On the Automatic Computation of Network Invariants

Felice Balarin* and Alberto L. Sangiovanni-Vincentelli

Department of Electrical Engineering and Computer Science
University of California, Berkeley, CA, USA 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 study the decidability of the existence of such an invariant, 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 [3]). 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 [4], only recently there has been a significant interest in the formal verification of such systems. Browne, Clarke and Grumberg [2], and Shtadler and Grumberg [8] have studied conditions under which the satisfaction of formulae of certain temporal logics is independent of the size of the system. In [8] the conditions seem to be quite restrictive, while in [2] the conditions cannot in general be checked automatically.

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

Wolper and Lovinfosse [9] have studied formal verification of iterative systems generated by interconnecting identical processes in certain regular fashion. They also present some decidability results for related problems. Kurshan and McMillan [5] 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 [6, 7], 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 there exists a *finite* subclass such that any behavior exhibited by any system in the class is also exhibited by some system in the 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.

In this paper we consider only networks of chain structure, i.e. every component can communicate only with its left and right neighbors. Also, we consider only automata on finite tapes. Thus, using our approach only safety properties can be verified.

The rest of the paper is organized as follows. In Sect. 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 Sect. 3 we illustrate these concepts on typical examples. The focus of our paper is the computation of the finite invariant presented in Sect. 4, 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 a state of the basic cell is fully observable and that transitions of the basic cell can depend on the present and next states of its left and right neighbors (see Fig. 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 [9] and the ever-so-popular Dining Philosophers Problem (e.g. [5]).

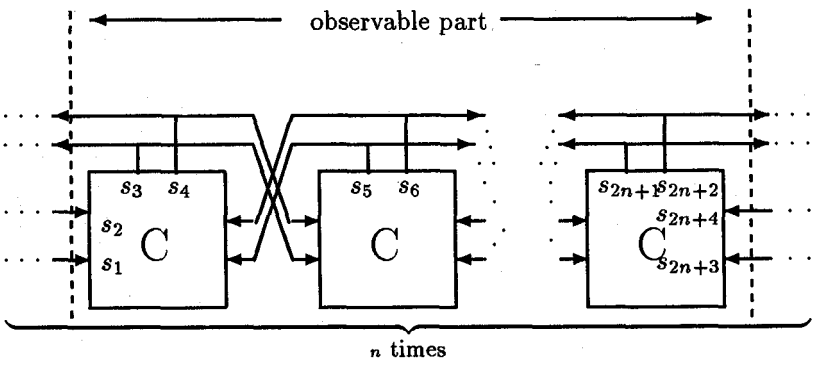

Fig. 1. An open network N_n.

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 $i = 1, \ldots, k$.

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:

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

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 Fig. 2.

To formalize connecting cells into larger units in [1] we define *concatenation* "$\cdot$". Given two automata A and B the automaton the automaton $A \cdot B$ is well defined only if the alphabet of A is S^n and the alphabet of B is S^{k-n+4}, where

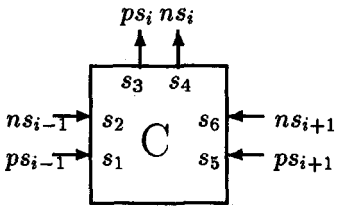

Fig. 2. A basic cell; *ps* and *ns* denote present and next states.

S is the state space of the basic cell and $k \geq n \geq 4$. The alphabet of $A \cdot B$ if S^k, and $\mathcal{L}(A \cdot B)$ satisfies the following:

$$\left(\begin{bmatrix} s_{1,1} \\ \vdots \\ s_{1,n-3} \\ \vdots \\ s_{1,n} \\ \vdots \\ s_{1,k} \end{bmatrix} \cdots \begin{bmatrix} s_{|s|,1} \\ \vdots \\ s_{|s|,n-3} \\ \vdots \\ s_{|s|,n} \\ \vdots \\ s_{|s|,k} \end{bmatrix} \right) \in \mathcal{L}(A \cdot B) \text{ if and only if}$$

$$\left(\begin{bmatrix} s_{1,1} \\ \vdots \\ s_{1,n} \end{bmatrix} \cdots \begin{bmatrix} s_{|s|,1} \\ \vdots \\ s_{|s|,n} \end{bmatrix} \right) \in \mathcal{L}(A) \text{ and } \left(\begin{bmatrix} s_{1,n-3} \\ \vdots \\ s_{1,k} \end{bmatrix} \cdots \begin{bmatrix} s_{|s|,n-3} \\ \vdots \\ s_{|s|,k} \end{bmatrix} \right) \in \mathcal{L}(B) \ .$$

Given a cell $C = (S, I, T, F)$ and an automaton E over alphabet S^4 (called an *environment*) a *network* of length n is defined by:

$$N_n = E \cdot \underbrace{C \cdot C \cdot \ldots \cdot C}_{n \text{ times}} \ .$$

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

To compose networks into larger ones, only "peripheral" components of their languages need to be considered (see Fig. 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 and languages (for technical details see [1]). For clarity, we write $\mathcal{L}_n$ instead of $O(\mathcal{L}(N_n))$.

An *iterative system* is the class of all networks of different length generated by the same cell and environment. If $\{N_n | n \geq 1\}$ is an iterative system and $\mathcal{L}_\infty = \bigcup_{n=1}^{\infty} \mathcal{L}_n$, then an *invariant* is any finite-state automaton A over alphabet S^8 satisfying $\mathcal{L}(A) \supseteq \mathcal{L}_\infty$. 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*.

Obviously, a tight invariant exists if and only if $\mathcal{L}_\infty$ is regular. One may try to find a "tightest regular invariant", i.e. 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 exists). 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.

3 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. In all examples, all states are final and 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:

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

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:

$$\begin{aligned} &\mathit{idle} \longrightarrow \mathit{idle} : \quad \text{if } ps_{n-1} = \mathit{idle} \\ &\mathit{idle} \longrightarrow \mathit{token} : \text{ if } ps_{n-1} = \mathit{token} \\ &\mathit{token} \longrightarrow \mathit{dead} : \text{if } ps_{n-1} = \mathit{dead} \\ &\mathit{dead} \longrightarrow \mathit{dead} : \text{ if } ps_{n-1} = \mathit{dead} \end{aligned}$$

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 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.

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.

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

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.

4 Computing a Finite Invariant

4.1 Decidability and Existence

In this section we give some results on decidability as well as a simple semi-decision procedure that will find a finite invariant if one exists. Due to the space limitations we omit the proofs. They can be found in the extended version of this paper [1].

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

The proof is by reduction of the finite memory problem for Turing machines. At present, it is not clear whether that proof can be extended to open systems. In fact, this result is similar to Theorem 4.3 in [9] and Theorem 4 in [4]. 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 [9] the first cell is explicitly distinguished, and in [4] 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.

Theorem 2. *Let* $\{N_n | n \geq 1\}$ *be an iterative system, and let* A *be some automaton. If* $\mathcal{L}_1 \subseteq \mathcal{L}(A)$ *and* $O(\mathcal{L}(A \cdot C)) \subseteq \mathcal{L}(A)$, *then* A *is an invariant of* $\{N_n | n \geq 1\}$.

The proof is by induction. Both Kurshan and McMillan [5] and Wolper and Lovinfosse [9] 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 to language containment. 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 of an iterative system* $\{N_n | n \geq 1\}$ *exists if and only if there exists* $n^* < \infty$ *such that:*

$$\mathcal{L}_{n^*+1} \subseteq \bigcup_{n=1}^{n^*} \mathcal{L}_n \ . \tag{1}$$

In fact, a stronger claim follows from the proof of Theorem 3: if (1) holds, then an automaton with the language $\bigcup_{n=1}^{n^*} \mathcal{L}_n$ is an invariant. This immediately gives us the following semi-decision procedure:

for every integer n^*: **if** (1) holds **then** HALT.

If the procedure terminates, it will produce n^* which can be used to construct a finite invariant with the language $\bigcup_{n=1}^{n^*} \mathcal{L}_n$. However, if a finite invariant does not exist, the procedure will not terminate.

4.2 Proving Non-existence of Finite Invariant

In this section, we show a sufficient condition for non-existence of 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 to parts of the string we use the naming scheme detailed in Fig. 3. We call a pair $s_{x,y} = (s^1_{x,y}, s^2_{x,y})$ a

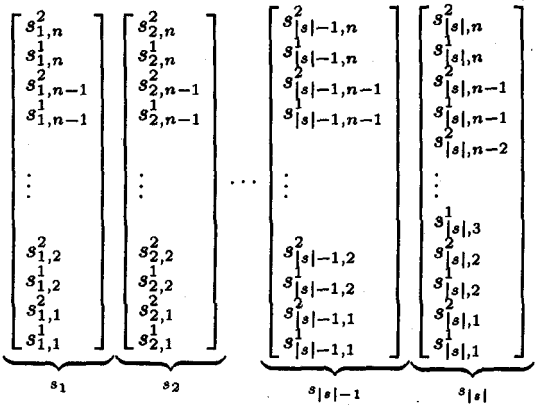

Fig. 3. A naming scheme for parts of the string.

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:

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

To prove the non-existence of a finite invariant, we 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 obtained from s by adding one row and one column of transitions, i.e.:

$$t_{x,y} = s_{x,y} \text{ for all } x = 1, \ldots, |s|, \ \ y = 1, \ldots, n \ , \tag{2}$$

2. the observable part of t is the same as the observable part of s, except that the i'th column is repeated twice, i.e.:

$$O(t_x) = \begin{cases} O(s_x) & \text{for all } x = 1, \ldots, i \ , \\ O(s_{x-1})) & \text{for all } x = i+1, \ldots, |t| \ , \end{cases} \tag{3}$$

3. the last column of t is the same as the last column of s, except that the k'th row is repeated twice, i.e.:

$$t_{|t|,x} = \begin{cases} s_{|s|,x} & \text{for all } x = 1, \ldots, k \ , \\ s_{|s|,x-1} & \text{for all } x = k+1, \ldots, n+1 \ . \end{cases} \tag{4}$$

In the rest of this paper we assume that all extensions 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 4. *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.

Lemma 5. *If $s \in \mathcal{L}(N_{n-2})$, and s satisfies:*

C4: $s_{i,n-1} = s_{i+1,n-1}$,
C5: $s_{|s|,k} = s_{|s|,k+1} = s_{|s|,k+2}$,

then $t = \alpha(s)$ satisfies:

$$t^2_{x,y} = t^1_{x+1,y} \text{ for all } x = 1, \ldots, |s| - 1, \ y = 2, \ldots, n \ , \quad (5)$$
$$t^1_{1,x} \in I \text{ for all } x = 2, \ldots, n \ , \quad (6)$$
$$t^1_{|t|,x} \in F \text{ for all } x = 2, \ldots, n \ , \quad (7)$$
$$(t_{x,y-1}, t_{x,y}, t_{x,y+1}) \in T \text{ for all } x = 1, \ldots, |t|, y = 2, \ldots, n-1 \ . \quad (8)$$

Proof. By definition, the assumption $s \in \mathcal{L}(N_{n-2})$ is equivalent to:

$$s^2_{x,y} = s^1_{x+1,y} \text{ for all } x = 1, \ldots, |s| - 1, \ y = 2, \ldots, n-1 \ , (9)$$
$$s^1_{1,x} \in I \text{ for all } x = 2, \ldots, n-1 \ , \quad (10)$$
$$s^1_{|s|,x} \in F \text{ for all } x = 2, \ldots, n-1 \ , \quad (11)$$
$$(s_{x,y-1}, s_{x,y}, s_{x,y+1}) \in T \text{ for all } x = 1, \ldots, |s|, y = 2, \ldots, n-1 \ . \quad (12)$$

Now, (5) follows from (2), (4), **C4**, and (9); (6) follows from (2), (4), and (10); (7) follows from (3), and (11); and finally (8) follows from (2), (3), **C5**, and (12). □

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.

The part of our strategy is to find a sequence of strings $x_1, x_2, \ldots$ 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 6. *Let s satisfy* **C1**–**C5**, *and let $t = \alpha(s)$. If:*

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

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

then:

$$s \in \mathcal{L}(N_{n-2}) \Longrightarrow t \in \mathcal{L}(N_{n-1}) \ , \tag{13}$$

$$s \in \mathcal{L}(N_{n-2}) \Longrightarrow \alpha^j(s) \in \mathcal{L}(N_{n-2+j}) \text{ for all } j \geq 0 \ . \tag{14}$$

Proof. Conditions (5)–(8), **C6** and **C7** are exactly the conditions for $t \in \mathcal{L}(N_{n-1})$ to be satisfied. Thus, (13) holds. If s satisfies **C6** and **C7** so does $\alpha(s)$. Therefore, we can repeatedly apply (13) to get (14). □

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 7 we establish a condition which enables us to prove this relation in a finite number of steps.

Lemma 7. *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* $u_n, \ldots, u_4, u_3$ *such that* $u_n = t_{|t|,n}$, *and* u_x *(for all* $x = n-1, \ldots, 3$*) is the unique element of the set:* $\{v | \exists w : (w, u_{x+1}, v) \in T\}$,

then:

$$O(t) \in \mathcal{L}_{n-1} \Longrightarrow O(s) \in \mathcal{L}_{n-2} \ , \tag{15}$$

$$O(s) \notin \mathcal{L}_{n-2} \Longrightarrow O(\alpha^j(s)) \notin \mathcal{L}_{n-2+j} \text{ for all } j \geq 0 \ . \tag{16}$$

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 conditions analog to (9), (10), and (12) follows directly from $t' \in \mathcal{L}(N_{n-1})$. It follows from **C9** that $t'_{|t'|,x} = u_x$ for all $x = n, \ldots, 3$, so $t' \in \mathcal{L}(N_{n-1})$ also implies $u^1_x \in F$, for all $x = 3, \ldots, n$. It follows from **C2** that $s'_{|s'|,n-1} = t'_{|t'|,n} = u_n$, so we can again apply **C9** to obtain $(s'_{|s'|,x})^1 = u^1_{x+1} \in F$, for all $x = n-1, \ldots, 2$.

It is easy to check that $\alpha(s)$ satisfies **C8** and **C9** if s does. Therefore, we can repeatedly apply (15) to get (16). □

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 7 holds even if **C9** is not satisfied.

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

Theorem 8. *If s is such that it satisfies* **C1–C9** *and:*

$$s \in \mathcal{L}(N_{n-2}) \ , \tag{17}$$

$$O(\alpha^j(s)) \notin \mathcal{L}_{n-2}, \text{ for all } j > 0 \ , \tag{18}$$

$$O(\alpha^j(s)) \notin \mathcal{L}_m, \text{ for all } j \geq 0, \ m = 1, \ldots, n-3 \ , \tag{19}$$

then:

a) *a finite invariant does not exist,*
b) $\mathcal{L}_\infty \supseteq \{O(\alpha^j(s)) | j \geq 0\}$.

Proof. From (17) and Lemma 6 we have:

$$O(\alpha^j(s)) \in \mathcal{L}_{n-2+j} \text{ for all } j \geq 0 \ . \tag{20}$$

We can rewrite (18) as $O(\alpha^{j-m}(s)) \notin \mathcal{L}_{n-2}$ for all $j > 0$, $0 \leq m < j$, and combine it with Lemma 7 to get:

$$O(\alpha^j(s)) = O(\alpha^m(\alpha^{j-m}(s))) \notin \mathcal{L}_{n-2+m} \text{ for all } j > 0, \ 0 \leq m \leq j \ . \tag{21}$$

Thus, for every $j \geq 0$ there exists a string (specifically $O(\alpha^j(s))$) in $\mathcal{L}_{n-2+j}$ (by (20)), which is not in $\mathcal{L}_m$ for any $m < n-2+j$ (by (19) and (21)), so a finite invariant cannot exist. Also, part **b)** follows from (20). □

Consider Example 2 in section 3 and the following string[2] in $\mathcal{L}_3$:

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

It is straightforward to check that conditions **C1–C9** are satisfied for $\alpha_{2,1}$. It is also straightforward to define an automaton accepting $\{O(\alpha_{2,1}^j(s)) | j \geq 0\}$, thus (17)–(19) can be easily checked by a language containment checking tool. Since **C1–C9**, and (17)–(19) are all satisfied we conclude that a finite invariant does not exist and that $\mathcal{L}_\infty \supseteq (\mathcal{L}_1 \cup \mathcal{L}_2 \cup \{O(\alpha_{2,1}^j(s)) | j \geq 0\})$.

The following procedure shows how Theorem 8 can be used to search for an invariant:

1. construct A s.t. $\mathcal{L}(A) = \mathcal{L}_1$,
2. choose a string s and let n denote its "width" (i.e. $s \in (S^{2n})^*$),
3. construct B s.t. $\mathcal{L}(B) = \mathcal{L}(A) \cup \bigcup_{i=1}^{n-2} \mathcal{L}_i$; let $A := B$,
4. **if** $O(\mathcal{L}(A \cdot C)) \subseteq \mathcal{L}(A)$ **then** *HALT*,
5. **if** s satisfies **C1–C9**, (17)–(19) for some α_{ik}, where $1 \leq i \leq |s|$, $1 \leq k \leq n$ **then** construct B s.t. $\mathcal{L}(B) = \mathcal{L}(A) \cup \{O(\alpha_{ik}^j(s)) | j \geq 0\}$; let $A := B$,
6. **if** $O(\mathcal{L}(A \cdot C)) \subseteq \mathcal{L}(A)$ **then** *HALT* **else** go to step 2.

[2] We omit writing $s_{x,y}^2$ for $x < 4$, and assume that $s_{x,y}^2 = s_{x+1,y}^1$.

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.

5 Conclusions

We have studied the existence of the finite invariant of an iterative system consisting of many identical automata. We have shown that if constraints on the environment are allowed, 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 a 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

1. Felice Balarin and Alberto L. Sangiovanni-Vincentelli. On the automatic computation of network invariants, 1994. UCB/ERL M94/18.
2. 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.
3. 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.
4. Frederick C. Hennie. *Iterative Arrays of Logical Circuits.* MIT Press and John Eiley Sons, Inc., 1961.
5. R. P. Kurshan and K. L. McMillan. A structural induction theorem for processes. In *Proceedings of the 8th ACM Symp. PODC*, 1989.
6. 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.

7. 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.
8. 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.
9. 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.