Parametrized Verification of Linear Networks Using Automata as Invariants *

A. Prasad Sistla[1]

Electrical Engineering and Computer Science Department, The University of Illinois at Chicago, USA

1 Introduction

Recently there has been much interest in parametrized verification, i.e., verification of a family of systems $\{P_i\}_{i=1}^{\infty}$, where P_i is a system consisting of i number of processes, against a specification given in temporal logic or by an automaton. Such interest is motivated by the fact that many algorithms in practice are designed to work with arbitrary number of processes.

In general, the problem of determining if a family of networks of similar processes satisfies a temporal logic specification is undecidable [AK86]. Nevertheless, automated and semi-automated methods for verification for restricted classes of parametrized systems have been proposed in literature. The works of [SG87, GS92, EN96] present fully automated methods for systems composed of a single control process and an arbitrary number of identical client processes.

One of the semi-automated methods is to show that for certain class of problems (specifically, rings of arbitrary size or client-server problesm) there exists a k such that correctness of families of networks upto size up to k implies the correctness of networks of all sizes; this has bee done in the works of [EN95, GS92].

An alternate method, which we use, is to use induction on the number of processes. Roughly speaking, this method, when applied to a linear network composed of an arbitrary number of processes P works as follows. An inductive invariant I, specified as another process, is obtained. The induction step is shown by establishing that the process $I \odot P \leq P$ where $\odot$ is the composition operator, $\leq$ is an appropriate monotonic pre-order on processes. The basis step is obtained by showing that the system composed of a small number of processes is less than or equal to I in the pre-order. The correctness of the family of systems is established by showing that I itself satisfies the correctness property. All the three steps can be automated if I is a finite state process. The above approach has been taken in [WL89, KM89, BCG89].

Although the above approach of specifying the invariant as another process is elegant, it has the following drawback. It is difficult to specify invariants that involve predicates on global states as well as predicates on communication patterns. To overcome this problem, Abstract Transition Systems (ATS) were

* This work is partly supported by the NSF grants CCR-9623229, CCR-9212183, CCR-9633536

employed in [CGJ95] to specify the invariant. An abstract transition system consists of abstract states and transitions between the abstract states. An abstract state is specified by a regular expression or automaton that denotes a predicate on the global states of systems with arbitrary number of processes. Thus, ATS uses two different formalisms— the regular expressions to specify properties of global states and the transitions in the ATS to specify computation steps.

In this paper, we propose a unified formalism based on automata on two dimensional strings to specify the inductive invariant. Use of such automata is based on the observation that the computation of a linear network of n processes can be looked as a two dimensional string. One dimension is the time dimension which is infinite and the other is the space dimension consisting of the states of processes at any particular time. An automaton on two dimensional string takes as input the states and transitions of individual processes from the two dimensional string where these inputs are generated by scanning the string from left to right in the space dimension, and bottom to top in the time dimension. The automaton accepts the string by going through a final state infinitely often.

In our approach, the inductive invariant is specified by a finite state automaton A on two dimensional strings. The set of computations accepted by the automaton denotes the inductive invariant. We show how to compose an automaton with a process P to obtain another automaton $A \odot P$. The inductive step is proved by showing that $L(A \odot P)$ is contained in $L(A)$ where $L(A)$ is the set of two dimensional strings accepted A. This method is shown to be sound and complete for verifying correctness of families of linear networks (as well as circular networks) of the form $I \odot P^i \odot E$ (for all $i \geq 0$) where I, E and P are processes. The completeness result we prove is a semantic completeness result. It is for the first time that such a completeness result has been given for induction based proof systems for parametrized networks.

In general checking if $L(A_1) \subseteq L(A_2)$ is an undecidable problem. However, one can show that $L(A_1) \subseteq L(A_2)$ by exhibiting a simulation relation between the states of A_1 and A_2, or by showing that $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ where $\mathcal{L}(A_i)$ is the set of one dimensional strings accepted by A_i. While automatically checking for the existence of simulation relations can be done efficiently, the problem of checking if $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ can be done in exponential time using traditional automata theoretic approaches. We illustrate our approach by simple examples.

The above approach is extended for verifying correctness under fairness as well. For this we use generalized Buchi automata [GW91] and define fair composition of such an automaton with a process. The inductive invariants are specified by generalized Buchi automata. We show that this induction based approach for verifying correctness under fairness is sound. However, we do not have a completeness theorem for this case. To the best of our knowledge, all the earlier induction based approaches did not consider fairness.

Our paper is organized as follows. Section 2 defines the automata that we plan to use and the composition of an automata with a process. Section 3 presents the inductive approach for linear and circular networks. It also presents the induction based approach for correctness under fairness. It contains examples illustrating the approach. Section 4 contains concluding remarks.

2 Definitions

Automata. A Buchi automaton on two dimensional strings is defined exactly similar to the way a Buchi automaton on infinite strings is defined. The only difference is that the input alphabet to the two dimensional automaton has a special structure. Each input symbol to the automaton is of the form (u, a, d, u') where u and u' are process states, a is an action symbol which can be an element from $\Delta \cup \{\epsilon, i\tau, c\tau\}$ and d is an indicator denoting if the process is an internal process or a boundary process (if so which). Here u denotes the current local state of the process being scanned by the automaton and u' is the state of the same process in the next global state. If the action symbol a is ϵ then it indicates that the process does not make any transition in the current computational step; if a is $i\tau$ then it indicates that the process makes an internal transition (i.e., no communication) in the current computational step; if a is $c\tau$ then it denotes a synchronized communication with a neighboring process in the current computational step; if a is in Δ then it indicates that the process offers a communication denoted by a to the external world and only boundary processes can offer such communication.

Throughout this section, we assume that S is a set of process states and the states of all the processes are drawn from this set. We also assume that Δ, Δ' are communication alphabets that do not contain the symbols $\epsilon, i\tau, c\tau$, and that they contain complementary action/communication symbols; for an action symbol a, we let $\overline{a}$ denote the complement of a; furthermore, if $b = \overline{a}$ then $\overline{b} = a$.

Formally, an automaton A over Δ, is a 5-tuple $(Q_0, \Sigma, Q, \delta, F)$ where $\Sigma = S \times (\Delta \cup \{i\tau, c\tau, \epsilon\}) \times \{\textit{left}, \textit{right}, \textit{internal}\} \times S$ is the set of input symbols as indicated above, Q is the set of automaton states, Q_0 is the set of initial states, $\delta \subseteq Q \times \Sigma \times Q$ is the next state relation and $F \subseteq Q$ is the set of final states.

Let n be a positive integer. A two-dimensional string σ of width n over an alphabet Δ is an infinite alternating sequence $s_0, a_0, s_1, a_1, ..., s_i, a_i, ...$ where $s_i = (s_{i,0}, s_{i,1} ..., s_{i,n-1})$ is an n-tuple of process states and $a_i = (a_{i,0}, a_{i,1}, ..., a_{i,n-1})$ is an n-tuple of process actions where each $a_{i,j} \in \Delta \cup \{\epsilon, i\tau, c\tau\}$ (Note that the representation we used here is a row major representation of a 2-d string; this representation does not cause any loss of information). Each such string denotes a computation in which $s_{i,j}$ denotes the local state of process j in the global state s_i, and $a_{i,j}$ denotes the action taken by process j in the computational step from s_i to s_{i+1}.

A linearization of a 2-d string is obtained by scanning the 2-d string row by row, left to right, bottom to top, and outputting all quadruples of the form (s, a, d, s') where s and s' are the current and next states of the process being scanned, a is the action executed by the process and d is an indicator denoting the position of the process. Formally, a linearization $l(\sigma)$ of a 2-d string $\sigma = s_0, a_0, s_1, a_1, ...s_i, a_i, ...$, as given above, is the infinite sequence $u_{0,0}, u_{0,1}, ..., u_{0,n-1}, ..., u_{i,0}, u_{i,1}, ..., u_{i,j}, ...$ where $u_{i,j} = (s_{i,j}, a_{i,j}, d_{i,j}, s_{i+1,j})$ and

for $0 \leq i < \infty$, $d_{i,0} = \mathit{left}$, $d_{i,n-1} = \mathit{right}$, and for $0 < j < n-1$, $d_{i,j} = \mathit{internal}$. Here $d_{i,j}$ denotes whether j is the left boundary process , right boundary process or an internal process.

Let $\mathcal{L}(A)$ denote the set of all strings from Σ^ω accepted by the Buchi automaton A when A is considered as an automaton over infinite strings. We say that a 2-d string σ is accepted by the automaton A if $l(\sigma) \in \mathcal{L}(A)$, i.e., the linearization of σ is accepted by the automaton A when A is considered as an automaton on infinite strings. For an automaton A, we let $L(A)$ denote the set of all 2-d strings accepted by A.

Composition of a 2-d string and a Process. Now, we define the composition of a 2-d string with a process. This is needed in order to define a composition of an automaton with a process. A process over an alphabet of actions Δ is a triple (S, R, S_0) where S is a set of states, $R \subseteq S \times (\Delta \cup \{i\tau\}) \times S$ is a set of transitions and S_0 is the set of initial states.

Let σ be a 2-d string of width n over an alphabet Δ and P be a process over the alphabet Δ'. Let $\Delta'' = (\Delta \cup \Delta') - (\Delta \cap \Delta')$. Now, we define the right composition of σ and P, denoted as $\sigma \odot P$, to be a set of 2-d strings of width $n+1$ over the alphabet Δ'' as follows. Intuitively, a 2-d string δ is in $\sigma \odot P$ if it is obtained by fusing the 2-d string σ with with a computation of P where the fusion is carried out by synchronizing on complementary actions. The formal definition of $\sigma \odot P$ is is given in the full paper [Si97].

Composition of an Automaton with a Process. Let A be an automaton over Δ and $P = (S, R, S_0)$ be a process over Δ'. We define an automaton $A \odot P$ over (S, Δ''), where $\Delta'' = \Delta \cup \Delta' - \Delta \cap \Delta'$ as follows. We call $A \odot P$ to be the right composition of A with P. As we will show later $L(A \odot P)$ will be equal to the union of $\sigma \odot P$ where the union is taken over all $\sigma \in L(A)$.

Intuitively, $A \odot P$ works as follows. When it runs on a two dimensional string, as it goes from left to right, it behaves like A until it reaches the right end, at the right end it will model possible synchronization of action with P and move one more position to the right and possibly simulate a transition of P, after this it will move to the left of the next top row and repeat this process. It knows that it is at the right end if it gets an input symbol of the form $(t, a, \mathit{right}, t')$ where *right* denotes the right end.

Formally, $A \odot P = (Q_0', \Sigma', Q', \delta', F')$ and $\Sigma' = S \times \Delta'' \times \{\mathit{left}, \mathit{right}, \mathit{internal}\} \times S$ where Δ'' is as given above. Each state in Q' is a 5-tuple of the form $(q, s, \mathit{flag1}, \mathit{flag2}, \mathit{flag3}, a)$ where q, s are the states of A and P respectively, *flag1*, *flag2*, *flag3* are binary flags and a is an action symbol. Here *flag1*$=1$ indicates that the next computational step will be caused by a transition of P which is either an internal transition or a transition offering an action in $(\Delta' - \Delta)$. If *flag1*$=1$ then all first n processes in the computational step will not change states and only process P which is the $(n+1)st$ process will make the above type of transition. *flag2*$=1$ indicates that the automaton has scanned the first n states in the current row; this implies that the next state scanned will be that of the right

most process which is process P. When *flag3*= 1 at that time *flag2* will also have value 1; this case indicates that the right most process, i.e., process P, should make a transition offering action a; in this case $a \in \Delta \cap \Delta'$ and this occurs when synchronization on complementary action occurs between P and the n^{th} process. The formal and complete definition of the $A \odot P$ can be found in [Si97].

Theorem 1. *A 2-d string δ of width $(n+1)$ over the alphabet Δ'' is accepted by the automaton $A \odot P$ iff there exists a 2-d string σ of width n over Δ such that $\delta \in \sigma \odot P$, i.e., $L(A \odot P) = \bigcup_{\sigma \in L(A)} \sigma \odot P$.*

We define similarly left composition $P \odot \sigma$ of a process P with a 2-d string σ, and the left composition $P \odot A$ of a process P with an automaton A. We also define left-right composition of a process P with a 2-d string σ and also with a process P, denoted by $P \otimes \sigma$ and $P \otimes A$ respectively. In this case both P and σ and also A should be on the same action alphabet. $P \otimes \sigma$ forces every action of P to be synchronized with a complementary action in σ and vice versa; thus, the resulting set of 2-d strings do not offer any communication to the external world. $P \otimes A$ is defined similarly. The input alphabet of the resulting automaton $P \otimes A$ has only input symbols of the form (s, a, d, s') where $a \in \{\epsilon, i\tau, c\tau\}$ and $d = internal$. The automata $P \odot A$ and $P \otimes A$ satisfy similar properties as given by Theorem 1.

3 Verification of Linear Networks Using Induction

We consider linear networks of processes in which adjacent processes communicate using CCS/CSP type of actions. We assume that the communication actions of processes are drawn from a set Δ which consists of actions of the from $a?$ and $a!$ which are called complemented pair of actions. Communication occurs when two adjacent processes execute complementary actions. In this section, we show how we can verify the correctness of such linear networks and circular networks using automata as invariants. Let Δ be an action alphabet.

3.1 Linear Networks

In a linear network, each process can communicate only with its left and right neighbors. To model this, we assume that the actions a process uses are of the form *left.a* and *right.a* where $a \in \Delta$. Let $\Delta' = \{left.a, right.a : a \in \Delta\}$.

Let $P_0, P_1, ..., P_{n-1}$ be a set of processes with action symbols taken from Δ'. We define the linear composition $P_0 \odot P_1 \odot ... \odot P_{n-1}$, called a *linear chain*, by using CCS composition operator as follows. For each i, such that $0 \leq i < (n-1)$, we rename all actions of the form *right.a* in P_i to a_i, and for each i such that $0 < i \leq n-1$, we rename all actions of the form *left.a* in P_i to a_{i-1}. Note that actions of the form *left.a* in P_0 and actions of the form *right.a* in P_{n-1} are not renamed. Let the resulting processes be $P'_0, P'_1, ..., P'_{n-1}$ (Note that the $\overline{a_i}$ is the complement of a_i). Now we define the linear chain $P_0 \odot P_1 \odot ...P_{n-1}$ to be same as $P'_0 \circ P'_1 ... \circ P'_{n-1}$ where $\circ$ is the traditional CCS/CSP composition operator.

The set of infinite computations of the chain of processes, denoted as $\mathcal{C}(P_0 \odot ... \odot P_{n-1})$ is the set of 2-d strings $\sigma = u_0, a_0, u_1, a_1, ...u_i, a_i, ...$ of width n over the alphabet Δ' satisfying the following properties: for each j, $u_{0,j}$ is an initial state of process P'_j and for each $i \geq 0$, the computational step (u_i, a_i, u_{i+1}) either involves an internal transition of process j, for some $0 \leq j < n$, and in this case $a_{i,j}$ is $i\tau$, or it involves synchronized communication involving two adjacent processes j and $j+1$ and in this case $a_{i,j}$ and $a_{i,j+1}$ are both $c\tau$. If a process j is not involved in the computational step (u_i, a_i, u_{i+1}) then $a_{i,j}$ is ϵ and $u_{i+1,j} = u_{i,j}$.

Let P, I and E be a processes over the alphabet Δ' such that all the actions in I are of the form *right.a* and all the actions in E are of the form *left.a* where $a \in \Delta$, i.e., I and E do not offer communication to the left and right respectively. Consider a computation of a linear chain of the form $I \odot P^i \odot E$ for some $i > 0$, where P^i denotes the composition $P \odot ... \odot P$ taken i times. All the action symbols appearing in such a computation are from the set $\{\epsilon, i\tau, c\tau\}$. This is due to the fact that the left most process I does not offer any communication to the left, and E does offer any communication to the right. Let ϕ be an automaton over the alphabet $\{\epsilon, i\tau, c\tau\}$. We say that the linear chain $I \odot P^i \odot E$ satisfies the property specified by ϕ if $\mathcal{C}(I \odot P^i \odot E) \subseteq L(\phi)$.

Let $\mathcal{F}_i$ be be the linear chain $I \odot P^{c+i} \odot E$. Now, consider the family of linear chains $\mathcal{F}_i$ for $i = 0, 1....$ For an automaton $\mathcal{A}$ over the alphabet Δ', let $\mathcal{A}[a/right.a]$ denote the automaton obtained by renaming all actions of the form *right.a* to a in $\mathcal{A}$. Similarly, $\mathcal{A}[a/left.a]$ and $\mathcal{A}[a/left.a][a/right.a]$ are defined. In the last case, both type of actions *left.a* and *right.a* are renamed to a. For a process P, we let $P[a/left.a]$, $P[a/right.a]$ and $P[a/left.a][a/right.a]$ denote processes obtained by similar renaming. Let $\Delta'' = \{right.a : a \in \Delta\}$. The following theorem presents a method for verification of the correctness of a family of linear networks.

Theorem 2 (Soundness and Completeness Theorem). *For $i = 0, 1...$, let $\mathcal{F}_i = I \odot P^{c+i} \odot E$ be a family of linear networks and ϕ be an automaton over $\{\epsilon, i\tau, c\tau\}$. Then, $\forall i \geq 0$, $\mathcal{F}_i$ satisfies the specification given by ϕ iff there exists a finite state automaton $\mathcal{A}$ over Δ'' satisfying the following properties.*

1. $\mathcal{C}(I \odot P^c) \subseteq L(\mathcal{A})$;
2. $L(\mathcal{A}[a/right.a] \odot P[a/left.a]) \subseteq L(\mathcal{A})$;
3. $L(\mathcal{A}[a/right.a] \odot E[a/left.a]) \subseteq L(\phi)$;

Soundness part of the above theorem, given by the "if" part, is shown by proving that, $\forall i \geq 0$, $L(\mathcal{A})$ contains all the computations of $I \odot P^{c+i}$; this follows from part 1 of the theorem and the property of the composition operator of an automaton with a process. The completeness, specified by the "only if" part, is shown by obtaining an automaton $\mathcal{A}$ with the property that the set of 2-d strings accepted by $\mathcal{A}$ is exactly the computations of a linear network of the form $I \odot P^{c+i}$; the automaton $\mathcal{A}$ moves on the 2-d string and ensures that the string is according to the transitions of the individual processes.

In the above theorem $\mathcal{A}$ is the automaton that specifies the inductive invariant, step (1) is the basis of the induction, step (2) is the induction step, and step (3) is the step that checks the correctness with respect to ϕ.

3.2 Linear Networks with Fairness

In this subsection, we present an induction based method for verification of correctness under fairness. The fairness that we consider here is the classical weak fairness; roughly speaking, a computation is said to be weakly fair if every process is infinitely often disabled or infinitely often executed.

The set of fair computations of a linear chain $P_0 \odot ... \odot P_{n-1}$, denoted by $\mathcal{FC}(P_0 \odot ... \odot P_{n-1})$, is defined to be the set of computations that satisfy the fairness requirements for all the processes excepting the boundary process P_{n-1} (the boundary process P_{n-1} is open and its fairness will be taken into consideration when we compose the chain with another process on the right side).

In order to specify invariants, we use generalized Buchi automata [GW91] on 2-d strings. A generalized Buchi automata is almost same as a Buchi automata defined earlier excepting that the acceptance condition is a collection of subsets of states; formally, a generalized Buchi automata $\mathcal{A}$ is a 5-tuple $(Q_0, \Sigma, Q, \delta, F)$ where F is a collection $\{F_1, ..., F_k\}$ of subsets of Q; Q_0, Σ, Q, δ are defined as before. An input string in Σ^∞ is accepted by $\mathcal{A}$ if there exists a run of $\mathcal{A}$ on the input such that for each $i = 1, ..., k$ some state in F_i appears infinitely often in the run. The languages $\mathcal{L}(A)$ and $L(A)$ are defined as before.

Now we define the fair right composition of a generalized Buchi automaton $\mathcal{A}$ and a process P, denoted as $\mathcal{A} \odot_f P$ as follows. The construction of the automaton $\mathcal{A} \odot_f P$ is similar to $\mathcal{A} \odot P$ excepting that it uses an additional flag to capture the fairness condition on the previous right most process; note that after the composition the new right most process will be P. If F and F', respectively, are the acceptance conditions in $\mathcal{A}$ and $\mathcal{A} \odot P$ then the number of subsets in F' is one more than that in F; the additional subset in F' captures the weak fairness requirement on the previous right most process, and the remaining subsets of F' correspond with those of F. Details of the construction of $A \odot_f P$ will be given in the full paper.

Let I, P and E be processes as defined in the previous subsection and ϕ be a generalized Buchi automaton over the alphabet $\{\epsilon, i\tau, c\tau\}$. As usual we say that the the linear chain $I \odot P^i \odot E$ satifies ϕ under fairness if all fair computations of the chain are accepted by ϕ. Now, we have the following soundness theorem for verifying correctness under fairness.

Theorem 3 (Soundness Theorem). *Let $\mathcal{F}_i = I \odot P^{c+i} \odot E$, for $i = 0, 1...$, be a family of linear networks and ϕ be an automaton over $\{\epsilon, i\tau, c\tau\}$. Then, $\forall i \geq 0$, $\mathcal{F}_i$ satisfies the specification given by ϕ under fairness if there exists a finite state generalized Buchi automaton $\mathcal{A}$ over Δ'' satisfying the following properties.*

1. $\mathcal{FC}(I \odot P^c) \subseteq L(\mathcal{A})$;

2. $L(\mathcal{A}[a/right.a] \odot_f P[a/left.a]) \subseteq L(\mathcal{A})$;
3. $L(\mathcal{A}[a/right.a] \odot_f E[a/left.a]) \subseteq L(\phi)$;

3.3 Circular Networks

Now, we discuss how to verify families of circular networks of processes. Let Δ and Δ' be as defined above. Let $P_0, ..., P_{n-1}$ be processes over the alphabet Δ'. For each i, $0 \leq i < n$, let P_i'' denote the process where actions of the form $left.a$, $right.a$ are renamed to $a_{(i-1) \bmod n}$ and a_i respectively. Note that, we are also renaming $left.a$ in P_0 and $right.a$ in P_{n-1}. We define a circular network, denoted $P_0 \oplus P_1 \oplus ... P_{n-1}$, to be the composition of the CCS processes $P_0'', ..., P_{n-1}''$, i.e., $P_0'' \circ P_1'' \circ ... \circ P_{n-1}''$. The set of computations of such a circular network, denoted $\mathcal{C}(P_0 \oplus ... \oplus P_{n-1})$, is defined as a set of 2-d strings over the alphabet $\{\epsilon, i\tau, c\tau\}$ as before.

Let I,E and P be processes over Δ'. We define $\mathcal{G}_i$ to be the circular network $I \oplus P^{c+i} \oplus E$ where P^i denotes the expressions $P \oplus ... \oplus P$ containing $c+i$ number of Ps. The following theorem tells us how to use induction for verification of families of circular networks. As before we define what it means for a circular network to satisfy a specification given by an automaton.

Theorem 4 (Soundness and Completeness). *Let $\mathcal{G}_i = I \oplus P^{c+i} \oplus E$, for $i = 0, 1...$, denote a family of circular networks. Let ϕ be an automaton over $\{\epsilon, i\tau, c\tau\}$. Then, $\forall i \geq 0$ $\mathcal{G}_i$ satisfies the specification ϕ iff there exists a finite state automaton $\mathcal{A}$ over Δ' satisfying the following properties.*

1. $\mathcal{C}(I \odot P^c) \subseteq L(\mathcal{A})$;
2. $L((\mathcal{A}[a/right.a]) \odot P[a/left.a]) \subseteq L(\mathcal{A})$;
3. $L(\mathcal{A}[a/right.a][a'/left.a] \otimes E[a/left.a,][a'/right.a]) \subseteq L(\phi)$;

It is to be noted that, unlike in Theorem 2, the automaton $\mathcal{A}$ in the above theorem is over Δ'. Also, we use the $\otimes$ operator in step 3. We can prove a soundness theorem, similar to Theorem 3, for correctness under fairness for circular networks as well.

Now we discuss how to automate the different steps in the above two theorems. All the three steps require checking language containment of automata on 2-d strings. Step 1 can be done by obtaining the automaton corresponding to $I \odot P^c$ and then checking language containment. In steps 2 and 3, we need to compute the composition of an automata with a process, and this can be done using the method given in the previous section. The complexity of this algorithm is proportional to the product of the sizes of the automaton and the process.

3.4 Language Containment Problem

The language containment problem for generalized 2-d Buchi automata (and even for 2-d Buchi automata) is undecidable. However, the following lemma gives us a sufficient condition for language containment that is decidable. It states that, for automata A_1 and A_2, if the set of one-dimensional strings accepted by A_1 is contained in the set accepted by A_2 then $L(A_1) \subseteq L(A_2)$.

Lemma 5. *Let A_1 and A_2 be generalized Buchi automata on Δ. If $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ then $L(A_1) \subseteq L(A_2)$.*

The condition $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ can be checked by complementing A_1 using the methods of [SVW85]. However this will have exponential complexity. Now, we give another condition that is easily checkable. This method uses simulation relations.

Let $A_1 = (Q_0, \Sigma, Q, \delta, F)$ and $A_2 = (Q_0', \Sigma, Q', \delta', F')$ be generalized 2-d Buchi automata. (Note that F and F' are collections of subsets of states). A simulation relation $R \subseteq Q_1 \times Q_2$ is a binary relation satisfying the following properties.

- For every $q \in Q_0$ there exists a $q' \in Q'$ such that $(q, q') \in R$.
- If $(q, x, r) \in \delta$ then for every q' such that $(q, q') \in R$ there exists an $r' \in Q'$ such that $(q', x, r') \in R$.
- There exists a on-to function f from F to F' such that the following property is satisfied:
  For every subset $C \in F$ and for every $q \in C$ and every $q' \in Q'$, if $q \in C$ and $(q, q') \in R$ then $q' \in f(C)$.

The lemma given below follows from known results.

Lemma 6. *If there exists a simulation relation between A_1 and A_2 then $L(A_1) \subseteq L(A_2)$.*

Checking if there exists a simulation relation can be done efficiently for the case when the automata are Bcuhi automata.

Examples. We first consider a simple token passing algorithm for circular networks considered in [CG87]. Here we have circular network of identical processes that communicate by passing a token around in the anti-clockwise direction. The diagram denoting such a process is given in figure 1. A process P is initially in the state N and receives a token from the left by executing the action $left.t?$ and enters the critical region C (here, left and right are defined with respect to a person facing the center of the circle). In state C, it executes an internal transition and stays in the same state, or it executes the action $right.t!$ and goes to state N. Let I be the same process as P excepting that its initial state is C, and E be same as P. Let ϕ be the automaton on 2-d strings, given in figure 2, that states that exactly one process is in state C at any time.

To verify this property for all circular networks of the type $I \oplus P^{i+1} \oplus E$ for all $i \geq 0$, we use the invariant automaton given in figure 3. The transitions of this automaton are oblivious to the last component in each input symbol (i.e., the component denoting the next state of a process); for this reason each input symbol is given as a triple; a value * in an input symbol should be taken as a wild card; a value of l or r or i appearing as the last component indicates the automaton is scanning the left most, right most and internal processes respectively. States a and g denote the occurrance of a sequence of global states in

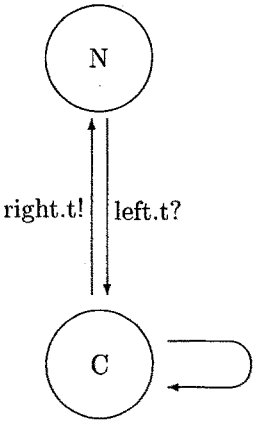

Fig. 1. Process P

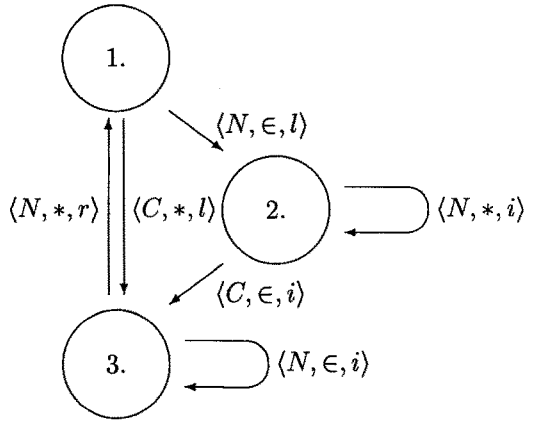

Fig. 2. Automaton Φ

which the left most process has the token and is in the critical section. (State f denotes the situation where a global state has more than one token; this is an error state). States e, c and d denote a sequence of global states in which an inner process has the token and is in critical section. States b and h denote a sequence of global states in which the right most process is in critical section. The transition from h to i occurs when the right most process gives the token to the external world. The transition from i to j denotes the case where the left most process gets the token from the external world.

It can be shown that the above invariant automaton satisfies all the three conditions given in Theorem 4. This can be shown by exhibiting simulation relations.

Now, we show how the same invariant automaton, given in figure 3, can be changed to verify the liveness property that process I enters the critical section infinitely often. The change to the invariant automaton is the addition of the following acceptance condition F which makes it in to a generalized Buchi automaton. F consists of two sets F_1 and F_2. F_1 consists of all states other than a and g, F_2 consists of all states other than c, d, e.

We can also prove the safety and liveness properties of the more complex token ring example considered in [CGJ95]. Details are left out due to lack of space.

4 Conclusions

In this paper we proposed a formalism based on automata on two dimensional strings for specifying inductive invariants for proving correctness of families of linear and circular networks. We proved our inductive approach to be sound and complete (semantical completeness). We have illustrated our approach by simple examples.

We have also given the inductive approach for verification of parametrized systems under fairness. In this case, we use generalized Buchi automata (or

Streett automata) as inductive invariants. For this case, we have proved the soundness theorem.

As part of future work, it will be interesting to automate the different parts of the induction based approach and apply them to real practical examples. It will also be interesting to extend our approach to networks defined by context free grammars [SG89]. Further more, it will also be interesting to investigate logic based approaches for specification of the invariants.

References

[AK86] K. R. Apt and D. Kozen: Limits to Automatic Program Verification. Information Processing Letters, 22.6 (1986), 307-309.

[BCG89] M. Browne, E. M. Clarke and O. Grumberg: Reasoning About Networks with Many Identical Finite State Processes. Inf. and Computation, 81(1):13-31, Apr. 1989.

[CGJ95] E. M. Clarke, O. Grumberg and S. Jha: Verifying Parametrized Networks Using Abstraction and Regular Languages. CONCUR 95.

[CG87] E. M. Clarke, and O. Grumberg: Avoiding the State Explosion Problem in Temporal Logic Modelchecking Problem. In Proceedings of ACM Symposium on Principles of Distributed Computing 1987.

[EN95] E. A. Emerson and K. S. Namjoshi: Reasoning About Rings, Proceedings of 22nd POPL conferece, Jan 1995.

[EN96] E. A. Emerson and K. S. Namjoshi: Automatic Verification of Parametrized Synchronous Systems. Proceeding of the International Conference on Computer Aide Verification 1996.

[GS92] S. M. German and A. P. Sistla: Reasoning About Systems with many Processes. JACM, July 1992, Vol 39, No. 3, pp 675-735.

[GW91] P. Godefroid and P. Wolper: Using Partial Approach to Modelchecking. Proc 6th IEEE Symposium on Logic in Computer Science, pp 406-415, Amsterdam, July 1991.

[KM89] R. P. Kurshan and K. McMillan: A Structural Induction Theorem for Processes. ACM Sym. on Principles of Distributed Computing, Aug. 1989.

[SG89] Z. Shtadler and O. Grumberg: Network Grammars, Communication Behaviors and Automatic Verififcation. Proc. of International Workshop on Automatic Verification Methods for Finite State Systems, June 1989.

[Si97] A. P. Sistla: Parametrized Verification of Linear Networks Using Automata as Invariants. Technical report, University of Illinois at Chicago 1997.

[SVW85] A. P. Sistla, M. Vardi and P. Wolper: The Complementation Problem for Buchi Automata and Applications to Temporal Logics. Proceedings Of The 12th International Colloquium On Automata, Languages And Programming, Greece, August 1985; The journal version of the paper appeared in Theoretical Computer Science, 49, No 2,3 1987, pp 217-237.

[SG87] A. P. Sistla and S. M. German: Reasoning About many Processes. LICS 87.

[WL89] P. Wolper and V. Lovinfosse: Verifying Properties of Large Sets of Processes with Network Invariants. Proc. 1989 Intl Wokshop on Automatic Verification Methods for Finite State Systems 1989.

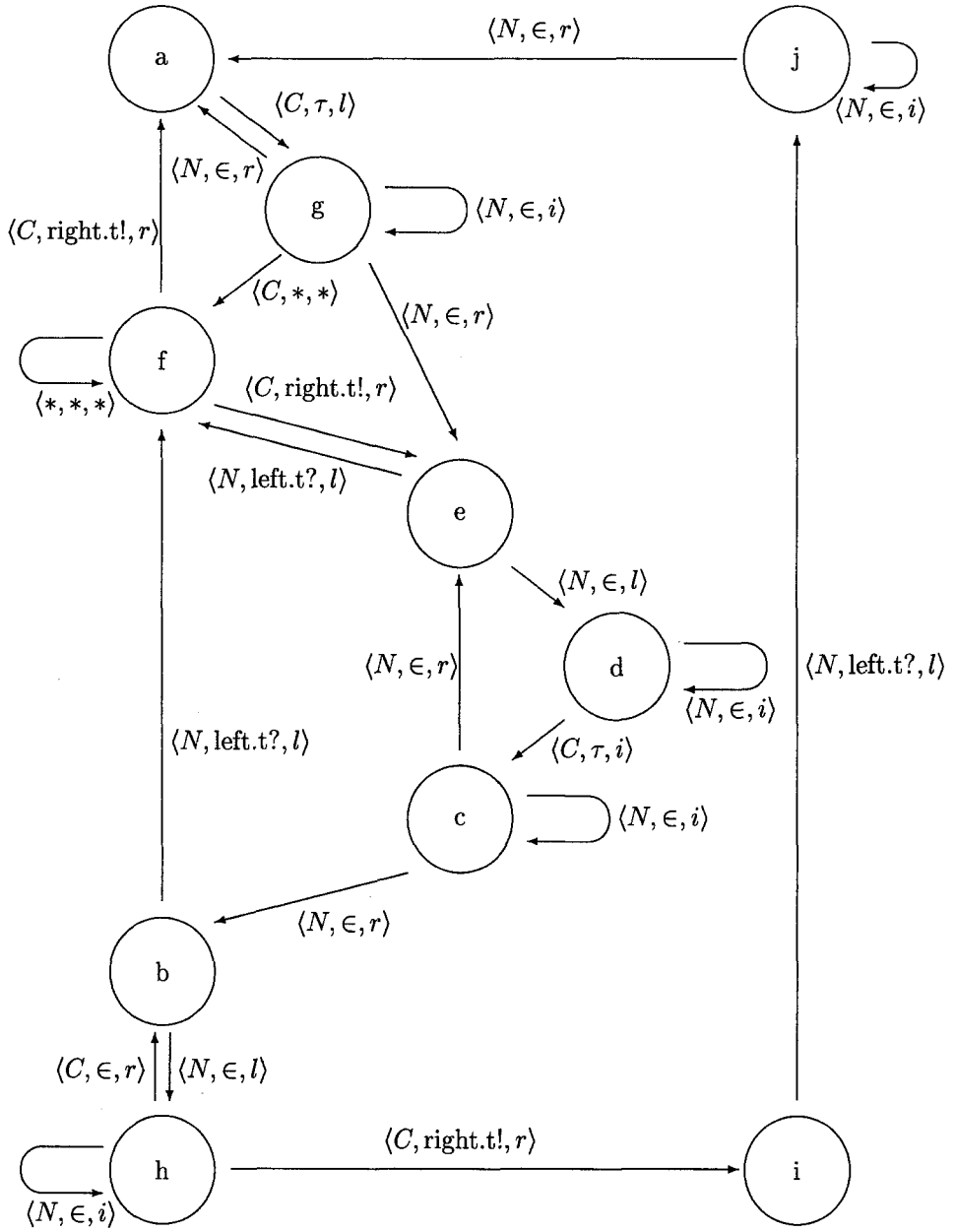

Fig. 3. The invariant automaton A