Automatic Temporal Verification of Buffer Systems

A. Prasad Sistla
Dept of Electrical Engineering and Computer Science, University of Illinois at Chicago
e-address: sistla@uicbert.eecs.uic.edu

Lenore D. Zuck
Department of Computer Science, Yale University
e-address: zuck@cs.yale.edu

October 16, 1991

1 Introduction

Propositional Linear Time Temporal Logic (TL) was introduced in [Pnu77] as a formal system for reasoning about concurrent programs. Since then it has been widely used for specification and verification of concurrent systems.

Most formal systems that allow temporal-like reasoning use the eventuality operator as the main modal operator. In this paper we consider a temporal logic that uses $\overrightarrow{\Diamond}$ (sometimes in the future) and $\overleftarrow{\Diamond}$ (sometimes in the past) as the only modal operator; we term this logic RTL (Restricted Temporal Logic). At first glance RTL seems to be a very limited language. Yet, if we are concerned with properties of concurrent programs, there are convincing indications that RTL is an adequate language:

Owicki and Lamport, when giving proof rules for liveness properties, considered only formulae of the form $\overrightarrow{\Box}(p \rightarrow \overrightarrow{\Diamond}q)$ ([OL82]) which are RTL formulae; Chandy and Misra conjectured that there is essentially one class of liveness properties of concurrent programs, namely, the class of *progress* properties ([CM86]); in their propositional version, progress properties are easily definable by RTL. Other specification techniques ([Lam83]) use state transition systems to specify safety properties and RTL to specify liveness properties. It is hardly surprising that most temporal properties of distributed programs discussed in the literature are given in RTL.

A natural problem associated with choosing a specification language for programs is how easy is it to verify that programs satisfy their specification. Ideally, we would like an automatic verifier that, given a program and its specification, would decide whether the program satisfies its specification. There are several *model checking* methods that are proposed in the literature for obtaining this goal which use (the full) temporal logic as the specification language (cf. [CES86, EL85, LP85]). Recently, Wolper and Vardi ([VW86]) have advocated the *automata theoretic approach* which is a variant of model checking. There, both the program and the specification are brought to the form of a finite-state graph, and model checking reduces to the emptiness problem for (ω-) finite-state automata.

All the model checking techniques that we are aware of apply only to programs which are either finite-state or those that, for verification purposes, can be considered finite-state. Many programs for distributed systems are, however, infinite-state. In particular, systems of (possibly finite-state) processes that communicate over unbounded message buffers are inherently infinite-state.

A major obstacle in verifying properties of systems that use message buffers is the complexity of the axiomatization of the behavior of the buffers used. As shown in [SCFM84], the theory of unbounded fifo buffers in the full TL is Π_1^1 complete. Hence, one cannot hope to have a model-checking procedure for verifying TL properties of systems that use unbounded fifo message buffers. However, once TL is replaced with a version of RTL that has $\overrightarrow{\Diamond}$ as a single modal operator, the theory of unbounded message buffers is co-NP-complete ([SZ90]).

In this paper, we modify the result in [SZ90] and show that the theory of unbounded message buffers in RTL is also co-NP-complete. Based on this, we present a "simple" automatic verification method that applies to a certain class of RTL formulae and (not necessarily finite-state) programs of distributed systems whose processes communicate by means of fifo message buffers. Our method is based on the principle that can be roughly stated as follows:

Given a system of n processes, $P_1, \ldots, P_n$ that communicate over unbounded message buffers and a formula ψ in an admissible logic L, then in order to show that ψ holds over all the executions of the system, it suffices to find n formulae $\phi_1, \ldots, \phi_n$ in L such that:

1. each P_i satisfies ϕ_i, and

2. $(\bigwedge_i \phi_i \rightarrow \psi)$ is in the theory of the message buffers in the logic L.

In order for the above idea to be applicable the logic L has to be admissible. Admissible logics are defined in section 2. RTL is an admissible logic and since the theory of fifo buffers in RTL is co-NP-complete, we can use the above principle for proving RTL properties of systems that communicate through unbounded fifo buffers. Intuitively, we can think of the above method to be approximating the behavior of each process P_i by the formula ϕ_i. This gives us a sound proof method for proving RTL properties of systems that communicate through unbounded fifo buffers.

It can easily be shown that the above method is *complete* for a class of processes if the logic L is expressive for this class. By taking the logic L to be the Propositional Temporal Logic without the "nexttime" operator and by considering unbounded unordered[1], we can use the above method for proving properties of systems of processes that communicate through unbounded *unordered* buffers whose message alphabet is finite. In this case, it can easily be shown L is admissible and the theory of unbounded unordered buffers in L has been shown to be decidable in [SCFM84]. If we are only considering finite state processes then L is expressive for this class, and we get a complete and fully automatic proof method for this class of processes.

Temporal logics are usually interpreted over sequences of states, and formulae that express properties of message buffers use the *send* and *receive* actions to the buffers as atomic propositions. We therefore have to interpret the logic over both states and actions. Consequently, we consider executions sequences that explicitly have the two. A natural such model is the I/O automaton model (cf. [LT87, LT89]). This model also allows to abstract away the (application) programs run by the processes and to concentrate on the interaction between the buffers and the processes. Finally, the model has inherent compositionality properties. We therefore chose it as the model on which to demonstrate our ideas. We would like to stress that this choice is for purposes of convenience of exposition, and our results do not depend on the I/O automaton model.

2 The Formal Framework

2.1 Admissible Logics

Let Φ be a finite set of *state assertions*. We are interested in logics for specifying properties of computations which include states as well as actions. With this as our motivation, we assume that the set Φ is partitioned into two nonempty disjoint sets, denoted by Φ_s and Φ_a. Intuitively, the members of Φ_s can be true only in states that represent states of processes and those in Φ_s can be true only in states that represent actions.

We define *admissible logics*, which are logics that satisfy certain properties defined later in this section. The formulas of such logics are constructed from state assertions in Π using the boolean connectives $\neg$ and $\vee$, and some other operators (e.g., temporal ones), and are interpreted over sequences of subsets of Φ.

Let L be such a logic. For a formula φ of L (an *L-formula*), let $\mathsf{prop}(\varphi)$ denote the set of state assertions in φ. A *model* $\mathcal{M}$ for a logic is a triple (Π, S, I) where $\Pi \subseteq \Phi$, S is a (possibly infinite) set of *states* and $I: S \rightarrow (2^\Pi - \emptyset)$ is an *evaluation* mapping each state $s \in S$ to a nonempty set $I(s)$, the set of state assertions true in s and such that either $I(s) \cap \Phi_s$ is empty or $I(s) \cap \Phi_a$ is empty. We say that a state $s \in S$ is *regular* if $I(s) \cap \Phi_s$ is nonempty; we say that a state $s \in S$ is *action* if $I(s) \cap \Phi_a$ is nonempty. It should be clear that every state in S is either regular or action. Given a model $\mathcal{M} = (\Pi, S, I)$, a *computation* σ is an infinite sequence of states, i.e.,

$$\sigma : s_0, s_1, \ldots \qquad s_i \in S.$$

We assume that a satisfiability relation $\models_L$, between a model $\mathcal{M} = (\Pi, S, I)$, a computation σ, and an L-formula φ, where $\mathsf{prop}(\varphi) \subseteq \Pi$, is defined so that the following properties are satisfied:

- $(\mathcal{M}, \sigma) \models_L$ true and $(\mathcal{M}, \sigma) \not\models_L$ false for every $\sigma \in S^\omega$.

[1] In unordered buffers the messages can be delivered in any order.

- For a state assertion $Q \in \Phi$, $(\mathcal{M}, \sigma) \models_L Q$ *iff* $Q \in I(s_0)$.
- $(\mathcal{M}, \sigma) \models_L \neg\varphi$ iff $(\mathcal{M}, \sigma) \not\models_L \varphi$.
- $(\mathcal{M}, \sigma) \models_L \varphi_1 \vee \varphi_2$ iff $(\mathcal{M}, \sigma) \models_L \varphi_1$ or $(\mathcal{M}, \sigma) \models_L \varphi_2$.
- For every model $\mathcal{M}' = (\Pi' \supseteq \mathsf{prop}(\varphi), S', I')$ and computation $\sigma' = s'_0, \ldots, s'_i, \ldots$ over S' such that $I'(s'_i) \cap \mathsf{prop}(\varphi) = I(s_i) \cap \mathsf{prop}(\varphi)$ for every $i \geq 0$,

  $$(\mathcal{M}, \sigma) \models_L \varphi \qquad \text{iff} \qquad (\mathcal{M}', \sigma') \models_L \varphi.$$

  In other words, the satisfaction of a φ by σ only depends on the evaluation of $I(\sigma) \cap \mathsf{prop}(\varphi)$.

(Additional boolean connectives (such as $\wedge$, $\rightarrow$, $\leftrightarrow$) are defined in the usual way.)

If $(\mathcal{M}, \sigma) \models_L \varphi$, we say that σ satisfies φ in $\mathcal{M}$.

For any sequence α whose elements are from a set A, we let $\alpha|A'$ denote the *restriction of* α *to* A'. In other words, $\alpha|A'$ is the subsequence of α that includes all the A' elements of α.

Let $\mathcal{M} = (\Pi, S, I)$ be a model. For any $s \in S$ and $\Pi' \subseteq \Pi$, we say that s is a Π'*-state* if $I(s) \cap \Pi' \neq \emptyset$. We use $\sigma|\Pi'$ to denote σ's restriction to the Π'-states, that is,

$$\sigma|\Pi' = \sigma|\{s \in S : s \text{ is a } \Pi'\text{-state}\}.$$

We say that L has *projection property* if for every L-formula φ and every finite set Π such that $\Pi \subseteq \Phi$, there exists a formula $\varphi_{(\Pi)}$ such that for every model $\mathcal{M} = (\Pi', S, I)$ where $\Pi' \supseteq (\mathsf{prop}(\varphi) \cup \Pi)$ and every computation σ such that $\sigma|\Pi$ is infinite and the first state in σ is a Π-state,

$$(\mathcal{M}, \sigma) \models_L \varphi_{(\Pi)} \qquad \text{iff} (\mathcal{M}, \sigma|\Pi) \models \varphi$$

The logic L is *admissible* if it has the projection property. In the sequel we will only consider admissible logics. The projection property does not imply a unique projected formula $\varphi_{(\Pi)}$. However, we fix such a formula and let $\varphi_{(\Pi)}$ denote it. It easily follows from the definition that for every L-formula φ, $\varphi \leftrightarrow \varphi_{(\Pi)}$ holds in every model $\mathcal{M}$ whose states are all Π-states.

Let $\sigma = s_0, s_1, \ldots$ be a computation over S. If σ' is obtained from σ by possibly duplicating some of the regular states in σ, then we say that σ' is a *stuttered extension of* σ. We say that an L-formula φ is *invariant under stuttering* if for every model $\mathcal{M}$, a computation σ satisfies φ (in $\mathcal{M}$) iff every stuttered extension of σ satisfies φ (in $\mathcal{M}$).

2.2 I/O Automata

A concurrent system is often described by the specification of its possible interactions with the environment, i.e., its *observable behavior*. In the I/O automaton model, the interface between the system and its environment is described as interleaved sequences of *actions* which are called *behaviors*. When the system runs in a certain environment, certain behaviors may arise. The *specification* says which are the allowed behaviors.

Actions at the interface between the system and its environment are of two kinds. *Input* actions originate in the environment and are imposed by the environment on the system. This means that they can occur at any time and are not under the control of the system. *Output* actions are generated by the system and are imposed by it on the environment.

Formally, given two disjoint sets of actions, $inp(S)$ and $out(S)$, a specification S is a subset $beh(S)$ of the set of (finite or infinite) sequences over $acts(S) = inp(S) \cup out(S)$.

While a specification describes the interaction of a system with its environment, an *I/O automaton* is a state machine which models the system. An I/O automaton A therefore has input actions, $inp(A)$, and output actions, $out(A)$. In addition, it has *internal actions*, $int(A)$, which are not observable by the environment. Let $acts(A)$ denote the union of these three (mutually disjoint) sets of actions. The automaton also has *states*, $states(A)$), *transitions*, $trans(A) \subseteq states(A) \times acts(A) \times states(A)$, *initial states*, $i\text{-}state(A) \subseteq states(A)$, and a fairness condition, $fair(A)$, described as a partition on $out(A) \cup int(A)$. See [LT87, LT89, AAF+] for an elaborate discussion.

An *execution* η of A is a (possibly infinite) sequence of the form:

$$s_0, a_1, s_1, a_2, \ldots$$

where s_0 is an initial state of A and for every $i \geq 0$, (s_i, a_{i+1}, s_{i+1}) is a transition of A. The execution η is *fair* if it is infinite, and for every set of actions $acts' \in fair(A)$, either actions from $acts'$ are taken infinitely many times in η (i.e., for infinitely many i's, $a_i \in acts'$), or actions from $acts'$ are disabled infinitely many times in η (i.e., for infinitely many i's, for all actions $a \in acts'$, there is not s' such that $(s_i, a, s') \in trans(A))$[2].

If A and B are both I/O automata whose only mutual actions are input of one and output of the other, then the composition of A and B, $A \circ B$, is an I/O automaton such that:

1. $out(A \circ B) = out(A) \cup out(B)$, $int(A \circ B) = int(A) \cup int(B)$, and $inp(A \circ B) = inp(A) \cup inp(B) - (out(A) \cup out(B)$.
2. $states(A \circ B) = states(A) \times states(B)$, and $A \circ B$'s initial states are the cartesian product of its components' initial states.
3. $A \circ B$'s transitions are such that only the components to which the action belongs is affected, i.e., $((s_A, s_B), a, (s'_A, s'_B)) \in trans(A \circ B)$ if $(s_A, a, s'_A) \in trans(A)$ when $a \in acts(A)$ and $s_A = s'_A$ otherwise, and similarly for s'_B.
4. The fairness condition on $A \circ B$ is the union of the fairness conditions of its components.

The definition of composition of two I/O-automata is extended in the obvious way to the definition of the composition of and n I/O automata.

Let α be an execution of the composition of $A_1, \ldots, A_n$. For every $i = 1, \ldots, n$, let $\alpha[i]$ denote the execution of A_i which is defined by α. That is, $\alpha[i]$ is obtained from α by eliminating all the actions which are not in $acts(A_i)$, eliminating states which are not preceded by actions, and replacing each remaining state with its i^{th} component.

3 Communication Systems

Consider a distributed network of n processes, $P_1, \ldots, P_n$ that communicate by means of k message buffers $B_1, \ldots, B_k$. Each process P_i is an I/O automaton and each message buffer B_j is a specification. Let $acts(B)$ denote the union $\cup_{j=1}^{k} acts(B_j)$. In the next section we define (specification of) message buffers precisely. For the purpose of this section, message buffers are specifications that share actions with the processes. The actions of the processes, however, are assumed to be mutually disjoint.

Let P denote the composition $P_1 \circ \ldots \circ P_n$. From the definition of fair executions it follows that in every fair execution of P, each process has infinitely many actions. Since the processes communicate through the message buffers, we only want to consider executions of P that obey the buffers' specification. Let $\mathcal{P}_B$ denote this set, i.e.,

$$\mathcal{P}_B = \{\eta : \eta \text{ is a fair execution of } P \text{ and } (\eta | acts(B_j)) \in beh(B_j) \text{ for every } j = 1, \ldots, k\}.$$

Fix some admissible logic L. For any model $\mathcal{M} = (\Pi, S, I)$, we say that M is *acceptable* if for every $s \in S$, if $s \in acts(B)$ then $I(s) = \{s\}$, otherwise $I(s) \cap acts(B) = \emptyset$.

We now interpret L-formulae over the executions in $\mathcal{P}_B$. With each process P_i we associate an acceptable model

$$\mathcal{M}_i = (\Pi_i \subseteq \Phi, S_i = acts(P_i) \cup states(P_i), I_i)$$

such that each for every $s \in states(P_i)$, $I_i(s) \subseteq \Phi_s$ and for every $a \in acts(P_i)$, $I_i(a) \subseteq \Phi_a$. Given an L-formula φ, we say that φ *is i-valid* if for every fair execution η of P_i, $(\mathcal{M}_i, \eta) \models_L \varphi$. We assume that the Π_is are mutually disjoint. Let $\mathcal{M} = (\Pi, S, I)$ be the model where:

1. $\Pi = \cup_{i=1}^{n} \Pi_i$,
2. $S = states(P) \cup acts(P)$,
3. for every $(s_1, \ldots, s_n) \in states(P)$, $I(s) = \cup_{i=1}^{n} I_i(s_i)$.
4. for every $a \in acts(P)$, if $a \in acts(P_i)$, then $I(a) = I_i(a)$. (Recall that each action is "owned" by a unique P_i.)

[2]This definition of fairness diverges from the usual one which allows for finite fair executions

Obvioulsy, $\mathcal{M}$ is an acceptable model. Also, each $s \in \mathit{states}(P)$ is a regular state and each $a \in \mathit{acts}(P)$ is an action state. We say that an L-formula φ is *$\mathcal{P}_B$-valid* if $(\mathcal{M}, \eta) \models_L \varphi$ for every execution η of $\mathcal{P}_B$.

We next define the theory of B—the set of L-formulae that are valid over all executions of systems of processes that communicate through $B_1, \ldots, B_k$. The theory of B, denoted by $\mathcal{T}(B)$, is the set of all formulae φ such that for all acceptable models $\mathcal{M}' = (\Pi', S', I')$, where $\Pi' \supseteq (\mathit{prop}(\varphi) \cup \mathit{acts}(B))$ and $(S' \cap \mathit{acts}(B)) \neq \emptyset$, and for all $\sigma \in (S')^\omega$ such that $(\sigma|\mathit{acts}(B_j)) \in \mathit{beh}(B_j)$ for $j = 1, ..., k$, $(\mathcal{M}', \sigma) \models_L \varphi$. Obviously, φ is *$\mathcal{P}_B$-valid* for every $\varphi \in \mathcal{T}(B)$.

We now formally state and prove our main theorem. Roughly speaking, the theorem states that if every process satisfies some local L-specification, and if the formula "χ: the conjunction of the local L-specifications implies ψ" is in the theory of B, then the whole system satisfies ψ. Since we want to guarantee that the "local" specifications above are indeed local, we replace their occurrence in χ by their projected form (see Section 2).

Theorem 3.1 *Given L-formulae $\psi, \varphi^1, \ldots \varphi^n$ over $\Pi, \Pi_1, \ldots, \Pi_n$, respectively, which are invariant under stuttering, such that:*

1. *φ^i is i-valid for every $i = 1, \ldots, n$, and*
2. *the formula $(\bigwedge_{i=1}^n \varphi^i_{(\Pi_i)} \rightarrow \psi)$ is in $\mathcal{T}(B)$.*

Then ψ is $\mathcal{P}_B$-valid.

Proof Let $\eta = \eta_0, \ldots$ be an execution in $\mathcal{P}_B$. Our goal is to show that $(\mathcal{M}, \eta) \models_L \psi$. We first show that $(\mathcal{M}, \eta) \models_L \varphi^i_{(\Pi_i)}$ for ever $i = 1, \ldots, n$, and then show how this implies that $(\mathcal{M}, \eta) \models_L \psi$.

Assume $1 \leq i \leq n$. Let $\eta' = \eta'_0, \ldots$ be the sequence obtained from η by deleting every action which is not in $\mathit{acts}(P_i)$, and let $\eta'' = \eta''_0, \ldots$ be the sequence obtained from η' by replacing every state by its i^{th} component. Since φ^i is i-valid, $(\mathcal{M}_i, \eta[i]) \models_L \varphi^i$. Since φ^i is invariant under stuttering and since η'' is a stuttered extension of $\eta[i]$, we have

$$(\mathcal{M}_i, \eta'') \models_L \varphi^i.$$

¿From the definition of $\mathcal{M}_i$s and $\mathcal{M}$, it follows that $I(\eta'_j) \cap \mathrm{prop}(\varphi^i) = I_i(\eta''_j) \cap \mathrm{prop}(\varphi)$ for every $j \geq 0$. Hence, it now follows from the assumption on $\models_L$ that

$$(\mathcal{M}, \eta') \models_L \varphi^i.$$

The sequence η' is exactly the sequence $\eta|\Pi_i$. Since η is an execution of P, $\eta_0 \in \mathit{states}(P)$. From the definition of I it now follows η_0 is a Π_i state. Consequently, η'_0 is also a Π_i-state. Moreover, since η is a fair execution of P, it has infinitely many Π_i states, hence η' is infinite. It now follows from the projection property that

$$(\mathcal{M}, \eta) \models_L \varphi^i_{(\Pi_i)}.$$

The above argument shows that $(\mathcal{M}, \eta) \models_L \varphi^i_{(\Pi_i)}$ for every $i = 1, \ldots, n$. Consequently,

$$(\mathcal{M}, \eta) \models_L \bigwedge_{i=1}^{n} \varphi^i_{(\Pi_i)}.$$

Since $(\bigwedge_{i=1}^n \varphi^i_{(\Pi_i)} \rightarrow \psi) \in \mathcal{T}(B)$, it follows that

$$(\mathcal{M}, \eta) \models_L (\bigwedge_{i=1}^{n} \varphi^i_{(\Pi_i)} \rightarrow \psi).$$

We can therefore conclude that

$$(\mathcal{M}, \eta) \models_L \psi.$$

■

For each $i = 1, \ldots, n$, we say that a formula φ^i characterizes the process P_i with respect to the model $\mathcal{M}_i$ if φ^i is invariant under stuttering and for any $\sigma \in S_i^\omega$, $(\mathcal{M}_i, \sigma) \models_L \varphi^i$ iff either σ is an execution of P_i or it is a stuttered extension of such an execution. Now the following theorem states that if the logic L is expressive enough to characterize the processes $P_1, \ldots, P_n$ then any L-formula that is $\mathcal{P}_B$-valid can be proven so by using the method given in theorem 3.1.

Theorem 3.2 *If $\psi, \varphi^1, \ldots \varphi^n$ are L-formulae such that:*

1. *$\psi, \varphi^1, \ldots \varphi^n$ are invariant under stuttering,*
2. *φ^i characterizes P_i with respect to $\mathcal{M}_i$ for every i, and*
3. *ψ is $\mathcal{P}_B$-valid*

then $(\bigwedge_{i=1}^n \varphi^i_{(\Pi_i)} \rightarrow \psi)$ is in $\mathcal{T}(B)$.

4 Temporal Logics and Message Buffers

In Section 2 we introduced *admissible logics*. In Section 3 we defined communication systems and showed how to verify some properties of admissible logics in them. In this section we show that two variants of temporal logic, termed here PTL and RTL, are admissible. PTL is the usual propositional temporal logic with past operators. RTL is PTL without the Until, Since and Nexttime operators, i.e., a temporal logic that uses only the (past and future) eventuality operators.

4.1 PTL and RTL

Propositional Temporal Logic (PTL) is one of the logics defined in Section 2. In addition to the boolean connectives, PTL-formulae can include the temporal operators U (*until*), S (*since*) and $\bigcirc$ (*nexttime*) operators.

Before defining the satisfiability relation $\models_{PTL}$ (denoted by $\models_P$), we define an auxiliary satisfiability relation, $\models'$, between a model $\mathcal{M} = (\Pi, S, I)$, a computation $\sigma = s_0, s_1, \ldots$ over S, and a PTL-formula φ. The relation $\models'$ is defined inductively as follows:

$(\mathcal{M}, \sigma, j) \models'$ true and $(\mathcal{M}, \sigma, j) \not\models'$ false for every $\sigma \in S^\omega$ and $j \geq 0$.

For a state assertion $Q \in \Pi$, $(\mathcal{M}, \sigma, j) \models' Q$ *iff* $Q \in I(s_j)$.

$(\mathcal{M}, \sigma, j) \models' \neg\varphi$ *iff* $(\mathcal{M}, \sigma, j) \not\models' \varphi$.

$(\mathcal{M}, \sigma, j) \models' \varphi_1 \vee \varphi_2$ *iff* $(\mathcal{M}, \sigma, j) \models' \varphi_1$ or $(\mathcal{M}, \sigma, j) \models' \varphi_2$.

$(\mathcal{M}, \sigma, j) \models' \varphi_1 \mathsf{U} \varphi_2$ *iff* for some $j' \geq j$, $(\mathcal{M}, \sigma, j') \models' \varphi_2$ and for all i, $j \leq i < j'$, $(\mathcal{M}, \sigma, i) \models' \varphi_1$.

$(\mathcal{M}, \sigma, j) \models' \varphi_1 \mathsf{S} \varphi_2$ *iff* for some j' such that $0 \leq j' \leq j$, $(\mathcal{M}, \sigma, j') \models' \varphi_2$ and for all i, $j' < i \leq j$, $(\mathcal{M}, \sigma, i) \models' \varphi_1$.

$(\mathcal{M}, \sigma, j) \models' \bigcirc\varphi$ *iff* $(\mathcal{M}, \sigma, j+1) \models' \varphi$.

Additional boolean connectives (such as $\wedge$, $\rightarrow$, $\leftrightarrow$) can be defined in the usual way. We define additional temporal operators, $\diamondsuit$ (*eventually in the future*), its dual $\boxdot$ (*always in the future*), and their past counterparts, $\Diamond$ (*eventually in the past*) and $\boxminus$ (*always in the past*) by:

$$\diamondsuit\varphi \leftrightarrow \mathsf{true}\mathsf{U}\varphi \quad \Diamond\varphi \leftrightarrow \mathsf{true}\mathsf{S}\varphi$$
$$\boxdot\varphi \leftrightarrow \neg\diamondsuit\neg\varphi \quad \boxminus\varphi \leftrightarrow \neg\Diamond\neg\varphi$$

The satisfiability relation $\models_P$ is defined by:

$$(\mathcal{M}, \sigma) \models_P \varphi \qquad \text{iff} \qquad (\mathcal{M}, \sigma, 0) \models' \varphi.$$

RTL (Restricted Temporal Logic) formulae are PTL formula that only do not use the U, S and $\bigcirc$ operators, that is, they only use the $\diamondsuit$, $\Diamond$, and their duals as temporal operators. We denote $\models_{RTL}$ by $\models_R$.

4.2 Admissibility of PTL and RTL

We now show that both PTL and RTL are admissible logics. We first note that both $\models_P$ and $\models_R$ trivially satisfy all the five requirements of a satisfiability relation of admissible logics listed in Section 2. We next show that both logics have the projection property.

Theorem 4.1 *PTL has the projection property.*

Proof We prove the claim by defining, for every PTL-formula φ and set $\Pi \subseteq \Phi$, a projected formula $\varphi_{(\Pi)}$ and showing that the defined $\varphi_{(\Pi)}$ satisfies the requirements of a projected formula.

For every finite set $\Pi \subseteq \Pi$, let $\overline{\Pi}$ denote the formula $\bigvee_{\pi \in \Pi} \pi$. Given a PTL-formula φ, we define $\varphi_{(\Pi)}$ by induction on the structure of φ as follows:

- If $\varphi \in \Phi$ or $\varphi \in \{\mathsf{true}, \mathsf{false}\}$, , then $\varphi_{(\Pi)} = \varphi$.
- If $\varphi = \neg\varphi'$, then $\varphi_{(\Pi)} = \neg\varphi'_{(\Pi)}$.
- If $\varphi = \varphi' \vee \varphi''$, then $\varphi_{(\Pi)} = \varphi'_{(\Pi)} \vee \varphi''_{(\Pi)}$.
- If $\varphi = \varphi' \mathsf{U} \varphi''$, then $\varphi_{(\Pi)} = (\overline{\Pi} \rightarrow \varphi'_{(\Pi)}) \mathsf{U} (\overline{\Pi} \wedge \varphi''_{(\Pi)})$.
- If $\varphi = \varphi' \mathsf{S} \varphi''$, then $\varphi_{(\Pi)} = (\overline{\Pi} \rightarrow \varphi'_{(\Pi)}) \mathsf{S} (\overline{\Pi} \wedge \varphi''_{(\Pi)})$.
- If $\varphi = \bigcirc\varphi'$, then $\varphi_{(\Pi)} = \bigcirc(\neg\overline{\Pi} \mathsf{U} (\overline{\Pi} \wedge \varphi'_{(\Pi)}))$

It remains to show that for every model $\mathcal{M} = (\Pi', S, I)$ where $\Pi' \supseteq (\mathsf{prop}(\varphi) \cup \Pi)$ and every computation $\sigma = s_0, s_1, \ldots$ such that $\sigma|\Pi$ is infinite and s_0 is a Π-state,

$$(\mathcal{M}, \sigma) \models_P \varphi_{(\Pi)} \qquad \text{iff} (\mathcal{M}, \sigma|\Pi) \models_P \varphi.$$

Let $\mathcal{M} = (\Pi', S, I)$ such a model and σ be such a computation. Let h be a mapping between indices of $\sigma|\Pi$ to indices of σ states such that $\sigma|\Pi = s_{h(0)}, s_{h(1)}, \ldots$. We claim that for every $i \geq 0$,

$$(\mathcal{M}, \sigma, h(i)) \models' \varphi_{(\Pi)} \qquad \text{iff} (\mathcal{M}, \sigma|\Pi, i) \models' \varphi.$$

The proof of the claim is by induction on the structure of φ and is left to the reader. Since s_0 is a Π-state, $h(0) = 0$, we therefore conclude that

$$(\mathcal{M}, \sigma) \models_P \varphi_{(\Pi)} \qquad \text{iff} (\mathcal{M}, \sigma|\Pi) \models_P \varphi.$$

■

Theorem 4.2 *RTL has the projection property.*

Proof The proof is similar to that of Theorem 4.1. The only difference is the definition of the projected formula: Since RTL formula do not have U and S operators, neither should projected formulae have them. We therefore add to the definitions given in the proof of Theorem 4.2 the following:

- If $\varphi = \diamondsuit\!\!\!\!\cdot\,\varphi'$ then $\varphi_{(\Pi)} = \diamondsuit\!\!\!\!\cdot\,(\overline{\Pi} \wedge \varphi'_{(\Pi)})$.
- If $\varphi = \diamondsuit\varphi'$ then $\varphi_{(\Pi)} = \diamondsuit(\overline{\Pi} \wedge \varphi'_{(\Pi)})$.

The proof that $\varphi_{(\Pi)}$ as defined above satisfies the requirements from a projected formula follows immediately from the proof of Theorem 4.1. ■

We can therefore conclude:

Theorem 4.3 *Both PTL and RTL are admissible logics.*

5 Message Buffers

In Section 3 we considered message buffers to be arbitrary specifications. In this section we study four examples of unbounded message buffers with finite message alphabets: first-in-first-out (fifo) buffers, fifo buffers with deletions, unordered buffers without liveness, and unordered buffers with liveness. As we show, the theory of the first two message buffers in RTL is decidable (in fact, in co-NP), and the theory of the last two message buffers in PTL is decidable. Hence, Theorem 3.1 can be used to prove properties of systems that communicate through these buffers. We should note that the theory of fifo message buffers in the full temporal logic is Π_1^1-complete and thus not axiomatizable ([SCFM84]).

5.1 Specifications of Message Buffers

A message buffer is characterized by the sequences of read/write actions it admits. These sequences depend on the properties of the buffer (e.g., fifo) and on the message alphabet. We denote a message buffer of type x with message alphabet M by $\mathsf{x}(M)$. For every $\mathsf{x}(M)$, we have

$$inp(\mathsf{x}(M)) = \{\mathsf{write}(m) : m \in M\} \quad \text{and} \quad out(\mathsf{x}(M)) = \{\mathsf{read}(m) : m \in M\}.$$

Since all the buffers of type x over M have the same input and output actions, we omit 'x' when discussing these actions.

Recall the system presented in Section 3. There we assume k message buffers of the same type whose actions are mutually disjoint, hence their message alphabets are mutually disjoint. Given such k disjoint message alphabets $M_1, \ldots, M_k$, we denote by $\mathsf{x}(\hat{M})$ the joint system of the k buffers of type x that communicate over these alphabets. We also assume that:

- $inp(\hat{M}) = \bigcup_{i=1}^{k} inp(M_i))$,
- $out(\hat{M}) = \bigcup_{i=1}^{k} out(M_i)$, and
- $beh(\mathsf{x}(\hat{M})) = \{\alpha \in (acts(\hat{M}))^{\infty} : \alpha | acts(\mathsf{x}(M_i))$ is in $beh(\mathsf{x}(M_i))$ for every $i = 1, \ldots, k\}$.

For every buffer type x, (mutually disjoint) message alphabets $M_1, \ldots, M_k$, and admissible logic L, let $\mathcal{T}_L(\mathsf{x}(\hat{M}))$ denote the theory of $\mathsf{x}(\hat{M})$ in L. That is, the set of L-formulae φ such that for every acceptable model $\mathcal{M}' = (\Pi', S', I')$ $(\Pi' \supseteq (\mathsf{prop}(\varphi) \cup acts(\hat{M})))$, for every computation σ over S' such that $(\sigma | acts(\hat{M}))$ is in $beh(\mathsf{x}(M))$, $(\mathcal{M}', \sigma) \models_L \varphi$.

We abbreviate $\mathcal{T}_{PTL}(\mathsf{x}(\hat{M}))$ to $\mathcal{T}_P(\mathsf{x}(\hat{M}))$ and $\mathcal{T}_{RTL}(\mathsf{x}(\hat{M}))$ to $\mathcal{T}_R(\mathsf{x}(\hat{M}))$

We now describe the four buffer types and some results about their theories.

Fifo Buffers

Fifo buffers are message buffers where every read operation returns the value of the oldest unread message. Moreover, in every sequence of observable behaviors, if there are infinitely many write actions in the sequence then there are also infinitely many read actions in the sequence. Formally, for every message alphabet M, $beh(\mathsf{fifo}(M))$ is the set of all sequences α over $acts(M)$ such that the following holds:

1. for every finite prefix α' of α, $\alpha' | out(M)$, is a prefix of $\alpha' | inp(M)$, and
2. if the $\alpha | inp(M)$ is infinite, then $\alpha | out(M)$ is infinite.

In the full version of the paper we prove the following theorem, which is an extension of a similar theorem of [SZ90].

Theorem 5.1 $\mathcal{T}_R(\mathsf{fifo}(\hat{M}))$ *is in co-NP, i.e., there exists a procedure in NP that decides whether a given RTL formula* φ *is not in* $\mathcal{T}_R(\mathsf{fifo}(\hat{M}))$.

Fifo Buffers with Deletions

Fifo buffers with deletions (fifod buffers) are fifo buffers that can delete messages. From such buffers we require that every message that is written infinitely many times is read infinitely many times. Formally, for every message alphabet M, $beh(\mathsf{fifod}(M))$ is the set of all sequences α over $acts(M)$ such that the following holds:

1. for every finite prefix α' of α, $\alpha' | out(M)$, is a subsequence of $\alpha' | inp(M)$, and
2. for every $m \in M$, if the $\alpha | \mathsf{write}(m)$ is infinite, then $\alpha | \mathsf{read}(m)$ is infinite.

In the full version of the paper we prove the following theorem, which is an extension of a similar theorem of [SZ90].

Theorem 5.2 $\mathcal{T}_R(\mathsf{fifod}(\hat{M}))$ *is in co-NP, i.e., there exists a procedure in NP that decides whether a given RTL formula* φ *is not in* $\mathcal{T}_R(\mathsf{fifod}(\hat{M}))$.

Unordered Buffers

Unordered buffers (unor) are buffers that can deliver messages in any order. As before, let M denote a message alphabet. Formally, for every message alphabet M, $beh(\text{unor}(M))$ includes all the sequences α over $acts(M)$ such that there exists a one-to-one mapping, h_α, that maps each read(m) event in α to a write(m) event in α that precedes it.

The following theorem is proven in [SCFM84]:

Theorem 5.3 *$\mathcal{T}_P(\text{unor}(\hat{M}))$ is elementary decidable.*

Unordered Buffers with Liveness

Unordered buffers with livenness (unorl) are unor buffers cannot read finitely many messages if infinitely many messages were written. Formally, for every message alphabet M, $beh(\text{unorl}(M))$ includes all the sequences α in $beh(\text{unor}(M))$ such that if there are infinitely many write events in α, then there are infinitely many read events in α.

The following theorem can be proved using the techniques as those given in [SCFM84].

Theorem 5.4 *$\mathcal{T}_P(\text{unorl}(\hat{M}))$ is decidable.*

5.2 A Note about Finite Systems

Let A be an I/O automaton such both $states(A)$ adn $acts(A)$ are finite. Following the method preseted in Section 3, we can associate with A a model $\mathcal{M} = (\Pi, S, I)$ such that every $s \in states(A)$ (resp., $a \in acts(A)$) is mapped to a distinct unique state assertion in Π. It is now possible to give a PTL-formula which characterizes the A's behavior with respect to $\mathcal{M}$. Hence PTL is expressive for the process P with respect to the model $\mathcal{M}$.

Using Theorems 3.2, 5.4, and 5.3, it is easily seen that the method presented in Theorem 3.1 gives us complete and fully automatic proof methods for proving temporal properties of systems of finite state concurrent processes that communicate through unordered buffers with and without liveness properties.

6 Alternating Bit Protocol

Assume a two process system, where one process, the *sender* tries to reliably communicate a sequence of data items X over a domain D of data items, to another process, the *receiver*. The receiver has to write the data items into Y. The sender and the receiver communicate via fifod message buffers.

The Alternating Bit Protocol (ABP), introduced in [BSW69], offers a solution to the problem, namely, protocols for both sender and receiver that guarantees that at any given time, Y is a prefix of X, and that eventually $Y = X$. Presented in the terminology introduced in the previous sections, ABP uses two message buffers, one from sender to receiver with message alphabet $M_{sr} = D \times \{0,1\}$, and the other from receiver to sender with a message alphabet $M_{rs} = \{0,1\}$. Let A_s denote the sender's automaton, and let A_r denote the receiver's automaton. ABP proceeds as follows:

If $= x_0, \ldots$, then, for every A_s writes $(x_i, \bar{i})$ messages onto its outgoing buffer ($\bar{i}$ denotes the parity of i), until it reads a $\neg\bar{i}$ message, at which point it knows that A_r had set $y_i := x_i$ and is awaiting x_{i+1}. A_s then increments i and proceeds to the next data item. A_r is similar.

Formally, A_s's states include the input sequence $x_0, \ldots$ and an index i, initially 0, denoting the index of the data item currently transmitted. Similarly, A_r's states include the output sequence $y_0, \ldots$ and an index j, initially 0, denoting the index of the data item that A_r is waiting for.

The code for the sender and receiver appears in Figure 1.

Although we cannot express (and, consequently, prove) all the properties of the Alternating Bit protocol in RTL, we can use the fifod version of Theorem 3.1 to show some of its properties. Recall that the evlauation of any buffer action is the action itself. We use the following abbreviations for every $d \in D$:

$$\begin{array}{llll}
inp_s = \bigvee_{a \in inp(A_s)} a & & inp_r = \bigvee_{a \in inp(A_r)} a \\
out_s = \bigvee_{a \in out(A_s)} a & & out_r = \bigvee_{a \in out(A_r)} a \\
acts_s = inp_s \vee out_s & & acts_r = inp_r \vee out_r \\
\text{past0}_s(d) = \boxminus(out_s \rightarrow \text{write}(d,0)) & & \text{past0}_r(d) = \boxminus(inp_r \rightarrow \text{read}(d,0)) \\
\text{first}_s(d) = \diamondminus(\text{write}(d,0) \wedge \text{past0}_s(d)) & & \text{first}_s(d) = \diamondminus(\text{read}(d,0) \wedge \text{past0}_r(d))
\end{array}$$

A_s	A_r
write(x,b) $((x,b) \in M_{sr})$: **precondition:** $x = x_i$ and $b = \bar{i}$	write(b) $(b \in M_{rs})$: **precondition:** $b = \bar{j}$
read(b) $(b \in M_{rs})$: **effect:** if $b \neq \bar{i}$ then $i := i + 1$	read(y,b) $((y,b) \in M_{sr})$: **effect:** if $b = \bar{j}$ then $y_j := y;\ j := j + 1$

Figure 1: The Alternating Bit Protocol

The formula

$$\psi : \text{first}_s(x) \rightarrow \text{first}_r(x)$$

therefore asserts that the first message read by the received is same as the first message written by the sender. Using Theorem 3.1, we can prove ψ by setting φ^s to:

$$\boxminus\Big((\text{write}(x,0) \wedge \text{past0}_s(x)) \rightarrow \Big((\boxminus(acts_s \rightarrow \text{write}(x,0)) \wedge \boxminus\diamondminus acts_s)) \vee \diamondminus(\text{read}(1) \wedge \text{post0}_s(x))\Big)\Big)$$

and φ^r to

$$\boxminus(\text{write}(1) \rightarrow \Diamond(\bigvee_{d \in D} \text{read}(d,0))$$

If Π_s and Π_r denote all the propositions of the sender and the receiver, it can be shown that φ^s and φ^r can be replaced by $(\varphi^s_{(\Pi_s)}$ and $\varphi^r_{(\Pi_r)})$ respectively. It can also be shown that $(\varphi^s \wedge \varphi^r) \rightarrow \psi$ is in the theory of the system of two buffers with deletions connecting the sender and receiver in the two directions.

7 Conclusions

In this paper we have presented a formal model for processes that communicate through fifo message buffers and have given a sound automatic proof system for verifying RTL definable properties of such systems. The proof method is modular. Although our method is not complete, we feel, as illustrated by the example, that it can be applied to some practical examples. Theorem 3.1 holds for any fragment L of temporal logic as long as the formulae in L do not distinguish between two computations one of which is a stuttered extension of the other. In this case, we can use our approach for proving properties given by formulae in L as long as the theory of fifo buffers in the logic L is decidable.

References

[AAF+] Y. Afek, H. Attyia, A. Fekete, M. Fischer, N. Lynch, Y. Mansour, D.-W. Wang, and L. Zuck. Reliable communication using unreliable channel. to appear in JACM.

[BSW69] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communcation of the ACM*, 12:260–261, 1969.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *Transactions on Programming Languages and Systems*, 8(2), 1986.

[CM86] K. M. Chandy and J. Misra. *Parallel Program Design: A Fundation.* (A draft), 1986.

[EL85] E. A. Emerson and C. L. Lei. Modalities for model checking: Branching time strikes back. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 84–96, 1985.

[Lam83] L. Lamport. Specifying concurrent program modules. *ACM TOPLAS*, 5(2):190–222, 1983.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite-state concurrent programs satisfy their linear specifications. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 97–107, 1985.

[LT87] N.A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 137–151, 1987.

[LT89] N.A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989.

[OL82] S. Owicki and L. Lamport. Proving liveness properties of concurrent programs. *TOPLAS*, 4(3):455–495, 1982.

[Pnu77] A. Pnueli. The Temporal Logic of programs. In *Proc. 18th IEEE Symp. on Foundation of Computer Science*, pages 46–57, 1977.

[SCFM84] A. P. Sistla, E. M. Clarke, N. Francez, and A. R. Meyer. Can message buffers be axiomatized in linear temporal logic? *Information and Control*, 63(1/2):88–112, 1984.

[SZ90] A. P. Sistla and L. D. Zuck. Reasoning in a restricted temporal logic. submitted for publication, parts appeared in [SZ87], 1990.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proc. 1st IEEE Symp. on Logics in Computer Science*, 1986.