searches for an invariant.

1 Introduction

The ability to create abstractions has been key in formal verification of complex digital systems (for example, see [CGH$^+$93]). Usually, an abstraction is generated manually, at the considerable expense of time by the expert with the deep understanding of both the verification tool, and the system being designed.

One specific class of abstractions applies to systems with many identical components (also referred to as networks or iterative systems). Ideally, an abstraction of such a system should not depend on the actual number of components. Such an abstraction is called a ***network invariant***. Once an invariant of manageable size is found it allows:

- a verification of a large system with a fixed number of components; and at the same time also
- a verification of the entire class of systems with the same structure but with different number of components.

This is of particular interest for distributed systems where algorithms (e.g. mutual exclusion) are usually designed to be correct for systems of any number of concurrent processes.

Although iterative systems have been studied for a long time [Hen61], only recently there has been a significant interest in the formal verification of such systems. In the framework of model checking, Browne, Clarke and Grumberg [BCG89], and Shtadler and Grumberg [SG90] have studied conditions under which

*Supported by SRC under grant # 94-DC-008.

the satisfaction of formulae of certain temporal logics is independent of the size of the system. In [SG90] the conditions seem to be quite restrictive, while in [BCG89] the conditions cannot in general be checked automatically. Wolper and Lovinfosse [WL90] have studied formal verification of iterative systems generated by interconnecting identical processes in a certain regular fashion. They also present some decidability results for related problems. Kurshan and McMillan [KM89] present slightly more general results which can be applied both to process algebra and automata-based approaches. In both cases, automatic tools are used only to verify that a finite state system suggested by the user is indeed an invariant. Kurshan and McMillan have hinted that it might be a good idea to check whether a system of some fixed size serves as an invariant. This idea was further developed by Rho and Somenzi [RS92, RS93], who have studied different network topologies and presented several sufficient conditions for the existence of such an invariant.

In this paper we address a problem of finding an invariant automatically. More precisely, we introduce a notion of a *tight* invariant, and give some results that can help a search for it. Intuitively, an invariant of a class of systems is a finite-state system that can exhibit any behavior that some system in the class can, and possibly some additional behaviors. Thus, in pre-order based formal verification paradigms (where a system is verified if it does not exhibit any undesirable behavior), an invariant is a conservative abstraction: if an abstraction is verified, so is every system in the class, but not vice versa. A tight invariant is an exact abstraction: if an abstraction is verified, so is every system in the class, and if an abstraction is not verified than there exists a system in the class which exhibits undesirable behavior. Thus, a tight invariant must exhibit *exactly* those behaviors that are exhibited by systems in the class.

Finding a tight invariant is easy if a *finite invariant* exists, i.e. if any behavior exhibited by a system in the class is also exhibited by some system in some *finite* subclass. The main contribution of this paper is the test that can show that a finite invariant does not exist. The test is constructive in a sense that if successful, it identifies a set of behaviors that cannot be "covered" by any finite subclass, but must be exhibited by a tight invariant. Once identified, such a behavior can be added to the behaviors of some finite subclass to possibly generate a tight (but not finite) invariant. Unfortunately, we can not hope for a general algorithm that identifies all such sets of behaviors, because the existence of a finite invariant is undecidable (see Theorem 1). To the best of our knowledge no other algorithmic tests for the non-existence of a finite invariant are available.

The rest of the paper is organized as follows. In Section 2 we formally define the class of automata we consider, as well as rules by which these automata can be combined to form iterative systems. In Section 3 we apply these concepts to formal verification by language containment, and in Section 4 we illustrate them on typical examples. The focus of our paper is the computation of the finite invariant presented in Section 5,

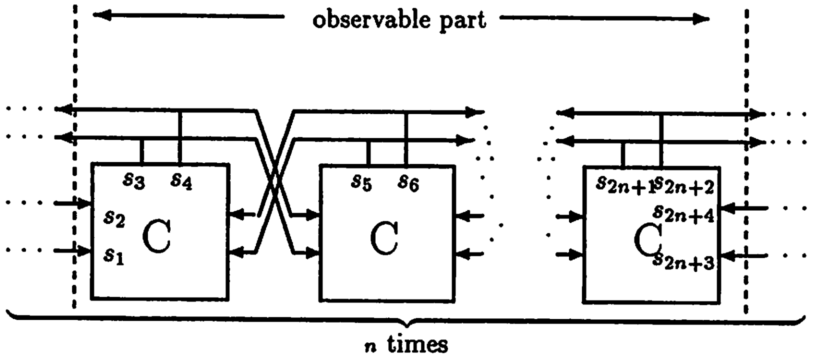

Figure 1: An open network N_n.

where results on the decidability and the existence as well as the test for the non-existence of a finite invariant are given.

2 Basic Definitions

In this section we formalize the notion of an iterative system consisting of many identical automata. We assume that the state of the basic cell is fully observable and that the transitions of the basic cell can depend on the states of its left and right neighbors (see Figure 1). These restrictions still enable us to model many regular hardware arrays, such as stacks, FIFO buffers and counters. Other examples that fit into our framework are a token passing mutual exclusion protocol [WL90] and the ever-so-popular Dining Philosophers Problem [KM89]. We restrict ourselves to automata on finite tapes. Thus, using our approach only safety properties can be verified.

In this paper, we adopt a standard definition of an automaton. More precisely, an *automaton* A over some finite alphabet Σ is a quadruple (S, I, T, F), where S is some finite *set of states*, $I \subseteq S$ is a *set of initial states*, $T \subseteq S \times \Sigma \times S$ is a *transition relation*, and $F \subseteq S$ is a *set of final states*. The language of A (denoted by $\mathcal{L}(A)$ is a set of all strings $x_1 x_2 \ldots x_k \in \Sigma^*$ for which there exists a sequence of states $s_0, s_1, \ldots, s_k$ such that $s_0 \in I$, $s_k \in F$ and $(s_{i-1}, x_i, s_i) \in T$ for all $i = 1, \ldots, k$.

Next, we formalize a notion of connecting two subsystems to form a bigger one. Given some sets[1] Σ, Σ_A and Σ_B and some automata $A = (S_A, I_A, T_A, F_A)$ over alphabet $\Sigma_A \times \Sigma$ and $B = (S_B, I_B, T_B, F_B)$ over alphabet $\Sigma \times \Sigma_B$, a *concatenation by* Σ, denoted by $A \cdot_\Sigma B$ is an automaton over alphabet $\Sigma_A \times \Sigma \times \Sigma_B$

[1] Any of Σ, Σ_A and Σ_B can be empty provided that $\Sigma_A \times \Sigma$ and $\Sigma \times \Sigma_B$ are non-empty. By convention if S is a set $\emptyset \times S = S \times \emptyset = S$.

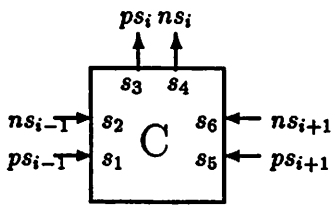

Figure 2: A basic cell; *ps* and *ns* denote present and next states.

defined by:

$$A \cdot_\Sigma B = (S_A \times S_B, I_A \times I_B, T_{A \cdot_\Sigma B}, F_A \times F_B),$$

where $T_{A \cdot_\Sigma B}$ contains exactly those 7-tuples $((s_A, s_B), (\sigma_A, \sigma, \sigma_B), (t_A, t_B))$ satisfying both $(s_A, (\sigma_A, \sigma), t_A) \in T_A$ and $(s_B, (\sigma, \sigma_B), t_B) \in T_B$.

A reader familiar with process algebra will easily interpret this operation as pairing ports in A that correspond to Σ, with corresponding ports in B.

We say that an automaton $A = (S, I, T, F)$ is a *cell* if it is defined over alphabet S^6 and the following condition holds:

$$(s, (s_1, s_2, s_3, s_4, s_5, s_6), t) \in T \iff s = s_3 \text{ and } t = s_4.$$

Intuitively, the alphabet of a cell consists of three parts: s_1 and s_2 are the present and the next state of the left neighbor, s_3 and s_4 are the present and the next state of the cell itself, and finally s_5 and s_6 are the present and the next state of the right neighbor, as shown in Figure 2.

Given a cell C over alphabet S^6 and an automaton E over S^4 (called an *environment*) a *network* of length n is defined by:

$$N_n = E \cdot_{S^4} \underbrace{C \cdot_{S^4} C \cdot_{S^4} \ldots \cdot_{S^4} C}_{n \text{ times}}.$$

If the behavior of the environment is unrestricted, i.e. if $\mathcal{L}(E) = (S^4)^*$ we say that the network is open. In that case:

$$N_n = \underbrace{C \cdot_{S^4} C \cdot_{S^4} \ldots \cdot_{S^4} C}_{n \text{ times}}.$$

Otherwise, we say that the network is left-closed. In this paper, we will consider only open and left-closed networks, but the results are easily dualized for right-closed networks. Also, in the rest of this paper all concatenation will be by S^4, so without ambiguity we write $\cdot$ for $\cdot_{S^4}$.

Conceptually, networks can be parts of larger systems, but we require that all coordination is done through concatenation by S^4, on the left or right. For such a coordination one needs to consider only the peripheral components of the language of N_n, as shown in Figure 1. Therefore, we introduce a notion of an *observable part*, first for elements of S^{2n+4}:

$$O((s_1, s_2, \ldots, s_{2n+4})) = (s_1, s_2, s_3, s_4, s_{2n+1}, s_{2n+2}, s_{2n+3}, s_{2n+4}),$$

and then, we extend the notion naturally to strings $s_1 s_2 \ldots s_k \in (S^{2n+4})^*$ by:

$$O(s_1 s_2 \ldots s_k) = O(s_1)O(s_2)\ldots O(s_k),$$

and languages $\mathcal{L} \subseteq (S^{2n+4})^*$ by:

$$O(\mathcal{L}) = \{x | x = O(y) \text{ for some } y \in \mathcal{L}\}.$$

For clarity, we write $\mathcal{L}_n$ for $O(\mathcal{L}(N_n))$ where N_n is a network of length n.

An *iterative system* $\{N_n | n \geq 1\}$ is the class of all networks generated by the same cell and environment. Given an iterative system $\{N_n | n \geq 1\}$, an *invariant* is any finite-state automaton A satisfying $\mathcal{L}(A) \supseteq \mathcal{L}_\infty$ where $\mathcal{L}_\infty = \bigcup_{n=1}^{\infty} \mathcal{L}_n$. If $\mathcal{L}(A) = \mathcal{L}_\infty$ we say that A is a *tight invariant.* If in addition $\mathcal{L}(A) = \bigcup_{n=1}^{n^*} \mathcal{L}_n$ for some $n^* < \infty$ we say that A is a *finite invariant.*

An invariant is an abstraction of every network in an iterative system. How such an abstraction can be utilized is described in the following section.

3 Applications to Language Containment

We now consider the application of these concepts to formal verification by language containment. Given an iterative system and some regular language $\mathcal{L}_p$ describing a property to be verified, we want to show that $\mathcal{L}_n \subseteq \mathcal{L}_p$, independently of n [KM89]. If A is an invariant of such a system, then:

$$\mathcal{L}(A) \subseteq \mathcal{L}_p \implies \mathcal{L}_n \subseteq \mathcal{L}_p, \; \forall n. \tag{1}$$

If A is a tight invariant (1) can be strengthen to:

$$\mathcal{L}(A) \subseteq \mathcal{L}_p \iff \mathcal{L}_n \subseteq \mathcal{L}_p, \; \forall n. \tag{2}$$

If A is a finite invariant (2) reduces to:

$$\mathcal{L}_n \subseteq \mathcal{L}_p, \; \forall n = 1, \ldots, n^* \iff \mathcal{L}_n \subseteq \mathcal{L}_p, \; \forall n. \tag{3}$$

Obviously, a tight invariant exists if and only if $\mathcal{L}_\infty$ is regular. One may try to find a finite-state automaton A such that (2) holds even if $\mathcal{L}_\infty$ is not regular. In other words, one may try to find an invariant that is not tight, but that has a language contained in the languages of all other invariants. Unfortunately, if $\mathcal{L}_\infty$ is not regular, such an invariant does not exist. To see this, assume that A is such an invariant. Let x be some string in $\mathcal{L}(A) - \mathcal{L}_\infty$ (since $\mathcal{L}(A)$ is regular and $\mathcal{L}_\infty$ is not, such a string always exist). Since a language containing just x is also regular, one can construct an automaton A' such that $\mathcal{L}(A') = \mathcal{L}(A) - \{x\}$. Now, A' is an invariant and $\mathcal{L}(A) \not\subseteq \mathcal{L}(A')$, contradicting the assumption that A is the tightest regular invariant.

We have defined an iterative system both in terms of the basic cell and the environment. An invariant of such a system is its abstraction in one specific environment. It is also useful to study the corresponding open system. An invariant of such a system is an abstraction applicable to any environment. If A is an invariant of an open system, and E is a model of some specific environment, we can verify the behavior of the system in the environment E by verifying relations similar to those in (1)–(3), except that $\mathcal{L}(A)$ is replaced by $\mathcal{L}(A) \cap \mathcal{L}(E)$. This approach may result in saving if we consider a system in several different environments, but it may be disadvantageous in cases when a tight invariant exists for a closed system, but not for an open one. We will describe one such a case in the following section.

4 Examples

The following three examples illustrate three possible cases: when a finite invariant exists (Example 1), when a tight invariant exists, but a finite one does not (Example 2), and finally when a tight invariant does not exists (Example 3). All three examples are abstractions of buffers with different discipline of passing a token. In all three cases cells are initially in *idle* state. The state *token* indicates that a particular cell holds a token. In Examples 1 and 2, a cell moves to a special *dead* state once it has delivered a token. In the description that follows variables ps_{n-1}, ns_{n-1}, ps_{n+1} and ns_{n+1} take value of the present and the next state of immediate neighbors of the cell under consideration. For simplicity, in all three examples all states are considered final. Also, all systems are open.

Example 1: In this example a cell can hold a token for any (possibly infinite) number of steps before delivering it to its neighbor. A cell can deliver only one token. More precisely the transition relation of the cell is defined by:

$$\begin{array}{ll}
\textit{idle} \longrightarrow \textit{idle}: & \text{if } ps_{n-1} \neq \textit{token} \text{ or } ns_{n-1} \neq \textit{dead} \\
\textit{idle} \longrightarrow \textit{token}: & \text{if } ps_{n-1} = \textit{token} \text{ and } ns_{n-1} = \textit{dead} \\
\textit{token} \longrightarrow \textit{token}: & \text{always} \\
\textit{token} \longrightarrow \textit{dead}: & \text{always} \\
\textit{dead} \longrightarrow \textit{dead}: & \text{always}
\end{array}$$

In this case a finite invariant exists. In fact it is achieved for $n^* = 3$. For $n \geq 3$, a language $\mathcal{L}_n$ can be described by languages of the leftmost and the rightmost cell and the following additional constraint:

"If the rightmost cell ever moves from *idle* to *token* it will happen at least $n-2$ steps after the leftmost cell leaves the *token* state."

Clearly, $\mathcal{L}_3$ (strictly) contains all $\mathcal{L}_n$'s, $n > 3$.

Example 2: In this example a cell holds a token for exactly one step. Again, once it delivers a single token, a cell will move to the *dead* state. The transition relation of the basic cell is:

$$\begin{array}{ll}
idle \longrightarrow idle: & \text{if } ps_{n-1} = idle \\
idle \longrightarrow token: & \text{if } ps_{n-1} = token \text{ and } ns_{n-1} = dead \\
token \longrightarrow dead: & \text{if } ps_{n-1} = dead \text{ and } ns_{n-1} = dead \\
dead \longrightarrow dead: & \text{if } ps_{n-1} = dead \text{ and } ns_{n-1} = dead
\end{array}$$

In this case a finite invariant does not exist. All strings in $\mathcal{L}_n$ $(n \geq 2)$ that are long enough must satisfy the following constraint:

"The rightmost cell moves *idle* to *token* *exactly* $n-2$ step after the leftmost cell leaves the *token* state."

Obviously, for all $n \geq 3$ there are some strings in $\mathcal{L}_{n+1}$ which are not in $\mathcal{L}_n$. However, a tight invariant invariant exists, and it is similar to the one in the previous example, except that the peripheral cells are restricted to remain in the *token* state for one step only. In fact, a tight invariant can be obtained as a union of $\mathcal{L}_1$, $\mathcal{L}_2$ and the observable part of the language of the network consisting of the two peripheral cell like those in this example and one internal cell like those in Example 1.

Example 3: This example is similar to the previous one, except that once a cell delivers a token it will move back to the *idle* state and become ready to accept a new token.

$$\begin{array}{ll}
idle \longrightarrow idle: & \text{if } ps_{n-1} \neq token \\
idle \longrightarrow token: & \text{if } ps_{n-1} = token \\
token \longrightarrow idle: & \text{always}
\end{array}$$

In this case $\mathcal{L}_\infty$ is not regular, so a tight invariant cannot exist. Indeed, $\mathcal{L}_\infty$ must include $\mathcal{L}_1$ and all strings for which there exists $k \geq 0$ such that the rightmost cell moves from *idle* to *token* exactly k steps after the leftmost cell leaves the *token* state. Notice that for any given string k must be constant. It is straightforward to show that such a language is not regular.

However, if we include in the description of the system an environment which allows at most one token in the system, a tight invariant exists and is similar to the one in Example 2.

5 Computing a Finite Invariant

In this section we focus on the problem of computing a finite invariant. We first discuss the decidability of the existence of finite invariant. Then, we give a simple semi-decision procedure that will find a finite invariant, if one exists. Finally, we attack the problem from the other side, giving some sufficient condition for the non-existence of a finite invariant.

5.1 Decidability

First, we describe a particular iterative system that we use for proving the key result of this section. For an arbitrary (one-way tape, deterministic) Turing machine TM with empty tape, let the iterative system $IS(TM)$ be generated by the cell that is a composition of two finite-state components described bellow.

The states of the component denoted by T are in 1–1 correspondence to the tape alphabet of TM. The initial state is blank. The state of component T of the i-th cell in the network is at any time equal to the value of the i-th letter on the tape.

The component C is a copy of the finite-state control of TM augmented by two special states L and R. If C is any other state than these two, we say that the cell is active. The initial state of C is L. While in it, a cell waits until its left neighbor is active and have to move to the right. At that points C moves to appropriate state of TM's control, possibly changing the state of T and after that moves to L or R depending on whether a head has to be moved left or right. Similarly to L, in state R a cell monitor its right neighbor.

An environment E of $IS(TM)$ is almost identical to the basic cell, except:

- it does not have an L state, hence it does not monitor its left neighbor,
- the component C must be modified to halt whenever TM's control calls for move to the left,
- the initial state of C must be equal to the initial state of TM's control.

It should be clear from the description that $IS(TM)$ is left-closed and that it simulates TM. In particular, the following result holds.

Lemma 1 *An left-closed iterative system $IS(TM)$ has a finite invariant if and only if TM uses finite memory.*

Proof: Assume that TM uses at most M tape symbols. Then $\mathcal{L}_n$ is the same for all $n > M$. Hence, an automaton with the language $\bigcup_{n=1}^{M+1} \mathcal{L}_n$ is a finite invariant. On the other hand, if TM uses infinitely many tape locations for all n there must be a string in $\mathcal{L}_n$ not in $\mathcal{L}_k$ for any $k < n$, because cell n must be activated for the first time at least one step after the k-th cell is. □

Theorem 1 *The existence of finite invariant for a left-closed iterative system is undecidable.*

Proof: It is undecidable whether an arbitrary one-way tape, deterministic Turing machine with empty tape uses finite memory, and by Lemma 1 that problem is reducible to deciding the existence of a finite invariant for a left-closed iterative system. □

At present, it is not clear whether this proof can be extended to open systems. In fact, this result is similar to Theorem 4.3 in [WL90] and Theorem 4 in [Hen61]. In all cases, the proofs substantially rely on the ability to distinguish one cell in the network: in our case, it is the environment, in [WL90] the first cell is explicitly distinguished, and in [Hen61] one cell is distinguished by different boundary condition. Therefore, the decidability of the existence of a finite invariant for open systems is still an open problem.

5.2 Existence of Finite Invariant

In this section we develop a sufficient and necessary condition for the existence of finite invariants, and propose a semi-decision procedure to check for that condition. But first, we need to introduce some auxiliary notation and results.

In what follows, all results refer to an iterative system $\{N_n | n \geq 1\}$ induced by a cell C. We use A, B and D to denote arbitrary automata, and assume that their alphabets are such that operations applied to them are well defined.

We denote by $A \oplus B$ an automaton with the language $\mathcal{L}(A) \cup \mathcal{L}(B)$ and by $\langle A \rangle$ an automaton with the language $O(\mathcal{L}(A))$. Both $A \oplus B$ and $\langle A \rangle$ are easily constructed, e.g. see a classical textbook by Hopcroft and Ullman [HU79]. We write $A \leq B$ if:

$$O(\mathcal{L}(A)) \subseteq O(\mathcal{L}(B)).$$

If both $A \leq B$ and $B \leq A$ hold, we write $A \equiv B$.

Proposition 1 *The following holds:*

a) *$\leq$ is a pre-order,*

b) *concatenation, language union and partial observation are monotonic with respect to $\leq$, i.e. if $A \leq B$, then the following also holds:*

$$\begin{aligned} A \cdot D &\leq B \cdot D, \\ A \oplus D &\leq B \oplus D, \\ \langle A \rangle &\leq \langle B \rangle, \end{aligned}$$

c) $N_n \cdot C \equiv \langle N_n \rangle \cdot C$,

d) $\langle A \rangle \oplus \langle B \rangle \equiv \langle A \oplus B \rangle$,

e) $(A \oplus B) \cdot D \equiv (A \cdot C) \oplus (B \cdot C)$.

Proposition 1 is easily proven by elementary automata theory.

Theorem 2 *If $N_1 \leq A$ and $A \cdot C \leq A$, then A is an invariant.*

Proof: It suffices to show that for all $n \geq 1$ $N_n \leq A$. The proof proceeds by induction. The base case $N_1 \leq A$ holds by the assumption of the theorem. Now assume $N_n \leq A$. We have

$$N_{n+1} = N_n \cdot C \leq A \cdot C \leq A,$$

where the first inequality follows from part b) of Proposition 1 and inductive assumption, and the second inequality is the assumption of the theorem. □

Both Kurshan and McMillan [KM89] and Wolper and Lovinfosse [WL90] require by definition that an invariant satisfy the conditions similar to those in Theorem 2. We have adopted a broader definition, motivated by the application described in Section 3. Still, Theorem 2 provides the only finite procedure known to us, for verifying that a given automaton with non-trivial language is indeed an invariant.

Theorem 3 *A finite invariant exists if and only if there exists $n^* < \infty$ such that:*

$$N_{n^*+1} \leq \bigoplus_{n=1}^{n^*} \langle N_n \rangle. \tag{4}$$

Proof: The only if part is obvious. By Theorem 2 to prove the "if" part it suffices to show that $A = \bigoplus_{n=1}^{n^*} \langle N_n \rangle$, satisfies the relation $A \cdot C \leq A$ ($N_1 \leq A$ holds trivially). We prove this by the following chain of reasoning:

$$\begin{array}{rcl}
\mathcal{L}_{n^*+1} & \subseteq & \bigcup_{n=1}^{n^*} \mathcal{L}_n \iff \\
\bigcup_{n=1}^{n^*} \mathcal{L}_{n+1} & \subseteq & \bigcup_{n=1}^{n^*} \mathcal{L}_n \iff \\
\mathcal{L} \bigoplus_{n=1}^{n^*} \langle \mathcal{L}(\langle N_n \rangle \cdot C \rangle & \subseteq & \mathcal{L}(\bigoplus_{n=1}^{n^*} \langle N_n \rangle) \iff \\
\bigoplus_{n=1}^{n^*} (\langle N_n \rangle \cdot C) & \leq & \bigoplus_{n=1}^{n^*} \langle N_n \rangle \iff \\
(\bigoplus_{n=1}^{n^*} \langle N_n \rangle) \cdot C & \leq & \bigoplus_{n=1}^{n^*} \langle N_n \rangle
\end{array}$$

$$\underbrace{\begin{bmatrix} s^2_{1,n}\\ s^1_{1,n}\\ s^2_{1,n-1}\\ s^1_{1,n-1}\\ \vdots \\ s^2_{1,2}\\ s^1_{1,2}\\ s^2_{1,1}\\ s^1_{1,1}\end{bmatrix}}_{s_1}\underbrace{\begin{bmatrix} s^2_{2,n}\\ s^1_{2,n}\\ s^2_{2,n-1}\\ s^1_{2,n-1}\\ \vdots \\ s^2_{2,2}\\ s^1_{2,2}\\ s^2_{2,1}\\ s^1_{2,1}\end{bmatrix}}_{s_2}\cdots\underbrace{\begin{bmatrix} s^2_{|s|-1,n}\\ s^1_{|s|-1,n}\\ s^2_{|s|-1,n-1}\\ s^1_{|s|-1,n-1}\\ \vdots \\ s^2_{|s|-1,2}\\ s^1_{|s|-1,2}\\ s^2_{|s|-1,1}\\ s^1_{|s|-1,1}\end{bmatrix}}_{s_{|s|-1}}\underbrace{\begin{bmatrix} s^2_{|s|,n}\\ s^1_{|s|,n}\\ s^2_{|s|,n-1}\\ s^1_{|s|,n-1}\\ s^2_{|s|,n-2}\\ \vdots \\ s^1_{|s|,3}\\ s^2_{|s|,2}\\ s^1_{|s|,2}\\ s^2_{|s|,1}\\ s^1_{|s|,1}\end{bmatrix}}_{s_{|s|}}$$

Figure 3: A naming scheme for parts of the string.

where the first equivalence is elementary, the second follows from the definition of $\oplus$ and $\langle\cdot\rangle$ and part c) of Proposition 1, the third one is by part d) and the final one by part e) of Proposition 1. $\square$

Note that we have actually proved a stronger claim: if the condition of Theorem 3 holds, then $\bigoplus_{n=1}^{n^*}\langle N_n\rangle$ is an invariant. This immediately gives us the following semi-decision procedure:

1. initialize $A = \langle N_1\rangle$ and $n = 2$,
2. **if** $A \cdot C \leq A$ **then** *HALT*
3. let $A = A \oplus \langle N_n\rangle$, let $n = n + 1$, go to step 2.

If the procedure terminates, it will produce a finite invariant A. However, if a finite invariant does not exist, the procedure will not terminate. We do not address here the efficiency issue for the procedure above. We just note that every iteration requires a solution of the difficult problem of language inclusion between (possibly non-deterministic) automata.

5.3 Proving the Non-existence of a Finite Invariant

In this section, we show a sufficient condition for the non-existence of a finite invariant. The condition can be checked algorithmically, and, if satisfied, it provides useful information on sets of strings that every invariant must include in its language.

In this section we consider a generic *open* iterative system induced by a cell (S, I, T, F). Unless stated otherwise, we assume that s is some string in $(S^{2n})^*$. We use $|\cdot|$ to denote the length of a string. To refer

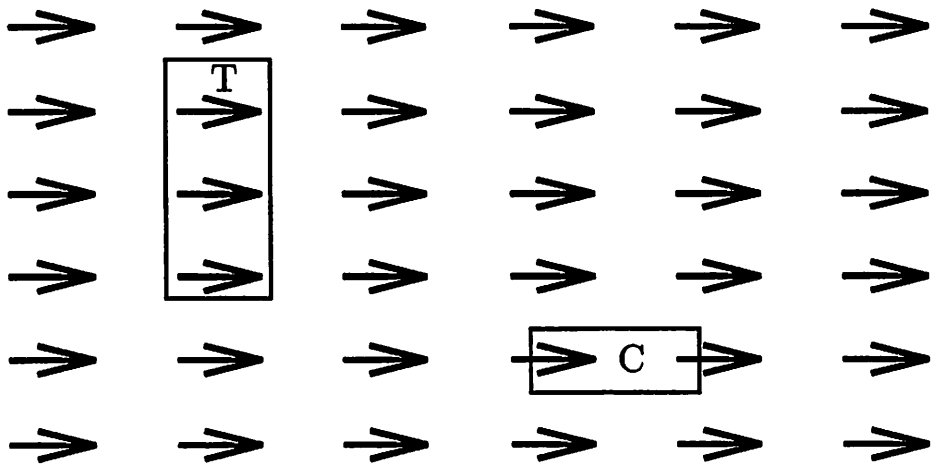

Figure 4: A symbolic representation of a string

to parts of the string we use the naming scheme detailed in Figure 3. We call a pair $s_{x,y} = (s^1_{x,y}, s^2_{x,y})$ a *transition*, for all $x = 1, \ldots, |s|$ and all $y = 1, \ldots, n$. If s, t and u are transitions, for simplicity we write $(s, t, u) \in T$ instead of:

$$(t^1, (s^1, s^2, t^1, t^2, u^1, u^2), t^2) \in T.$$

The reader might find it useful to visualize claims in this section, as suggested in Figure 4. It shows a symbolic representation of a string $s \in (S^{2n})^*$, with $n = 6$ and $|s| = 6$. Each transition is represented by an arrow. Conditions for s to be in $\mathcal{L}(N_4)$ can be restated in terms of this symbolic representation as follows:

initialization: present-state components of all transitions in the first column (except the top and the bottom one) must be initial,

consecution: in every row (except the top and the bottom one) any two neighboring transitions must be consecutive (i.e. the next-state component of the left one must be equal to the present-state component of the right one); we represent this constraint symbolically by a rectangle like the one marked C in Figure 4,

transition: in every column, any three neighboring transitions must satisfy transition relation T; we represent this constraint symbolically by a rectangle like the one marked T in Figure 4,

termination: next-state components of all transitions in the last column (except the top and the bottom one) must be final.

An n-row string corresponds to N_{n-2} network; the first and the last row do not correspond to states of any cell in the network. Therefore they have to satisfy only transition requirements imposed by the first and the last cell in the networks (i.e. by the cells corresponding to the second and next-to-last row of the string).

We now return to the non-existence of a finite invariant. Our strategy is to search for a sequence of strings: $x_1, x_2, \ldots$ satisfying $O(x_i) \in \mathcal{L}_i$, but $O(x_i) \notin \mathcal{L}_j$ for any $j < i$. We will show that in certain cases these relations can be established in a finite number of steps. We consider only a special case when x_{i+1} is obtained by extending x_i in a certain regular fashion. If that is the case we write $x_{i+1} = \alpha(x_i)$. Next, we define precisely the extension operator α.

Given strings $s \in (S^{2n})^*$ and $t \in (S^{2n+2})^*$ such that $|t| = |s| + 1$ we say that $t = \alpha_{ik}(s)$ if the following holds:

1. $t_{x,y} = s_{x,y}$ for all $x = 1, \ldots, |s|$, $y = 1, \ldots, n$ (t obtained from s by adding one row of transitions on the top, and one column of transitions on the right),
2. $O(t_x) = \begin{cases} O(s_x) & \text{for all } x = 1, \ldots, i \\ O(s_{x-1})) & \text{for all } x = i+1, \ldots, |t| \end{cases}$ (the observable part of t is the same as the observable part of s, except that the i'th column is repeated twice),
3. $t_{|t|,x} = \begin{cases} s_{|s|,x} & \text{for all } x = 1, \ldots, k \\ s_{|s|,x-1} & \text{for all } x = k+1, \ldots, n+1 \end{cases}$ (the last column of t is the same as the last column of s, except that the k'th row is repeated twice).

Figure 5 shows an example of such a pair of strings. Full thin lines connect transition that must be equal to satisfy conditions 2 and 3 above. All of these equalities can be satisfied only if s satsify certain constraints. It is easy to check that these constraints are exactly **C1** to **C3** defined below.

In the rest of this paper we assume that all exstensions have common i and k, so without ambiguity we write $\alpha(s)$ for $\alpha_{ik}(s)$. Also, we use the following abbreviation for any $j \geq 0$:

$$\alpha^j(s) = \underbrace{\alpha(\alpha(\ldots\alpha}_{j \text{ times}}(s)\ldots)).$$

Proposition 2 *A string $\alpha(s)$ exists if and only if all of the following hold:*

C1: $s_{x,n} = s_{x,n-1}$, *for all* $x = 1, \ldots, i$

C2: $s_{x+1,n} = s_{x,n-1}$, *for all* $x = i, \ldots, |s| - 1$,

C3: $s_{x,1} = s_{i,1}$, $s_{x,2} = s_{i,2}$, *for all* $x = i, \ldots, |s|$.

If $\alpha(s)$ exists, then it is unique.

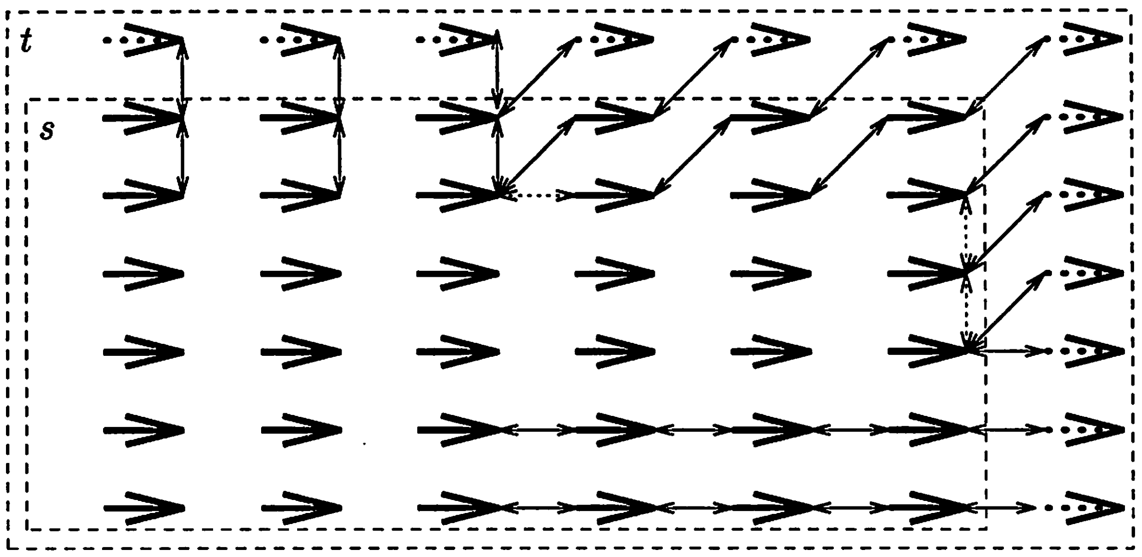

Figure 5: Strings satisfying $t = \alpha_{3,3}(s)$.

It is easy to check that $\alpha(s)$ satisfies initialization and termination requirements, if s does. Also, if s satisfies consecution requirement and:

C4: $s_{i,n-1} = s_{i+1,n-1}$,

then $\alpha(s)$ also satisfies consecution requirement. Similarly, if s satisfies transition requirements and:

C5: $s_{|s|,k} = s_{|s|,k+1} = s_{|s|,k+2}$,

then $\alpha(s)$ also satisfies transition requirement. From this point on, we do assume properties **C4** and **C5**. We make this assumption without loss of generality because they are always satisfied by $\alpha^2(s)$, which also satisfies all other restriction on s mentioned in this section. Equalities imposed by **C4** and **C5**, are shown with thin dotted line in Figure 5.

The part of our strategy is to find a sequence of strings $x_1, x_2, \ldots, x_i = \alpha(x_{i-1}), \ldots$ satisfying $x_i \in \mathcal{L}_i$. The following lemma describes a case when all of these relations are satisfied if one of them is.

Lemma 2 *Let s satisfy* C1–C5, *and let $t = \alpha(s)$. If:*

C6: $(s_{x,n-1}, s_{x,n}, t_{x,n+1}) \in T$ *for all* $x = 1, \ldots, |s|$,

C7: $s^2_{|s|,x} = t^1_{|t|,x}$ *for all* $x = 2, \ldots, n$,

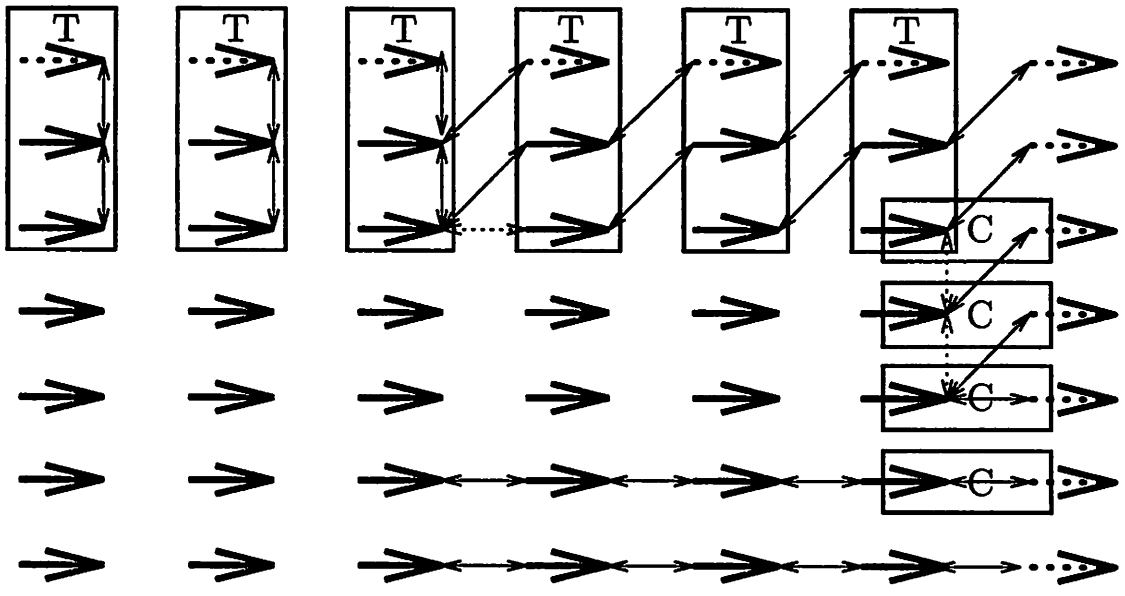

Figure 6: Illustration of Lemma 2.

then:

$$s \in \mathcal{L}(N_{n-2}) \implies t \in \mathcal{L}(N_{n-1}), \tag{5}$$

$$s \in \mathcal{L}(N_{n-2}) \implies \alpha^j(s) \in \mathcal{L}(N_{n-2+j}) \textit{ for all } j \geq 0. \tag{6}$$

Proof: Conditions **C6** and **C7** include exactly those conditions for $t \in \mathcal{L}(N_{n-1})$ to be satisfied, which are not already satisfied by $s \in \mathcal{L}(N_{n-2})$. Thus, (5) holds. If s satisfies **C6** and **C7** so does $\alpha(s)$. Therefore, we can repeatedly apply (5) to get (6). □

Figure 6 illustrates the requirements imposed by **C6** and **C7**.

The second part of our strategy is to find a sequence of strings $x_1, x_2, \ldots$ that $x_j \notin \mathcal{L}_i$ for any $j < i$. In Lemma 3 we establish a condition which enables us to prove this relation in a finite number of steps.

Lemma 3 *Let s satisfy* **C1–C5**, *and let $t = \alpha(s)$. If:*

C8: *$s_{x,n-1}$ is the unique element of the set $\{u = (u^1, u^2) | \exists v, w, z = (u^2, z^2) : (v, t_{x,n}, u) \in T \wedge (w, t_{x+1,n}, z) \in T\}$ for all $x = 1, \ldots, |s|$, and*

C9: *there exists a sequence of transition $u_{n+1}, u_n, \ldots, u_4, u_3$ such that $u_{n+1} = t_{|t|,n+1}$, and u_x (for all $x = n, n-1, \ldots, 3$) is the unique element of the set: $\{v | \exists w : (w, u_{x+1}, v) \in T\}$,*

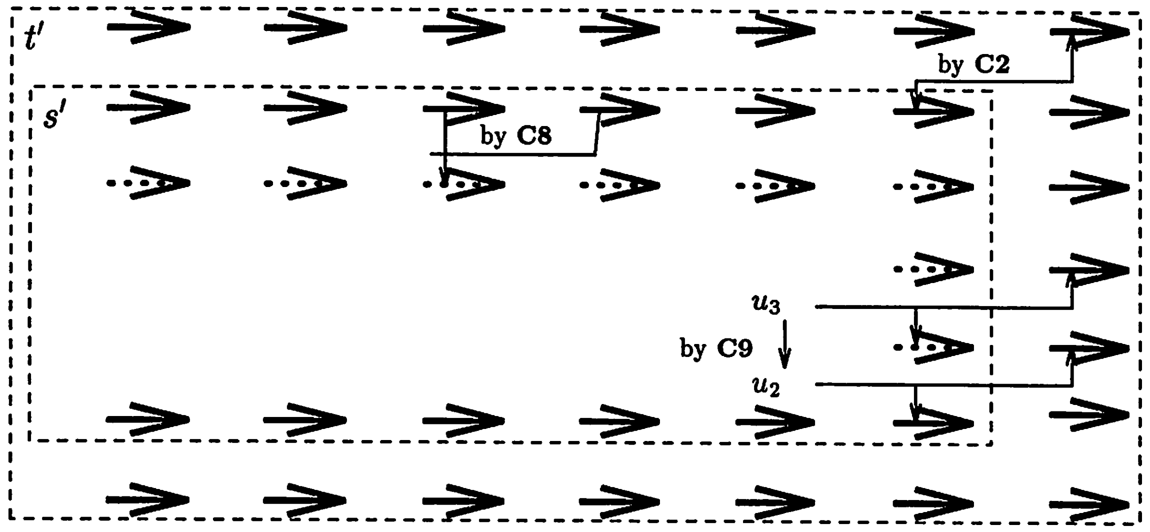

Figure 7: Illustration of Lemma 3.

then:

$$O(t) \in \mathcal{L}_{n-1} \implies O(s) \in \mathcal{L}_{n-2}, \tag{7}$$

$$O(s) \notin \mathcal{L}_{n-2} \implies O(\alpha^j(s)) \notin \mathcal{L}_{n-2+j} \text{ for all } j \geq 0. \tag{8}$$

Proof: Assume $O(t) \in \mathcal{L}_{n-1}$, let string t' be such that $t' \in \mathcal{L}(N_{n-1})$ and $O(t') = O(t)$. Also, let s' be the string obtained by removing from t' the $(n+1)$'st row and the last column. It follows from **C8** that the $(n-1)$'st row of s' is exactly equal to the $(n-1)$'st row of s. Since **C1**–**C3** also hold, we have that $O(s') = O(s)$. We claim that $s' \in \mathcal{L}(N_{n-2})$ and thus $O(s) \in \mathcal{L}_{n-2}$. That s' satisfies initialization, consecution, and transition requirements follows from the assumption: $t' \in \mathcal{L}(N_{n-1})$. It follows from **C9** that $t'_{|t'|,x} = u_x$ for all $x = n+1, \ldots, 3$, so we can conclude that all next-state components of u_x's are final states (since $t' \in \mathcal{L}(N_{n-1})$). It follows from **C2** that $s'_{|s'|,n} = t'_{|t'|,n+1} = u_{n+1}$, so we can again apply **C9** to obtain $s'_{|s'|,x} = u_{x+1}$ for all $x = n, \ldots, 2$. Thus, s' satisfies termination requirements as well.

It is easy to check that $\alpha(s)$ satisfies **C8** and **C9** if s does. Therefore, we can repeatedly apply (7) to get (8). □

Figure 7 illustrates the chain of reasoning in Lemma 3. Values of the transition in row 5 (from the bottom) are implied from transitions in row 6 by **C8**. Transitions in the last column are implied by **C9**, and they must also appear in column 6 by **C2**.

A reader will notice that the condition **C9** is used only to establish termination. If all the states of the basic cell are final, Lemma 3 holds even if **C9** is not satisfied.

We are now ready to postulate sufficient conditions for the non-existence of a finite invariant.

Theorem 4 *Let s be such that it satisfies* **C1–C9** *and let $\Omega(s) = \{O(\alpha^j(s)) | j \geq 0\}$. If s is such that:*

$$\begin{aligned} \mathcal{L}_j \cap \Omega(s) &= \emptyset, \textit{ for all } j = 1, \ldots, n-3 && (9) \\ \mathcal{L}_{n-2} \cap \Omega(s) &= \{s\} && (10) \end{aligned}$$

then:

a) *a finite invariant does not exist,*

b) $\mathcal{L}_\infty \supseteq \Omega(s)$

Proof: From (6) and (10) we have for all $j \geq 0$:

$$\alpha^j(s) \in \mathcal{L}_{n-2+j}, \tag{11}$$

but, from (10), it follows that for all $0 \leq m \leq j$:

$$\alpha^{j-m}(s) \notin \mathcal{L}_{n-2}, \tag{12}$$

so, from (6) and (12) we have:

$$\alpha^j(s) \notin \mathcal{L}_{n-2+m} \text{ for all } 0 \leq m \leq j. \tag{13}$$

Finally, we combine (11), (9), and (13), to claim that for every $j \geq 0$ there exists at least one string in $\mathcal{L}_{n-2+j}$ which is not in $\mathcal{L}_k$ for any $k < n-2+j$, so a finite invariant cannot exist. Also, part **b)** follows from (11). □

Consider Example 2 in section 4 and the following string[2] in $\mathcal{L}_3$:

$$s = \begin{bmatrix} idle \\ idle \\ idle \\ idle \\ token \end{bmatrix} \begin{bmatrix} idle \\ idle \\ idle \\ token \\ dead \end{bmatrix} \begin{bmatrix} idle \\ idle \\ token \\ dead \\ dead \end{bmatrix} \begin{bmatrix} idle \rightarrow token \\ token \rightarrow dead \\ dead \rightarrow dead \\ dead \rightarrow dead \\ dead \rightarrow dead \end{bmatrix}$$

Assume that the extension operator α_{ik} is applied with $i = 2$, and $k = 1$. It is now straightforward to check that conditions **C1–C9** are satisfied. It is also straightforward to define an automaton accepting $\Omega(s)$, thus (9) and (10) can be easily checked by a language containment checking tool. Since **C1–C9**, (9) and (10) are all satisfied we conclude that a finite invariant does not exist and that $\mathcal{L}_\infty \supseteq (\mathcal{L}_1 \cup \mathcal{L}_2 \cup \Omega(s))$.

The following procedure shows how Theorem 4 can be used to search for an invariant:

[2]For $x < 4$, we omit writing only $s^2_{x,y}$ and assume that $s^2_{x,y} = s^1_{x+1,y}$.

1. let $A = \langle N_1 \rangle$,

2. choose a string s and let n denote its "width" (i.e. $s \in (S^{2n})^*$),

3. let $A = A \oplus \bigoplus_{i=1}^{n-2} \langle N_i \rangle$; **if** $A \cdot C \leq A$ **then** *HALT*,

4. **if** s satisfies **C1–C9**, (9) and (10) for some α_{ik}, where $1 \leq i \leq |s|$, $1 \leq k \leq n$ **then** let $A = A \oplus \Omega(s)$,

5. **if** $A \cdot C \leq A$ **then** *HALT* **else** go to step 2.

If the procedure terminates it will generate a tight (but possibly not finite) invariant A. Unfortunately, we can not claim that the procedure will terminate even if a tight invariant exists and all strings are systematically enumerated in step 2. It might be more efficient to apply this procedure interactively, i.e. to let the user choose a string and then execute other steps automatically.

6 Conclusions

We have studied the existence of a finite invariant of an iterative system consisting of many identical automata. We have shown that the if we allow constraints on the environment, the existence is undecidable, but we have also pointed out that the proof does not exist presently for the case of an unconstrained environment. We have presented a semi-decision procedure that will generate a finite invariant, if one exists. We have also provided sufficient conditions for the non-existence of the finite invariant. Those conditions can be checked in finite time so a semi-decision procedure can be defined that will recognize a pattern satisfying those conditions, if such a pattern exists. These results can then be used in a search for an invariant that is possibly tight but not finite. It is possible that neither of these procedures terminate. This is consistent with the decidability result (at least for closed systems).

This work can be naturally extended in several ways. From the theoretical point of view, the decidability of existence of a finite invariant for open iterative systems needs to be studied. From the practical point of view, it would be useful to generalize the conditions for non-existence. This can be done by analyzing sequences of strings where not only a single element, but a whole substring is repeated many times. It is also possible to construct cases where the non-existence can be proved by analyzing a sequence of sets of string, rather then just a sequence of strings. Finally, in order to verify liveness properties, these results need to be extended to the automata on infinite tapes.

References

[BCG89] M.C. Browne, E.M. Clarke, and O. Grumberg. Reasoning about networks with many identical finite state processes. *Information and Computation*, 81(1):13–31, 1989.

[CGH$^+$93] E. M. Clarke, O. Grumberg, H. Hiraisi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness. Verification of the Futurebus+ cache coherence protocol. In *Proc. 11th Intl. Symp. on Comput. Hardware Description Lang. and their Applications*, 1993.

[Hen61] Frederick C. Hennie. *Iterative Arrays of Logical Circuits.* MIT Press and John Eiley Sons, Inc., 1961.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, languages and Computation.* Addison Wesley, 1979.

[KM89] R. P. Kurshan and K. L. McMillan. A structural induction theorem for processes. In *Proceedings of the 8th ACM Symp. PODC*, 1989.

[RS92] J.K. Rho and F. Somenzi. Inductive verification of iterative systems. In *Proceedings of the 29th ACM/IEEE Design Automation Conference*, pages 628–33, June 1992.

[RS93] J.K. Rho and F. Somenzi. Automatic generation of network invariants for the verification of iterative sequential systems. In Costas Courcoubetis, editor, *Computer Aided Verification: 5th International Conference, CAV'93, Elounda, Greece, June/July 1993, Proceedings*, pages 123–137. Springer-Verlag, 1993. LNCS vol. 697.

[SG90] Z. Shtadler and O. Grumberg. Network grammars, communication behaviors and automatic verification. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems, International Workshop Proceedings, Grenoble, France, 12-14 June 1989*, pages 151–65. Springer-Verlag, 1990. LNCS vol. 407.

[WL90] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems, International Workshop Proceedings, Grenoble, France, 12-14 June 1989*, pages 68–80. Springer-Verlag, 1990. LNCS vol. 407.