Parameterized verification of Linear networks using Automata as Invariants*

A. Prasad Sistla
V.Gyuris
Electrical Engineering and Computer Science Department,
The University of Illinois at Chicago, USA

November 18, 1998

1 Introduction

Recently there has been much interest in parameterized verification, i.e., verification of a family of systems $\{P_i\}_{i=1}^{\infty}$, where P_i is a system consisting of i 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 an 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 of restricted classes of parameterized 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 problems) there exists a k such that correctness of families of networks up to size up to k implies the correctness of networks of all sizes; this has been done in the works of [EN95, GS92].

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

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 $I \circ P \;\; \leq \;\; P$ where $\circ$ 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].

In this paper, we propose a 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. The inductive step is proved by showing that $L(A) \circ P$ is contained in $L(A)$ where $L(A)$ is the set of two dimensional strings accepted A and $\circ$ is a composition operator. Our method is sound and complete for verifying correctness of families of linear networks (as well as circular networks) of the form (I, P^i, E) (for all $i \geq 0$) where I, E and P are processes. The completeness of our method, which is semantic completeness, follows easily due to the expressive power of the two dimensional automata.

The above approach is extended for verifying correctness under fairness as well. For this we use Streett automata as invariants and the inductive step is specified using a fair composition operator. We show that this induction based approach for verifying correctness under fairness is sound. However, we do not have a completeness result for this case.

In general checking if $L(A_1) \subseteq L(A_2)$ is an undecidable for both Buchi and Streett automata. 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 propose two automated methods for checking correctness of families of linear chains of processes. The first method does not consider fairness, while the second method does consider fairness. Both these methods are based on generating invariants using a fix point computation. (For these methods we use universal automata; such automata are said to accept a string if all their runs on a string are accepting runs). The fix-point computations employ a function that computes a automaton $f(V, A)$ satisfying the following property: A 2-d string σ is accepted by $f(V, A)$ iff all strings obtained by composing σ with the process V are accepted by the automaton A. For checking correctness of a family of linear chains of the form (I, P^i, E), with respect to the correctness specification ϕ, a sequence of automata $A_0, A_1, ...$ are constructed such that $A_0 = f(E, \phi)$ and $L(A_i) = L(A_{i-1}) \cap L(f(P, A_{i-1}))$ until a fix-point is reached. In general, the fix-point may not terminate. The second method also uses a similar method for checking correctness under fairness; it uses Streett automata.

Both of the above methods have been implemented. On a given input, the system, when it terminates, gives an "yes" or "no" answer indicating if the parameterized system satisfies the correctness specification or not. We have successfully tested the system on some examples. It terminated in all these cases and gave correct answers.

Our paper is organized as follows. Section 2 contains comparison to related work. Section 3 gives the notation and definitions. Section 4 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 5 presents the automated approach for verification. Section 6 contains concluding remarks.

2 Related Work

As has been indicated earlier there has been much work done in parameterized verification. Fully automated methods have been proposed in [SG87, GS92, EN96] for client-server type architectures. These methods do not apply to linear networks. Many induction based methods were also proposed in earlier works [WL89, KM89, BCG89, SG89, MG91, CGJ95, K97, LHR96]. Compared to all these works, our method is simple and powerful enough so that the completeness of the method, for verification of safety properties, follows very easily. None of the earlier works are known to be complete even for safety properties. Below we compare our works to each of the above mentioned works individually.

In the approaches of [WL89, KM89, BCG89], the correctness property as well as the inductive invariant are specified as separate processes. In these methods the invariant is specified by the user where as in our system it is generated automatically. Furthermore, in their methods it is harder to specify correctness properties involving global states and communication patterns.

In [CGJ95], the authors employ Abstract Transition Systems (ATS) 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. In their approach, the correctness property is specified in the branching time temporal logic $\forall CTL$. Compared to their method, our approach has the following advantage. Firstly, our method is simpler as we use a single formalism for specifying properties of global states as well as properties over time. On the contrary they employ two different formalisms— the regular expressions to specify properties of global states and the transitions in the ATS to specify computation steps. Secondly, we handle fairness in our systems and their methods do not handle fairness. This allows us to handle both safety and liveness properties. It is to be noted that the approach of [CGJ95] apply to a larger class of networks, i.e. those generated by context free grammars.

In [K97], the authors consider parameterized verification of linear and tree networks of processes. They employ regular and tree regular sets for specifying invariance properties of global states and provide an invariant based method for verification; the invariants are specified symbolically using BDDs and are generated automatically. Their method pertains to verification of invariance properties only. On the contrary we can verify both safety and liveness properties.

In [LHR96], a method for verification of families of linear networks is presented. This method uses observer processes as network invariants which are also generated automatically; this method can only be used for verification of safety properties.

In [SG89, MG91] the authors consider families of networks generated by context free grammars. In their approach, the invariant is defined by requiring it to be equivalent to every system of processes generated by the grammar. This stronger requirement on the invariant makes it difficult to get invariants in many cases.

3 Definitions

Processes

We consider a computation model where processes communicate by synchronizing on actions. Our communication model is slightly different from the standard CCS/CSP model, but is equally powerful for defining linear networks. Our model allows us to define linear networks of identical processes in a much simpler way. It makes our notation and presentation much simpler. We assume that we have a finite set Δ of actions and a binary relation MP over Δ satisfying the following property: for each action $a \in \Delta$, there exists exactly one pair belonging to MP in which a appears. We say that (a, b) is a matching pair if $(a, b) \in MP$. We say that an action a is a *left action* (respectively, *right action*) if a appears as the first element (respectively, second element) in the unique matching pair in which a appears. Thus each action is either a left or a right action, and not both. We assume that Δ does not contain any of the symbols from the set $\{\epsilon, i\tau, c\tau\}$.

We assume that the states of all processes are drawn from a finite set S of states. The sets S, Δ and MP are fixed throughout the remainder of the paper.

We consider linear chains of processes where the processes communicate by synchronizing on matching pairs of actions. Formally, a process is a triple (S', R, S_0) where S' is a subset of states (i.e. $S' \subseteq S$), $R \subseteq S' \times (\Delta' \cup \{i\tau\}) \times S'$ is a set of transitions and $S_0 \subseteq S'$ is the set of initial states.

Figure 1 gives a process P which 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. In state C, it executes an internal transition (labeled by $i\tau$) and stays in the same state, or it executes the action $right.t!$ and goes to state N. For this example, $(right.t!, left.t?)$ is the single matching pair. In a linear chain, in which each process is given by P, communication occurs when one process executes the transition labeled by $right.t!$ and its right neighbor executes the transition labeled by $left.t?$ simultaneously. A formal

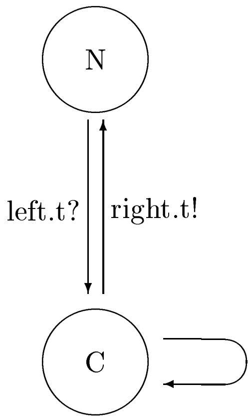

Figure 1: Process P

definition of computations of such linear chains is given in section 4.1. It is to be noted that to define the same linear chain in CCS we need to rename the actions in each process; this renaming is avoided in our formalism. It can easily be shown that our formalism as powerful as CCS for defining linear networks, i.e. for any linear network defined in CCS can also be defined in our formalism possibly using more actions.

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.

Formally, an automaton A is a 5-tuple $(Q_0, \Sigma, Q, \delta, F)$ where $\Sigma = S \times (\Delta \cup \{i\tau, c\tau, \epsilon\}) \times \{left, right, internal\} \times S$ 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. The automaton A is deterministic if for every $q \in Q$ and $x \in \Sigma$ there is exactly one $q' \in Q$ such that $(q, x, q') \in \delta$. A one dimensional string η over Σ is an infinite sequence $\eta_0, \eta_1, ..., \eta_i, ...$ A run r of the automaton A is an infinite sequence of automaton states $q_0, q_1, ...$ such that $q_0 \in Q_0$ and for each $i \geq 0$, $(q_{i+1}, \eta_i, q_i) \in \delta$. We say that the run r is accepting if some state in F appears infinitely often in r. We define two different languages $\mathcal{L}(A)$ and $\mathcal{L}_\forall(A)$ with respect to the automaton A. $\mathcal{L}(A)$ is the set of all η such that there exits an accepting run of A on η. $\mathcal{L}_\forall(A)$ is the set of all η such that every run of A on η is an accepting run. Intuitively, $\mathcal{L}(A)$ and $\mathcal{L}_\forall(A)$ are the one dimensional languages accepted by A when we consider it as an existential and as an universal automaton respectively; when A is deterministic then both these languages are identical.

Now we associate two dimensional languages with an automaton as follows. Let n be a positive integer. A two-dimensional string σ of width n 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 $a_{i,0}, a_{i,n-1} \in \Delta \cup \{\epsilon, i\tau, c\tau\}$ and for all j such that $0 < j < n-1$, $a_{i,j} \in \{\epsilon, i\tau, c\tau\}$ (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. Also, note that in a 2-d string only the boundary processes, i.e. left end and right end columns, can have open communications). 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 $0 \leq i < \infty$, $d_{i,0} = left$, $d_{i,n-1} = right$, and for $0 < j < n-1$, $d_{i,j} = internal$. Here $d_{i,j}$ denotes whether j is the left boundary process , right boundary process or an internal process.

For an automaton A, we let $L(A)$ (respectively $L_\forall(A)$) denote the set of 2-dimensional strings such that their linearizations are in $\mathcal{L}(A)$ (respectively in $\mathcal{L}_\forall(A)$). We say that a 2-d string σ is accepted by A, when considered as an existential (resp., universal) automaton, iff $\sigma \in L(A)$ (resp., $\sigma \in L_\forall(A)$).

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.

Let σ be a 2-d string of width n and P be a process. We define the right composition of σ and P, denoted as $\sigma \circ P$, to be a set of 2-d strings of width $n+1$ as follows. Intuitively, a 2-d string δ is in $\sigma \circ P$ if it is obtained by fusing the 2-d string σ with a computation of P where the fusion is carried out by synchronizing on matching pairs of actions. Let $\sigma = (u_0, a_0, ..., u_i, a_i, ...)$ and $P = (S', R, S_0)$. A 2-d string $\delta = (v_0, b_0, ..., v_i, b_i, ...)$ of width $(n+1)$ is in $\sigma \circ P$ iff $v_0 = (u_{0,0}, u_{0,1}, ..., u_{0,n-1}, s_0)$ for some $s_0 \in S_0$ and the following conditions are satisfied, for all $i \geq 0$:

1. For all j such that $0 \leq j < n-1$, $v_{i,j} = u_{i,j}$, $b_{i,j} = a_{i,j}$, and $v_{i,n-1} = u_{i,n-1}$. This states that all the first n states in the row v_i are same as the corresponding ones in u_i, and all the first $n-1$ actions in b_i are same as those in a_i.

2. If $a_{i,j} = \epsilon$ for all j such that $0 \leq j < n$, then $b_{i,n-1} = \epsilon$ and one of the following conditions holds:

   (a) $b_{i,n} = \epsilon$ and $v_{i+1,n} = v_{i,n}$. This denotes a stuttering step in which no process executes a transition.

   (b) $b_{i,n}$ is a left action or $b_{i,n} = i\tau$ and $(v_{i,n}, b_{i,n}, v_{i+1,n}) \in R$. This denotes the case where process P makes an internal move or offers an external communication (such a communication has to be a left action).

3. If $a_{i,n-1} \in \Delta$ then $a_{i,n-1}$ is a left action and the following conditions hold. $b_{i,n-1} = b_{i,n} = c\tau$ and there is an action $b \in \Delta$ such that $(a_{i,n-1}, b)$ is a matching pair and $(v_{i,n}, b, v_{i+1,n}) \in R$. This condition ensures that if the last action in a_i is a left action then it synchronizes with an appropriate right action of P.

4. If $a_{i,n-1} \notin \Delta$ and for some j such that $0 \leq j < n$, $a_{i,j} \neq \epsilon$ then the following holds. $b_{i,n-1} = a_{i,n-1}, b_{i,n} = \epsilon$ and $v_{i,n} = v_{i+1,n}$. This

condition ensures that process P does not execute any transition if some other process made a transition in the current row.

From the above definition, it is not difficult to see that the following properties hold for any $i \geq 0$. If $a_{i,n-1}$ is a right action, or $a_{i,n-1}$ is a left action and there is no synchronizing transition of P from state $v_{i,n}$ then $\sigma \circ P$ is an empty set (this is because a right action can not synchronize with a process on the right side). Also, if $b_{i,n} \in \Delta$ then $b_{i,n}$ is a right action.

We extend the compositional operator $\circ$ to a set C of 2-dimensional strings as follows— $C \circ P = \bigcup_{\sigma \in C}(\sigma \circ P)$ where $\bigcup_{\sigma \in C}$ denotes the union over all σ in C. For a set C of 2-d strings and processes $P_0, ..., P_{n-1}$, we let $C \circ P_0 \circ P_1 ... \circ P_{n-1}$ denote the set of 2-d strings obtained by application of $\circ$ from left to right, i.e. the set $((...((C \circ P_0) \circ P_1)...) \circ P_{n-1})$.

4 Verification of Linear Networks Using Induction

We consider linear networks of processes in which adjacent processes communicate by synchronizing on matching pairs of actions. In this section, we show how we can verify the correctness of such linear networks and circular networks using automata as invariants.

4.1 Linear Networks

Let $(P_0, P_1, ..., P_{n-1})$ be a linear sequence of processes. We can consider such a sequence as a linear chain of processes where processes communicate by synchronizing on a matching pair of actions. For each i such that $i \leq 0 < n-1$, P_i and P_{i+1} communicate by simultaneously executing transitions corresponding to a left action and a right action, respectively, of a matching pair. Further more, P_0 (or P_{n-1}) can offer an open communication to the outside world by executing a transition corresponding to a right action (respectively, left action).

For each j, $0 \leq j < n$, let $P_j = (S_j, R_j, S_{j,0})$. Formally, the set of infinite computations of the chain of processes, denoted as $\mathcal{COMP}(P_0, ..., P_{n-1})$ is the set of 2-d strings $\sigma = u_0, a_0, u_1, a_1, ...u_i, a_i, ...$ of width n satisfying the following properties: for each j, $u_{0,j}$ is an initial state of process P_j (i.e. $u_{0,j} \in S_{j,0}$) and for each $i \geq 0$, one of the following conditions is satisfied.

- For some j, such that $0 \leq j < n$, $a_{i,j} = i\tau$, $(u_{i,j}, i\tau, u_{i+1,j}) \in R_j$, and for all $k \neq j$, $u_{i+1,k} = u_{i,k}$ and $a_{i,k} = \epsilon$. This corresponds to the case where process j makes an internal transition.

- For some j, such that $0 \leq j < n-1$, there exists a matching pair of actions (d, e) such that $(u_{i,j}, d, u_{i+1,j}) \in R_j$ and $(u_{i,j+1}, e, u_{i+1,j+1}) \in R_{j+1}$ and $a_{i,j} = a_{i,j+1} = c\tau$ and the following is satisfied: for each k such that $k \neq j$ and $k \neq j+1$, $a_{i,k} = \epsilon$ and $u_{i,k} = u_{i+1,k}$. This denotes a communication step between two adjacent processes.

- For $j = 0$ or $j = n-1$ the following holds: $a_{i,j} \in \Delta$, i.e. it is a communication action, and $(u_{i,j}, a_{i,j}, u_{i+1,j}) \in R_j$; if $j = 0$ then $a_{i,j}$ is a right action, otherwise $a_{i,j}$ is left action; for all $k \neq j$, $a_{i,k} = \epsilon$ and $u_{i+1,k} = u_{i,k}$. This denotes a computational step where the boundary processes offer an external communication.

- For all j such that $0 \leq j < n$, $u_{i+1} = u_i$, and $a_{i,j} = \epsilon$. This denotes a stuttering step.

Note that 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 $u_{i+1,j} = u_{i,j}$. If $a_{i,j} \neq \epsilon$ then we say that process j is executed in this computational step.

The following lemma is immediate from the definition of the $\circ$, and it relates the computations of linear chains with the $\circ$ operator.

Lemma 4.1 *Let $P_0, P_1, ..., P_n$ be a sequence of processes where $n \geq 1$, and $C = \mathcal{COMP}(P_0, ...P_{n-1})$. Then, $C \circ P_n = \mathcal{COMP}(P_0, ..., P_n)$.*

Let I, P and E be a processes. For any $i \geq 0$, we let (I, P^i, E) and (I, P^i), respectively, denote the finite sequences $(I, P, ...P, E)$, $(I, P...P)$ in which P appears exactly i times. Let ϕ be an automaton. We say that the linear chain (I, P^i, E) satisfies the property specified by ϕ if $\mathcal{COMP}(I, P^i, E) \subseteq L(\phi)$.

Theorem 4.1 (Soundness and Completeness Theorem) *Let I, P, E be processes, ϕ be an automaton and $c \geq 0$ be an integer. Then, $\forall i \geq 0$, (I, P^{c+i}, E) satisfies the specification given by ϕ iff there exists a finite state automaton $\mathcal{A}$ satisfying the following properties.*

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

2. $L(\mathcal{A}) \circ P \subseteq L(\mathcal{A})$;

3. $L(\mathcal{A}) \circ E \subseteq L(\phi)$;

Proof: 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 in $\mathcal{COMP}(I, P^{c+i})$; this follows from part 1 of the theorem and the property of the composition operator given by lemma 4.1. 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 $\mathcal{COMP}(I, P^{c+i})$; the automaton $\mathcal{A}$ moves on the 2-d string and ensures that the string is according to the transitions of the individual processes. As it moves on each row, the automaton has to ensure that only one legal computational step (which is either an internal or a communication transition) takes place in each row. The construction of the automaton is fairly straightforward. In fact, we can actually construct such an automaton which is deterministic. $\square$

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. Each automaton A', mentioned in the above theorem, is considered as an existential automaton. The theorem continues to hold even when we consider universal automata, i.e. when we replace the language $L(A')$ by the language $L_\forall(A')$ for each of the automaton A' mentioned in the statement of the theorem. The proof of the soundness part of the theorem, for universal automata, is trivial as before. The completeness proof continues to hold for universal automata as well, since the automaton $\mathcal{A}$ constructed in the proof is deterministic.

4.2 Linear Networks with Fairness

In this subsection, we present an induction based method for verification of correctness under fairness.

Streett Automata

In order to specify invariants, we use Streett automata on 2-d strings. A Streett automaton is defined similar to the Buchi automaton excepting that the accepting condition F is a collection of pairs of subsets of automaton states. Formally, a Streett automaton $\mathcal{A}$ is a 5-tuple $(Q_0, \Sigma, Q, \delta, F)$ where F is a collection $\{(F_1, G_1), ..., (F_k, G_k)\}$ of pairs of subsets of Q; Q_0, Σ, Q, δ are defined as before. A run r of A on an input string σ is accepting if for each i, $1 \leq i \leq k$, the following condition holds: if some state in F_i appears infinitely often then some state in G_i also appears infinitely often. The languages $\mathcal{L}(A)$, $L(A)$, $\mathcal{L}_\forall(A)$ and $L_\forall(A)$ are defined as before.

Fair Composition

Let $P_0, ..., P_{n-1}$ be a sequence of processes where $P_j = (S_j, R_j, S_{j,0})$. Let $v = (v_0, ..., v_{n-1})$ be an n-tuple. For any j such that $0 < j < n-1$, we say that process j is *enabled* in v if any one of the following conditions is satisfied.

1. There is an internal transition of process P_j from v_j, i.e. $(v_j, i\tau, s) \in R_j$ for some state s.

2. Processes $j-1$ and j can communicate in the global state represented by v. That is, there exists a matching pair (d, e) such that $(v_{j-1}, d, s) \in R_{j-1}$ and $(v_j, e, s') \in R_j$ for some s, s'.

3. Processes j and $j+1$ can communicate in the global state represented by v. That is, there exists a matching pair (d, e) such that $(v_j, d, s) \in R_j$ and $(v_{j+1}, e, s') \in R_{j+1}$ for some s, s'.

We say that process $n-1$ is enabled in v if either of the conditions 1,2 is satisfied for $j = n-1$, or there exists a transition $(v_{n-1}, d, s) \in R_{n-1}$, for some s, d such that d is a left action. Similarly, we say that process 0 is enabled in v if either of the conditions 1,3 is satisfied for $j = 0$, or there exists a transition $(v_0, e, s) \in R_n$, for some s, e such that e is a right action.

Now we define the fair right composition of a 2-d string σ of width n and process Q, denoted $\sigma \circ_f Q$, as follows. Intuitively, $\sigma' \in \sigma \circ_f Q$ iff $\sigma' \in \sigma \circ Q$ and it satisfies the fairness requirement with respect to the last two processes, i.e. with respect to process $(n-1)$ and process n. Process n is Q. To know whether the fairness requirement of process $n-1$ is satisfied, we need to know this process and its neighbors (its left neighbor is process $n-2$ and right neighbor is Q). Let P_{n-2} and P_{n-1} denote processes $n-2$ and $n-1$ respectively (if $n = 1$ then P_{n-2} is undefined and is not needed). Our composition operator is defined with respect to these processes. (When we consider linear chains of the form (I, P^i, E), P_{n-1} is P for $n \geq 2$ and is I for $n = 1$; P_{n-2} is defined similarly).

Consider a 2-d string $\sigma' = u_0, a_0, u_1, a_1, ..., u_i, a_i, ...$ of width $n+1$ in $\sigma \circ Q$. Let $a_i = (u_{i,0}, u_{i,1}, ..., u_{i,n})$. We say that σ' is fair with respect to the last two processes if the following condition is satisfied for $k = n-1$ and $k = n$ and for all $i \geq 0$: if process k is enabled in u_i then there exists a $j \geq i$ such that either process k is not enabled in u_j, or $a_{j,k} \neq \epsilon$ (i.e. process k is executed in the j^{th} step).

Now, we define $\sigma \circ_f Q$ to be the set of all $\sigma' \in \sigma \circ Q$ such that σ' is fair with respect to the last two processes. For any set C of 2-d strings, we define $C \circ_f Q$ to be the set $\bigcup_{\sigma \in C}(\sigma \circ_f Q)$.

Inductive Proofs for Correctness Under Fairness

Let $(P_0, ..., P_{n_1})$ be a sequence of processes and $u = u_0, a_0, ...u_i, a_i, ...$ be a computation in $\mathcal{COMP}(P_0, ..., P_{n-1})$. We say that u is fair if the following condition is satisfied for all j such that $0 \leq j < n$ and for all $i \geq 0$: if process j is enabled in u_i then there exists some $k \geq i$ such that process is not enabled in u_k or $a_{k,j} \neq \epsilon$. Let $\mathcal{FCOMP}(P_0, ..., P_{n-1})$ denote the set of fair computations of the chain $(P_0, ..., P_{n_1})$. We have the following lemma relating fair computations and the operator $\circ_f$.

Lemma 4.2 *Let $P_0, ..., P_n$ be a set of processes and let C denote the set $\mathcal{FCOMP}(P_0, P_1, ...P_{n-1})$. Then $C \circ_f P_n = \mathcal{FCOMP}(P_0, P_1, ...P_n)$.*

Let ϕ be a Street automaton. Let I, P, E be processes. As usual we say that the the linear chain (I, P^i, E) satisfies ϕ under fairness if all fair computations of the chain are accepted by ϕ, i.e. $\mathcal{COMP}(I, P^i, E) \subseteq L(\phi)$. Now, we have the following soundness theorem for verifying correctness under fairness.

Theorem 4.2 (Soundness Theorem) *Let I, P, E be processes, ϕ be a Streett automaton and $c \geq 0$ be an integer. Then, $\forall i \geq 0$, (I, P^i, E) satisfies the specification given by ϕ under fairness if there exists a finite state Streett automaton $\mathcal{A}$ satisfying the following properties.*

1. $\mathcal{FCOMP}(I, P^c) \subseteq L(\mathcal{A})$;
2. $L(\mathcal{A}) \circ_f P \subseteq L(\mathcal{A})$;
3. $L(\mathcal{A}) \circ_f E \subseteq L(\phi)$;

Proof: Assume properties (1), (2) and (3) as stated in the lemma hold. By a simple induction on i, and by using lemma 4.2, it can easily be shown that $\mathcal{FCOMP}(I, P^{c+i}) \subseteq L(A)$, and $\mathcal{FCOMP}(I, P^{c+i}, E) \subseteq L(\mathcal{A}) \circ_f E$. Now, the theorem follows from the above observation and property (3). $\square$

The above theorem holds even for the case when all the automata are universal automata, i.e. for each automaton A' considered in the theorem we use $L_\forall(A')$ instead of $L(A')$.

4.3 Circular Networks

Now, we discuss how to verify families of circular networks of processes. Let $(P_0, ..., P_{n-1})$ be a sequence of processes. We can consider this sequence as a circular network. The set of computations of such a network, denoted as $\mathcal{CCOMP}(P_0, ..., P_{n-1})$, is defined to be the set of all infinite sequences $u_0, a_0, ..., u_i, a_i, ...$ such that, for each $i \geq 0$, the computational step u_i, a_i, u_{i+1} involves an internal transition of some process j, or communication between two neighboring processes j and $(j+1)$ mod n synchronizing on a matching pair; for each $j \geq 0$, $a_{i,j} \in \{\epsilon, i\tau, c\tau\}$. It is to be noted that process $n-1$ can synchronize with process 0 on a matching pair (d, e) in which the processes $n-1, 0$ execute transitions corresponding to the actions d, e respectively. It is straightforward to give a formal definition of $\mathcal{CCOMP}(P_0, ..., P_{n-1})$ and is left to the reader.

We define a new composition operator $\bullet$ between a 2-d string σ and a process Q as follows. Essentially, $\sigma \bullet Q$ gives a set of 2-d strings obtained by synchronizing open communications in the right most as well as left most columns (i.e. right most and left most processes) of σ with the process Q (note that in the definition of the composition operator $\circ$ we only synchronize open communications in the right most column). The definition of $\bullet$ is fairly straightforward and is left to the reader. For a set C of 2-d strings, we define $C \bullet Q$ to be the set which is the union, taken over all $\sigma \in C$, of $\sigma \bullet Q$.

We say that a circular network $(P_0, ..., P_{n-1})$ satisfies a specification given by an automaton ϕ, if $\mathcal{CCOMP}(P_0, ..., P_{n-1}) \subseteq L(\phi)$. 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 4.3 (Soundness and Completeness) *Let I, P, E denote processes, ϕ be an automaton and $c \geq 0$ be an integer. Then, $\forall i \geq 0$, the circular network (I, P^{c+i}, E) satisfies the specification ϕ iff there exists a finite state automaton $\mathcal{A}$ satisfying the following properties.*

1. $\mathcal{COMP}(I, P^c) \subseteq L(\mathcal{A})$;
2. $L(\mathcal{A} \circ P) \subseteq L(\mathcal{A})$;
3. $L(\mathcal{A}) \bullet E \subseteq L(\phi)$;

It is to be noted that we use the $\bullet$ operator in step 3. We can prove a soundness theorem, similar to Theorem 3, for correctness under fairness for circular networks as well. The above theorem holds even when we consider universal automata.

Now we discuss how to automate the different steps in the theorems 4.1,4.2 and 4.3. All the three steps require checking language containment of automata on 2-d strings. Step 1 can be done by obtaining the automaton that accepts exactly the elements in $\mathcal{COMP}(I, P^c)$ and then checking language containment. In steps 2 and 3, we need to compute the composition of the language of an automata with a process. We can actually construct another automaton $\mathcal{B}$ that accepts this language (see [Si97]) and check steps 2 and 3 by using language containment.

4.4 Language Containment Problem

The language containment problem, i.e. the problem of deciding if $L(A_1) \subseteq L(A_2)$ for two given automata A_1, A_2, for 2-d Street automata (and even for 2-d Buchi automata) is undecidable. However, 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, i.e. $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$, then $L(A_1) \subseteq L(A_2)$. The condition $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ can be checked by converting them into Buchi automata and using the complementation construction given in [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 2-d Streett automata. (Note that F and F' are collections of pairs of subsets of states). We use the following notation. Let $R \subseteq Q \times Q'$ be a binary relation. For any $C \subseteq Q$, we let $R(C)$ denote the set $\{q' : \text{for some } q \in C \ (q, q') \in R\}$, i.e. $R(C)$ denotes the states in Q' that are related to some state in C. For any $D \subseteq Q'$, we let $R^{-1}(D)$ denote the set $\{q : \text{for some } q' \in D \ (q, q') \in R\}$.

A binary relation $R \subseteq Q \times Q'$ is called a *simulation relation* from A_1 to A_2 if it satisfies the following properties.

1. For every $q \in Q_0$ there exists a $q' \in Q_0'$ such that $(q, q') \in R$.

2. If $(q, x, r) \in \delta$ then for every q' such that $(q, q') \in R$ there exists an $r' \in Q'$ such that $(q', x, r') \in R$.

Let R be a simulation relation from A_1 to A_2. Let $q = q_0, q_1, ..., q_i, ...$ be a run of A_1 on some input, and let $r = r_0, r_1, .., r_i, ...$ be a run of A_2 on the same input. Then, we say that r corresponds to q, if for all $i \geq 0$, $(q_i, r_i) \in R$. The following is a well known property of simulation relations.

Lemma 4.3 *If R is a simulation relation from A_1 to A_2 and q is a run of A_1 on some input σ, then there exists a corresponding run r of A_2 on σ.*

We say that a simulation relation R from A_1 to A_2 is an *existential simulation* relation if there exists an onto function f from F to F' such that the following property is satisfied:
For every pair of subsets $(C_1, C_2) \in F$ the following conditions are satisfied. First, let (D_1, D_2) denote the pair of subsets given by $f(C_1, C_2)$.

- $R(C_2) \subseteq D_2$, that is, all the elements in C_2 are only related only to elements in in D_2.
- $R^{-1}(D_1) \subseteq C_1$, that is, all the elements in D_1 are only related to elements in C_1.

We say that a simulation relation R from A_1 to A_2 is a *universal* simulation relation if there exists a onto function f from F' to F such that the following property is satisfied:
For every pair of subsets $(D_1, D_2) \in F'$ the following conditions are satisfied. First, let (C_1, C_2) denote the pair of subsets given by $f(D_1, D_2)$.

- $R^{-1}(D_2) \subseteq C_2$, that is, all the elements in D_2 are only related to elements in in C_2.
- $R(C_1) \subseteq D_1$, that is, all the elements in C_1 are only related to elements in D_1.

Lemma 4.4 *Let R be a simulation relation and let q, r be runs of A_1, A_2 respectively such that the run r corresponds to the run q. Then the following conditions hold.*

1. *If R is an existential simulation relation and q is an accepting run of A_1 then r is an accepting run of A_2.*
2. *If R is an universal simulation relation and r is an accepting run of A_2 then q is an accepting run of A_1.*

Proof: We will prove (1) by contradiction. Assume that q is an accepting run of A_1 and r is not an accepting run of A_2. This means that there exists a pair $(D_1, D_2) \in F'$ such that some state in D_1 appears infinitely often in r and all the states in D_2 only appear finitely often in r. By the definition of an existential simulation relation, there exists an onto function f from F to

F' such that $f((C_1, C_2)) = (D_1, D_2)$ and $R(C_2) \subseteq D_2$ and $R^{-1}(D_1) \subseteq C_1$. Since the run r corresponds to q, it should now be easy to see that all the states in C_2 only appear finitely often in q, and at least one state in C_1 appears infinitely often in q. This implies that q is not an accepting run, which is a contradiction. The proof of (2) is symmetric.

Lemma 4.5 *The following properties hold.*

1. *If there exists an existential simulation relation R from A_1 to A_2 then $L(A_1) \subseteq L(A_2)$.*
2. *If there exists an universal simulation relation from A_1 to A_2 then $L_{\forall(A_1)} \supseteq L_{\forall}(A_2)$.*

Proof: To prove (1) assume that R is an existential simulation relation. We show that $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$, and the lemma follows from this. Let σ be a string in $\mathcal{L}(A_1)$ and let $q = q_0, q_1, ..., q_i, ...$ be an accepting run of A_1 on σ. Since R is a simulation relation, by lemma 4.3, we see that there exists a run $r = r_0, r_1, ..., r_i, ...$ of A_2 on σ that corresponds to run q of A_1. By lemma 4.4, we see that r is also an accepting run of A_2 and hence $\sigma \in \mathcal{L}(A_2)$.

To prove (2), we show that $\mathcal{L}_{\forall}(A_2) \subseteq \mathcal{L}_{\forall}(A_1)$. Assume that $\sigma \in \mathcal{L}_{\forall}(A_2)$. Clearly all runs of A_2 on σ are accepting. Now, we show that $\sigma \in \mathcal{L}_{\forall}(A_1)$. If A_1 has no runs on σ then, by definition, $\sigma \in \mathcal{L}_{\forall}(A_1)$. Now consider the case where A_1 has at least one run on σ. Let q be any such run. Since R' is a simulation relation, by 4.3, we see that there exists a corresponding run r of A_2 on σ and r is an accepting run. Again, by lemma 4.4, it follows that q itself is an accepting run. Since, q is an arbitrary run of A_1, this means that all runs of A_1 on σ are accepting, and hence $\sigma \in \mathcal{L}_{\forall}(A_1)$. $\square$

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

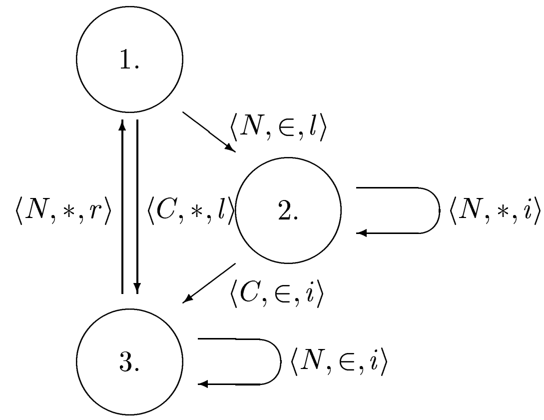

Figure 2: Automaton Φ

excepting that its initial state is C, and E be same as P. Let ϕ be the existential automaton on 2-d strings, given in figure 2, that states that exactly one process is in state C at any time. Here all the states of the automaton are final states.

To verify this property for all circular networks of the type (I, P^{i+1}, E) for all $i \geq 0$, we use the existential 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 occurrence of a sequence of global states in 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 4.3. This can be shown by exhibiting

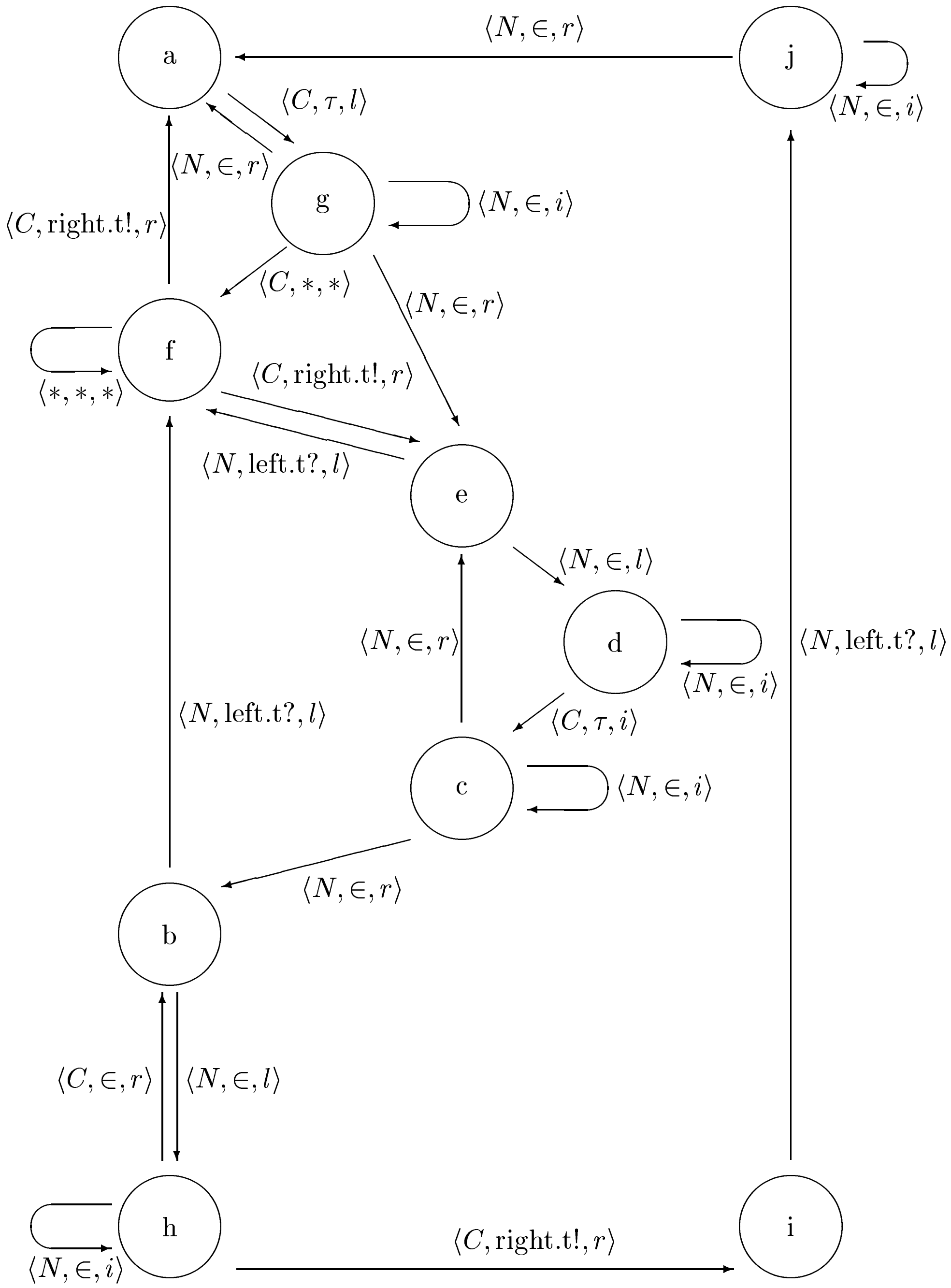

Figure 3: The invariant automaton A

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 Streett automaton. F consists of two pairs (Q, F_1) and (Q, F_2) where Q is the set of all states of the automaton; F_1 consists of all states other than a and g, F_2 consists of all states other than c, d, e. Hence a run is accepting if some state from F_i appears infinitely often for each $i = 1, 2$

5 An Automatic System for verification of Linear Networks

In this section we describe a method that automatically checks correctness of families of linear networks of the form (I, P^i, E) for all $i \geq c$ where $c \geq 0$. Let the correctness property be specified by 2-d automaton ϕ.

5.1 Correctness without Fairness Requirements

In our procedure one can use either universal or existential automata. We specify the procedure using universal automata. This is due to the fact that it is easier to construct universal automata, as we show later. Our procedure is given by the following fixed point computation.

Procedure Fix:

1. Construct an automaton A_0 such that $L_\forall(A_0) = \{\sigma : \sigma \circ E \subseteq L_\forall(\phi)\}$. Set i to 0.

2. Repeat the following step until $i > 0$ and $L_\forall(A_{i-1}) = L_\forall(A_i)$

   Increment i. Construct automata A_i and B_i such that $L_\forall(B_i) = \{\sigma : \sigma \circ P \subseteq L_\forall(A_{i-1})\}$ and $L_\forall(A_i) = L_\forall(A_{i-1}) \cap L_\forall(B_i)$.

3. If $\mathcal{COMP}(I, P^c) \subseteq L_\forall(A_i)$ then output "yes". Otherwise output "no".

It is to be noted that in the above procedure, in general, step (2) may not terminate. However, if it terminates then the final automaton satisfies all the conditions of theorem 4.1.

Lemma 5.1 *Assume that procedure Fix terminates after k iterations of step (2) and let $\mathcal{A}$ denote the automaton A_k. If the above procedure outputs "yes"* *in step (3) then $\mathcal{A}$ satisfies all the conditions of theorem 4.1 and the family $\{I, P^{c+j}, E\}_{j\geq 0}$ satisfies the specification ϕ. If the above procedure outputs "no" then the family of linear chains does not satisfy ϕ.*

Proof: It is easy to see that, for each $i > 0$, $L_\forall(A_i) \subseteq L_\forall(A_{i-1})$ and $L_\forall(A_0) \circ E \subseteq L_\forall(\phi)$. From the monotonicity of the operator $\circ$, it follows that $L_\forall(A_i) \circ E \subseteq L_\forall(\phi)$, and hence $L_\forall(\mathcal{A}) \circ E \subseteq L_\forall(\phi)$. Since $\mathcal{A}$ is a fix point of the computation in step (2), we see that $L_\forall(\mathcal{A}) \circ P \subseteq L_\forall(\mathcal{A})$. Thus conditions (2) and (3) of theorem 4.1 are satisfied. If procedure outputs "yes" in step (3) then it follows that condition (1) of theorem 4.1 is also satisfied. Assume that the procedure outputs "no" answer. It is easy to see that $L_\forall(\mathcal{A}) = \{\sigma : \forall j \geq 0, \sigma \circ P^j \circ E \subseteq L_\forall(\phi)\}$ (here $\sigma \circ P^j \circ E$ denotes $\sigma \circ P ... \circ P \circ E$). Since the procedure output a "no" answer, this implies that for some for some $\sigma \in \mathcal{COMP}(I, P^c)$, and some $j \geq 0$, $\sigma \circ P^j \circ E$ is not contained in $L_\forall(\phi)$. That is, for some $j \geq 0$, $\mathcal{COMP}(I, P^c) \circ P^j \circ E$ is not contained in $L_\forall(\phi)$. Using lemma 4.1, we see that $\mathcal{COMP}(I, P^c) \circ P^j \circ E = \mathcal{COMP}(I, P^{c+j}, E)$. This implies that $\mathcal{COMP}(I, P^{c+j}, E)$ is not contained in $L_\forall(\phi)$. Thus the procedure correctly outputs "no" answer. □

In steps (1) and (2) of procedure Fix, we need to compute the automata A_0 and B_i, respectively. For this, we use a function, subsequently given below, that computes a universal automaton $f(V, A)$ given a process V and another universal automaton A. In step (1) of procedure Fix we compute $A_0 = f(E, \phi)$, and in step (2) we compute $B_i = f(P, A_{i-1})$. Since we are considering universal automata, construction of A_i from A_{i-1} and B_i is trivial.

Construction of $f(V, A)$

Let $V = (S, R, S_0)$ be a process and $A = (Q_0, \Sigma, Q, \delta, F)$ be a universal Buchi automaton on 2-d strings. Now, we describe the construction of the automaton $f(V, A)$. Essentially, $f(V, A)$ accepts the set of all 2-d strings σ such that $\sigma \circ V \subseteq L_\forall(A)$. $f(V, A)$ employs non-determinism extensively. Each of its runs on a 2-d string σ corresponds to a run of A on some string in $\sigma \circ V$, and vice versa. Thus, $f(V, A)$ accepts σ iff A accepts all strings in $\sigma \circ V$.

We use the notation given below. Each input to A is a 4-tuple of the form (s, a, ind, s') where s, s' are process states, a is an action symbol, and ind is an indicator specifying the position of the symbol in each row.

Each state of $f(V, A)$ is a 4-tuple of the form $(s, q, flag1, flag2)$ where

s and q are the states of V and A respectively; $flag1$ is a flag used to keep track that no more than one computational step takes place in each global transition; $flag1$ takes any of three values $0, 1, 2$; $flag2$ is used to indicate if a final state of A has been encountered in a row, and it takes values $0, 1$. The set of initial states of $f(V, A)$ are those states of the form $(s, q, 0, 0)$ where s, q are the initial states of V and A respectively, i.e. $s \in S_0$ and $q \in Q_0$. For any state $u = (s, q, x, c)$ of $f(V, A)$, we let $u.V, u.A, u.flag1, u.flag2$ denote the states of A, V (i.e. s and q) and the values of $flag1$ and $flag2$ (i.e. x and c) in the state u. Intuitively, $f(V, A)$ works as follows whenever it is in state u.

Whenever it gets an input which is not the right most symbol in a row then the $u.A$ changes according to the transitions of A on that input, while the $u.V$ does not change. Whenever it gets an input which is the right most symbol of the current row, i.e. global state, $f(V, A)$ makes a single transition simulating two steps of the automaton A; the first of these two steps is the transition of A on the current input, and the second step is the transition of A on a hidden input (i.e. not visible) denoting a transition of V; in this case, $u.V$ may also change according to the transition of V.

The value of $u.flag1$ is updated so as to indicate if any computational step has occurred in the current row; a value of 0 for this flag indicates that no computational step has occurred in the current row thus far; value 1 indicates that a communication step is taking place between two adjacent processes and the automaton has only seen the transition of one of the two processes; value 2 indicates that a complete computational step has already taken place in the current row.

The value of $u.flag2$ indicates whether a final state of the automaton has been seen. Thus, the final states of $f(V, A)$ are all those states u such that $u.flag2 = 1$.

A detailed description of the automaton $f(V, A)$ is given in the appendix.

Lemma 5.2 *Let A be a universal automaton. Then for any 2-d string σ the following property holds: $\sigma \circ V \subseteq L_\forall(A)$ iff $\sigma \in L_\forall(f(V, A))$.*

Proof: Consider a 2-d string σ of width n for some $n > 0$. For any run $u = u_0, ..., u_i, ...$ of $f(V, A)$, let $proj(u)$ denote the sequence $u_0.A$, $u_1.A, ..., u_i.A, ...$, i.e. the sequence of states of A in u. For any run $v = v_0, v_1, ..., v_i, ...$ of A, let $proj'(v)$ denote the subsequence of v obtained by dropping v_{jn+j-1} for each $j > 0$, i.e. by dropping v_n, v_{2n+1}, v_{3n+2}, etc. From the construction of $f(V, A)$, it is not difficult to see that there exists a run $u = u_0, u_1, ...$ of $f(V, A)$ on σ iff there exists a run $v = v_0, v_1, ...$ of A on some element $\sigma' \in \sigma \circ V$ such that $proj(u) = proj'(v)$ and the following condition is satisfied: a final state of $f(V, A)$ (i.e. a state u' with $u'.flag2 = 1$) appears infinitely often in u iff a final state of A appears infinitely often in v. From this observation, the lemma easily follows. □

5.2 Handling Fairness Requirements

In order to handle fairness, we modify the procedure Fix as follows: in steps (1) and (2) replace the operator $\circ$ by $\circ_f$; in step (3), output "yes" if $\mathcal{FCOMP}(I, P^c) \subseteq L_\forall(A_i)$, otherwise output "no". Let the resulting procedure be called $Fair\_Fix$. With this modification, it is straightforward to prove that lemma 5.1 holds for checking correctness under fairness when we use $Fair\_Fix$ instead of Fix.

As indicated earlier, it is convenient to use Streett automata in case of fairness. It turn out that it is enough if we use streett automaton in which the acceptance condition has a single pair of subsets of states. Thus, we assume that all our automata are of this form.

In order to compute the automata A_0 and B_i in steps (1) and (2) of $Fair\_Fix$, we use a function that computes an automaton $g(V, A)$ given an automaton A and process V. Essentially $g(V, A)$ accepts the set of all σ such that $\sigma \circ_f V \subseteq L_\forall(A)$. In $Fair\_Fix$, we compute $A_0 = g(E, \phi)$ and $B_i = g(P, A_{i-1})$.

Construction of $g(V, A)$

In the previous section, we provided a method for constructing the automaton $f(V, A)$. Now, we show how this construction can be extended to obtain the desired automaton $g(V, A)$. (The construction of $g(V, A)$ assumes that the processes I, P of the linear chain are known, i.e. it depends on these process definitions also, although they are not explicitly specified as parameters to g; they are needed for fairness purposes). Let $V = (S, R, S_0)$ be a process and $A = (Q_0, \Sigma, Q, \delta, F)$ be a universal Streett automaton where $F = \{(G, H)\}$ (recall that a run of A is an accepting run if the following condition is satisfied: if some state belonging to G appears infinitely often on the run then some state belonging to H also appears infinitely often). We call V as the hidden process.

The construction of $f(V, A)$ is modified as follows to construct $g(V, A)$. Each state u of $g(V, A)$ will have the previous three components $u.V, u.A$

and $u.flag1$, together with five other components $u.pV$, $u.flag3$, $u.flag4$, $u.flag5$ and $u.flag6$ (note that we have discarded $flag2$). Here $u.pV$ is used to remember the process state in the previous input. This field together with $flag3$ is used to monitor the fairness condition of the rightmost process in each row of the input string σ, while $flag4$ is used for monitoring the fairness condition of the hidden process (i.e. of V). $flag5$ and $flag6$ are used to note if a G or a H state of automaton A has been seen. These flags are set and reset as follows. $flag3$ is set when the rightmost process is detected to be disabled or executed; flag4 is set when the hidden process, i.e. V, is detected to be disabled or executed. $flag5$ is set whenever the $u.A$ is in G, i.e. whenever automaton A goes into some state in G. If all these three flags have value 1 in a state u then in the next state all these are reset, i.e. simultaneously set to 0. $flag6$ is set whenever its current value is 0 and $u.A \in H$; otherwise, it is reset. The acceptance condition for $g(V, A)$ consists of the single pair (G', H') where G' is the set of all u such that $u.flag3 = u.flag4 = flag5 = 1$, and H' is the set of all v such that $v.flag6 = 1$.

Roughly speaking, the set of all runs of $g(V, A)$ on σ such that some state u, with $u.flag3 = u.flag4 = flag5 = 1$, appears infinitely often will correspond to the set of runs r of A on 2-d strings in $\sigma \circ_f V$ such that some state in G appears infinitely often in r. Similarly, the set of runs of $g(V, A)$ in which some state v, such that $v.flag6 = 1$, appears infinitely often corresponds to the set of runs of A on elements in $\sigma \circ_f V$ such that some state in H appears infinitely often. As a consequence all the runs of $g(V, A)$ on σ are accepting runs iff all the runs of A on elements in $\sigma \circ_f V$ are accepting.

A more detailed description of $g(V, A)$ is given in the appendix.

Lemma 5.3 *For any 2-d string σ and universal automaton A, the following condition holds: $\sigma \circ_f V \subseteq L_\forall(A)$ iff $\sigma \in L_\forall(g(V, A))$.*

Proof: Consider a 2-d string σ. For runs u, v of $g(V, A)$ and A, respectively, define $proj(u)$, $proj'(v)$ as in the proof of lemma 5.2. From the construction of $g(V, A)$, it is not difficult to see that there exists a run $u = u_0, u_1, ...$ of $f(V, A)$ on σ iff there exists a run $v = v_0, v_1, ...$ of A on some element $\sigma' \in \sigma \circ V$ such that $proj(u) = proj'(v)$ and the following conditions are satisfied:

- Some state in G' appears infinitely often in u (i.e. for infinitely many values of j, $u_j.flag3 = 1$ and $u_j.flag4 = 1$ $u_j.flag5 = 1$) iff σ' is fair with respect to the last two processes (i.e. $\sigma' \in \sigma \circ_f V$) and some state in G appears infinitely often in v.

- Some state in H' appears infinitely often in u iff some state in H appears infinitely often in H.

The construction $g(V, A)$ uses, for each state u, three additional flags together with an additional variable $u.pV$ compared to the construction of $f(V, A)$, we have alternate but more complex construction that uses fewer flags.

5.3 Implementation

The automatic verification described in subsection 5.1 has been implemented. In procedure Fix, given in subsection 5.2, we need to check for language containment between two given universal automata; this is needed in step (2) where we need to check for the fix point condition. We check this by checking for the existence of a universal simulation relation. We have implemented and used standard algorithms for checking the existence of universal simulation relations described in 4.4. Note that existence of a universal simulation relation is a sufficient condition for language containment, but not a necessary one. In step 3 also we need to check for language containment; here we used standard algorithms for checking language containment. As a consequence if the implemented system gives an "yes" answer then the family of systems satisfy the correctness specification. If the system outputs a "no" answer then the family of systems does not satisfy the correctness specification.

We have also done some optimizations in order to reduce the number of states in the invariant automata that is being generated. In each iteration our implementation compresses the automaton B_i using standard bisimulation reductions.

There are some other minor differences between the implementation and the description in the previous sections. For example, in the implementation, we allowed multiple computational steps to take place in each row, i.e. in each global step, with the restriction that each process can only involve in at most one computational step at a time. This does not effect the correctness of the procedure.

The implemented system has been applied to many of the standard examples successfully. More specifically, we have considered the various versions of the mutual exclusion algorithm on linear networks and circular networks. We verified the mutual exclusion property. We have also checked the more complex example considered in [CGJ95] where processes pass the token in the clockwise direction and signals are passed in the anti-clockwise direction. In this example the token is passed only if there is a request for it. The request for the token is passed in the anti-clockwise direction using signals. In this case, the fix-point computation terminated after 3 iterations, The initial correctness automaton had seven states and roughly about 100 transitions (the number of transitions is proportional to the number of inputs which is high). The final invariant automaton had 80 states.

We have also implemented the method for verification of correctness under fairness given in 5.2. We checked liveness properties of the different token passing algorithms using the implemented system. For the simple version of the token ring algorithm, we checked that each process gets into critical section infinitely often.

6 Conclusions

In this paper we have proposed an induction based approach for verifying safety and liveness properties of families of linear and circular networks. Our approach employs automata on two dimensional strings. It has been shown to be complete for verifying safety properties. We have also given automated methods for verification. These methods generate the invariant automaton automatically.

The proposed methods have been implemented and have been successfully tested on some examples. As part of future work, it will be interesting to apply our methods 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 Parameterized Networks Using Abstraction and Regular Languages. CONCUR 95; revised version appeared in ACM TOPLAS, Vol. 19, No. 5, Sept. 1997.

[CG87] E. M. Clarke, and O. Grumberg: Avoiding the State Explosion Problem in Temporal Logic Model-checking 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 conference, Jan 1995.

[EN96] E. A. Emerson and K. S. Namjoshi: Automatic Verification of Parameterized 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 Order Approach to Model-checking. Proc 6th IEEE Symposium on Logic in Computer Science, pp 406-415, Amsterdam, July 1991.

[K97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli and E. Shahar: Symbolic Model-checking with Rich Assertional Languages. Proceedings of 9th International Conference, CAV '97, Haifa, Israel, June 1997.

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

[LHR96] D. Lesens, N. Halbwachs, P. Raymond: Automatic Verification of Parametrized Linear Networks of Processes, Proceedings of ACM Symposium on Principles of Programming Languages, 1996.

[MG91] R. Marelly and O. Grumberg, GORMEL— Grammar Oriented Model-checker. Tech Report 697, Technion, Haifa, Israel 1991.

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

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

[Si97] A. P. Sistla: Parameterized Verification of Linear Networks using Automata as Invariants. Proceedings of International Conference, CAV'97, Haifa, Israel, June 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.

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

7 Appendix

7.1 Detailed Description of $f(V, A)$

The automaton $f(V, A)$ has a transition from state u to state v on input $\delta = (t, a, ind, t')$ if u, v satisfy the conditions given in the following cases (note that in δ, a denotes the action symbol; ind denotes the position, in the current row i.e. left, internal or right; t and t' denote the current and next states of the process being scanned by the automaton)

1. This corresponds to the case when the input is either the left most symbol or an internal symbol in a row, i.e. $ind \neq r$ (it is not the right most symbol). In this case, the following conditions should be satisfied: $v.V = u.V$ (i.e. the state of V in u does not change); A has a transition from $u.A$ to $v.A$ on the input δ; if $a = i\tau$ then $u.flag1 = 0$ and $v.flag1 = 2$ (this indicates that one computational step has taken place in the current row); If $a = \epsilon$ then $v.flag1 = u.flag1$; If $a = c\tau$ then $u.flag1$ equals 0 or 1 and $v.flag1 = u.flag1 + 1$ (this indicates one transition of synchronized communication between

adjacent processes); $v.flag1 = 1$ if $v.A$ is a final state of A; otherwise $v.flag1 = 0$.

2. These transitions correspond to the case when the input is the rightmost symbol in a row, i.e. $ind = r$. This case is divided in to the following sub-cases. In all the sub-cases, $v.flag1 = 0$ since the automaton will be at the beginning of next row.

   (a) In this sub-case, the input symbol has a communication action. $u.flag1 = 0$ (denotes no computational step has taken place thus far in the current row); V has a transition from $u.V$ to $v.V$ on action $\overline{a}$; there is a state q' of A such that A has transitions from $u.A$ to q' on input $(t, c\tau, i, t')$, and from q' to $v.A$ on input $(u.V, c\tau, r, v.V)$; $v.flag2 = 1$ if either q' or $v.A$ is a final state of A; otherwise, $v.flag2 = 0$.

   It is to be noted that on the same input symbol the automaton $f(V, A)$ can go to multiple states, i.e. there can be lot of non-determinism in the automaton. This happens if process V has transitions to more than one state from $u.A$ on the action $\overline{a}$.

   (b) In this case, the input symbol has $i\tau$ or $c\tau$ action, i.e. $a = i\tau$ or $a = c\tau$. The following conditions should be satisfied for the transition to take place: $u.V = v.V$; there is a state q' such that A has a transition from $u.A$ to q' on input (t, a, i, t'), and from q' to $v.A$ on input $(u.V, \epsilon, r, u.V)$; if $a = c\tau$ then $u.flag1 = 1$; if $a = i\tau$ then $u.flag1 = 0$; if q' or $v.A$ is a final state of A then $v.flag2 = 1$; otherwise $v.flag2 = 0$.

   (c) In this case, the input symbol has ϵ (i.e. empty) action, i.e. $a = \epsilon$ and the following conditions are satisfied. There exists a state q' such that A has transitions from $u.A$ to q' on input (t, ϵ, i, t'), and from q' to $v.A$ on input $(u.V, b, r, v.V)$ for some action symbol b where b is either ϵ or $i\tau$ or is a communication action; if $b = \epsilon$ then $v.V = u.V$ (this denotes an empty row and hence a stuttering step) ; if $b \neq \epsilon$ then V has a transition from $u.V$ to $v.V$ on action b and $u.flag1 = 0$; $v.flag2 = 1$ if either q' or q'' is a final state of A; otherwise, $v.flag2 = 0$.

   In this case, $f(V, A)$ can have transitions to many states on the same input; each such transition corresponds to a transition of V to different states for different values of b.

It is to be noted that $f(V, A)$ will be non-deterministic if A is non-deterministic. However, it can have non-determinism even if A is deterministic. Such non-determinism can be introduced in in cases 2(a) and 2(c) due to many possible transitions of V from state $u.V$.

7.2 Detailed Description of $g(V, A)$

This subsection gives a more detailed description of the automaton $g(V, A)$.

The automaton $g(V, A)$ has a transition from state u to state v on input $\delta = (t, a, ind, t')$ iff all the conditions as defined in the case of $f(V, A)$ are satisfied with the following changes. The values of $v.A, v.V$ and $v.flag1$ are defined as in the case of $f(V, A)$. In all cases $v.pV = t$ (thus, it stores the state of the previous process scanned by $g(V, A)$). The values of $flag3, flag4, flag5$ and $flag6$ in v are related to those in u according to the following cases. Each of these cases and sub-cases correspond to the ones in the definition of $f(V, A)$.

1. In this case $ind \neq r$. If $u.flag6 = 0$ and $v.A \in H$ then $v.flag6 = 1$; otherwise, $v.flag6 = 0$. If $u.flag5 = 0$ and $v.A \in G$ then $u.flag5 = 1$. If $u.flag3 = u.flag4 = u.flag5 = 1$ then $v.flag3 = v.flag4 = v.flag5 = 0$; otherwise, the values of $flag3, flag4$ and $flag5$ in v should be same as their respective values in u.

2. In this case $ind = r$, i.e. the input is the rightmost symbol in a row. In this case, if the values of $flag3, flag4$ $flag5$ in the state u are all 1 then in state v they are all reset to 0. Otherwise, these three flags are updated as described below. Let q' be as defined in cases (2a),(2b) and (2c) in the construction of $f(V, A)$. In these cases, $v.flag5 = 1$ if the state $q' \in G$ or $v.A \in G$; otherwise $v.flag5 = u.flag5$. Similarly, $v.flag6 = 1$ if $u.flag6 = 0$ and either the state $q' \in H$ or $v.A \in H$; otherwise, $v.flag6 = 0$.

   $v.flag3 = 1$ if the rightmost process is disabled or is executed, otherwise $v.flag3 = u.flag3$. The disabledness of the this process is determined by the state of the process which is t, by the state of the previous process which is $u.pV$ and the state of the hidden process which is $u.V$. Since current rightmost process is P (as we are considering chains of the form (I, P^i, E) and the one before is W (here W is either P or I) then on input δ, as given above, the rightmost process is disabled under the following conditions: there is no internal transition of P from state t, and there is no communication transition of P from state t that can synchronize with some transition of V from state $u.V$ or with some transition of W from state $u.pV$.

When $ind = r$ then $v.flag4 = 1$ if the hidden process (i.e. V) is either disabled or is executed; otherwise, $v.flag4 = u.flag4$. The disabledness of the hidden process is determined by its current state, which is $u.V$, and the state of the rightmost process, which is t; more specifically, the hidden process is disabled if there is no internal transition of V from state $u.V$ and there is no communication transition of V from state $u.V$ that can synchronize with some transition from state t of P. More specifically, the values of $flag3$ and $flag4$ are defined as in the following cases (we assume that σ is the input 2-d string and δ is the current symbol in σ)

(a) The action a is a communication action (note this denotes a communication with the hidden process in all corresponding rows of all 2-d strings in $\sigma \circ V$; the rightmost process as well as the hidden process are executed). $v.flga3 = v.flag4 = 1$.

(b) The action a is $i\tau$ or $c\tau$ (this is an execution of the rightmost process; the hidden process cannot execute in this row of any string in $\sigma \circ V$). $v.flag3 = 1$. If the hidden process is enabled then $v.flag4 = 1$; otherwise $v.flag4 = u.flag4$.

(c) The action a is ϵ. This indicates that rightmost process is not executed in the current row in all strings in $\sigma \circ V$. $v.flag3 = 1$ if the rightmost process is enabled; otherwise $v.flag3 = u.flag3$. $flag4$ is updated according to the value of b (see (2c) in the construction of $f(V, A)$). If $b \neq \epsilon$ or the hidden process is disabled then $v.flag4 = 1$; otherwise, $v.flag4 = u.flag4$.