Undecidability of Bisimulations in Concurrent Systems with Indefinite Number of Identical Processes

Mahesh Girkar *Robert Moll*
Department of Computer Science
University of Massachussetts
Amherst MA 01003

CMPSCI Technical Report 93-86
December 1993

Abstract

In this paper, we show that bisimulation equivalence between concurrent systems of the type described in [9] is undecidable for two logics — Linear Temporal Logic LTL and LTL without the nexttime operator. Such concurrent systems contain a control process and arbitrarily many identical user processes. These negative results indicate that it might be difficult to compare two such concurrent systems and obtain the advantages from equivalences that are possible in other simpler models of concurrency [3, 6, 1, 7].

1 Introduction

Most formal models for concurrent systems (referred to as CS subsequently) provide a means for describing, and ultimately, detecting such phenomenon as deadlock, starvation and liveness in such systems. In a resource-oriented model [2, 11], processes are entities executing operations which request and release resources. These resources are controlled by finite state devices called synchronizers. In this model, properties such as deadlock and liveness can be expressed naturally using temporal logic. In general, a proof that establishes such a property depends on the model having a fixed number of processes. When the number of processes increases, the task of proving that the new instance of the model possesses desirable properties must be performed all over again. In [9], a formal setting is described and several properties analyzed for systems with a synchronizer (referred to as a control process in [9]) and an indefinite number of identical user processes. A Milner style communication mechanism is used for processes to interact with each each other. In [5], two forms of such systems differing in their power to model user processes and in their power to interact are analyzed for detecting *strong stability* — an important property which asserts whether there are infinitely many model sizes where deadlock occurs.

Previously, research has been conducted in studying various notions of equivalences for simpler models of concurrency, such as Kripke structures [3]. Among other things, such research has been shown to be useful in reducing the size of Kripke structures [3] and in replacing a part of a hierarchical system by another equivalent part [6, 1, 7].

In this paper, we study bisimulation equivalences between CS of the type described in [9] with respect to two logics — Propositional Linear Temporal Logic (LTL), and LTL without the nexttime operator ($LTL/\mathbf{X}$). A CS of the type described in [9] satisfies a temporal formula *iff* there is a concurrent execution of the system which is a "witness" for this formula. Two CS are bisimilar *iff* they are witness to the same set of formulaes. Our main result is that bisimulation equivalence is undecidable for these CS with respect to LTL and $LTL/\mathbf{X}$. These results indicate that it might be difficult to compare two such CS and obtain the advantages from equivalences that are possible in other simpler models of concurrency [3, 6, 1, 7].

Using an undecidability result about Petri nets, we show in Section 2 that the equality problem for the language of executions of two arbitrary CS is undecidable. In Section 3 we show that the bisimilarity problem with respect to LTL and $LTL/\mathbf{X}$ is undecidable. In that section we also show that certain restricted variations of the bisimilarity problem are also undecidable. Appendix A contains a formal description of a CS, execution sequences of a CS, and the languages associated with these sequences. Appendix B contains proofs for all the propositions and theorems which are not obvious.

2 Undecidability of the Equality Problem for Language of Executions

The notation for describing LTL and our model for CS is from [9] and is summarized in Appendix A. $LTL/\mathbf{X}$ is LTL with the nexttime operator, $\mathbf{X}$, discarded. In ascertaining certain properties of a CS, the use of the nexttime operator can be dangerous, since it refers to a "global" next state as opposed to the local "next" state [4, 3]. $LTL/\mathbf{X}$ is thus a useful subset of LTL and can be used to ascertain properties of a CS in which the power to "count" steps is not deemed important.

Our model for concurrency consists of a unique *synchronizer* process and many identical *user* processes. Such a model, at a certain level of abstraction, is useful for describing concurrency problems in which user processes are entities executing operations which request and release resources controlled by the synchronizer [2, 11, 5]. Properties of interest, including mutual exclusion, reachability of a certain state in a synchronizer, etc., can be specified with temporal formulas.

In brief, processes are finite state machines that communicate with each other via an alphabet that contains pairs of complementary symbols, and a special symbol, ϵ. Two processes (either two user processes or a user process and the synchronizer) in states s and s' can communicate with each other provided there are two edges ϵ and ϵ' directed out of s and s' respectively, which are labeled with complementary symbols. In addition, any process at a state can undergo a transition provided there is an edge, labeled ϵ, directed out of that state.

The states of the synchronizer and user processes are annotated with symbols from a set of atomic propositions $\mathcal{AP}$. The finite state descriptions of both the synchronizer and user processes also contain a designated set of initial states. We denote by $(P^k, \mathcal{S})$ a CS with k user processes, each of whose finite state description is P, and a synchronizer whose finite state

description is $\mathcal{S}$. $(P, \mathcal{S})$ is a CS with an indefinite number of user processes and a synchronizer. Assuming all processes in the CS are in an initial state, an execution sequence is a sequence of global state configurations, each of which is caused by communication between two processes or by an ϵ transition. We consider three types of executions: (E_1) finite executions that arise in $(P^k, \mathcal{S})$ for any $k > 0$; (E_2) finite as well as infinite executions that arise from a system $(P^k, \mathcal{S})$ for any $k > 0$; and, lastly, (E_3) finite as well as infinite executions that arise in a system $(P, \mathcal{S})$.

By projecting execution sequences on to states of the synchronizer, and by using the annotations on the states of the synchronizer, we obtain sequences of subsets of atomic propositions that appear for a synchronizer in executions of CS. Such sequences are "witnesses" for the validity of a subset of temporal formulas constructed using the set $\mathcal{AP}$. If v is a finite or infinite sequence of subsets of atomic propositions and f is a temporal formula, then we write $v \models f$ iff v is a witness for f. For a formal definition of validity see Appendix A.

The set of sequences of subsets of atomic propositions that appear for type (E_1), (E_2), and (E_3) executions are denoted respectively by $L_{\mathcal{S}}^{finite}(P, \mathcal{S})$, $L_{\mathcal{S}}^{proper}(P, \mathcal{S})$, and $L_{\mathcal{S}}^{extended}(P, \mathcal{S})$. We obtain similar sequences by considering any user process and projecting execution sequences on to states of this process. Since all user processes are identical, the three sets obtained in this fashion will be independent of the user process selected for projection. For any user process, we denote these sets by $L_{\mathcal{P}}^{finite}(P, \mathcal{S})$, $L_{\mathcal{P}}^{proper}(P, \mathcal{S})$, and $L_{\mathcal{P}}^{extended}(P, \mathcal{S})$. In [9] a simple transformation from a CS $(P, \mathcal{S})$ to another CS $(P', \mathcal{S}')$ is provided such that $L_{\mathcal{P}}^{\epsilon}(P, \mathcal{S})$ equals $L_{\mathcal{S}'}^{\epsilon}(P', \mathcal{S}')$, where ϵ is one of *finite*, *proper*, or *extended*. Thus, it suffices to restrict our attention to the sets $L_{\mathcal{S}}^{\epsilon}(P, \mathcal{S})$, where ϵ is one of *finite*, *proper*, or *extended*.

Definition 1 *A temporal formula $f \in LTL$ holds in a system $(P, \mathcal{S})$ with respect to finite executions* iff $\exists v \in L_{\mathcal{S}}^{finite}(P, \mathcal{S})$ *such that $v \models f$. In this case, we write $(P, \mathcal{S}) \models_{LTL}^{finite} f$. Similarly, we define $(P, \mathcal{S}) \models_{LTL/\boldsymbol{X}}^{finite} f$, $(P, \mathcal{S}) \models_{LTL}^{proper} f$, $(P, \mathcal{S}) \models_{LTL/\boldsymbol{X}}^{proper} f$, $(P, \mathcal{S}) \models_{LTL}^{extended} f$, and $(P, \mathcal{S}) \models_{LTL/\boldsymbol{X}}^{extended} f$.*

The *model checking problem* for $(P, \mathcal{S})$ with respect to a set of executions E, where E is one of $L_{\mathcal{S}}^{finite}(P, \mathcal{S})$, $L_{\mathcal{S}}^{proper}(P, \mathcal{S})$, or, $L_{\mathcal{S}}^{extended}(P, \mathcal{S})$, and a formula f, is the problem of deciding whether there exists an element v in E such that $v \models f$.

The following proposition is proved in [9][1].

Proposition 1 *The model checking problem is decidable for the execution classes defined above and for all LTL formulas.*

Definition 2 *Two systems $(P, \mathcal{S})$ and $(P', \mathcal{S}')$ are* consistent *if their states are labeled with propositional symbols from the same set $\mathcal{AP}$. Two consistent systems $(P, \mathcal{S})$ and $(P', \mathcal{S}')$ are* bisimilar *with respect to LTL and $L_{\mathcal{S}}^{finite}(P, \mathcal{S})$ if $\forall f \in LTL \quad (P, \mathcal{S}) \models_{LTL}^{finite} f$ iff $(P', \mathcal{S}') \models_{LTL}^{finite} f$. We write $(P, \mathcal{S}) \equiv_{LTL}^{finite} (P', \mathcal{S}')$ when $(P, \mathcal{S})$ and $(P', \mathcal{S}')$ are* bisimilar *with respect to LTL and $L_{\mathcal{S}}^{finite}(P, \mathcal{S})$. Similarly, we define $(P, \mathcal{S}) \equiv_F^{\epsilon} (P', \mathcal{S}')$ when F is one of LTL or $LTL/\boldsymbol{X}$ and "ϵ" is one of "finite", "extended", or "infinite".*

In this paper we show that the following problems are undecidable:

Problem 1: Given two arbitrary consistent concurrent systems $(P, \mathcal{S})$ and $(P', \mathcal{S}')$, is $L_{\mathcal{S}}^{\epsilon}(P, \mathcal{S}) = L_{\mathcal{S}}^{\epsilon}(P', \mathcal{S}')$, where ϵ stands for one of *finite*, *proper*, or *extended*.

Problem 2: Given two arbitrary consistent concurrent systems $(P, \mathcal{S})$ and $(P', \mathcal{S}')$, is $(P, \mathcal{S}) \equiv_F^{\epsilon} (P', \mathcal{S}')$, where F stands for either LTL or $LTL/\mathbf{X}$ and ϵ stands for one of *finite*, *proper*, or *extended*.

The undecidability of Problem 1 follows from several transformations starting from a well-known undecidable problem in Petri nets, and is given below. In Section 3 we prove that Problem 2 is undecidable.

2.1 An Undecidable Problem in PN Domain

The discussion in this section follows the notation and treatment in [8].

[1] The proof in [9] must be modified slightly when $E = L_{\mathcal{S}}^{extended}(P, \mathcal{S})$.

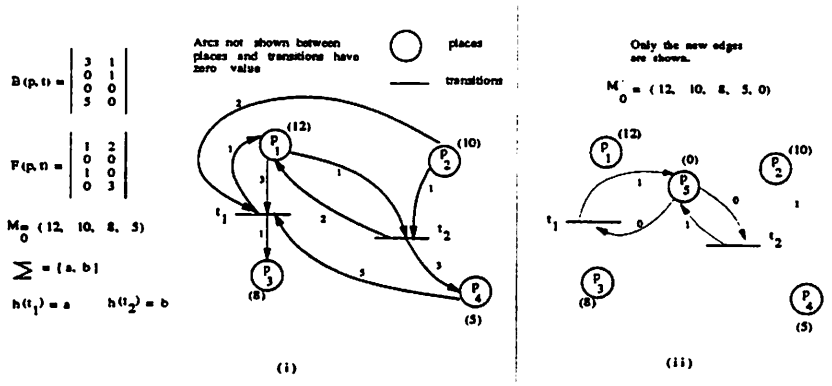


Figure 1: Petri Net. Transformation Γ_1

A *Labeled r-dimensional Petri net (referred to as PN subsequently)*, $\mathcal{N}$, is a 7-tuple $(P, T, B, F, M_0, \sum, h)$ where $P = \{p_1, p_2, \ldots, p_r\}$ is a set of *places*, $T = \{t_1, t_2, \ldots, t_s\}$ is a set of *transitions*, $B : P \times T \rightarrow \mathcal{N}$ is the *backwards incidence function*, $\mathcal{N}$ is the set of non-negative integers, and $F : P \times T \rightarrow \mathcal{N}$ is the *forwards incidence function*. A marking $M \in \mathcal{N}^r$ associates a non-negative number with each place. $M_0 \in \mathcal{N}^r$ is the *initial marking*. Let $\sum$ be a finite set of symbols and let $h : T \rightarrow \sum \bigcup \{\lambda\}$ be a labeling function, where λ is the empty symbol. If $h(t) \neq \lambda$ for all $t \in T$ the labeling function is called *λ-free* and the PN is called a λ-free PN.

For a string $w \in T^* \bigcup T^\omega$ we write $M_0(w >$ iff the sequence of transitions specified by w can be fired starting with M_0. We write $M_0(w > M$ iff $M_0(w >$ and the marking M is reached by firing the transitions specified in w. String w is *proper* if $M_0(w >$ and either (i) $w \in T^*$; or (ii) if there exists a $U > 0$, such that for every M', w', if $M(w' > M'$ and w' is a prefix of w, then $M' \leq (U, U, \ldots U)$(r times). Note that the bound U can be different for different proper strings in T^ω.

Figure 1(i) shows a PN as a graph. Each non-zero $F(p, t)$ value (respectively a non-zero $B(p, t)$ value) is represented by an edge from transition t to place p (from p to t) with value $F(p, t)$ (respectively $B(p, t)$).

Definition 3 *For a λ-free PN $\mathcal{N} = (P, T, B, F, M_0, \sum, h)$, let*

$$
\begin{aligned}
L^{finite}(\mathcal{N}) = &\quad \{h(w) | w \in T^* \text{ and } M_0(w >\}; \\
L^{proper}(\mathcal{N}) = &\quad L^{finite}(\mathcal{N}) \bigcup \{h(w) | w \in T^\omega, M_0(w >, \text{ and } w \text{ is proper}\}; \\
L^{extended}(\mathcal{N}) = &\quad L^{finite}(\mathcal{N}) \bigcup \{h(w) | w \in T^\omega \text{ and } M_0(w >\}
\end{aligned}
$$

The following result is proved in [8]:

Theorem 1 *Given two arbitrary λ-free labeled PN $\mathcal{N}$ and $\mathcal{N}'$, the equality problem for languages $L^{finite}(\mathcal{N})$ and $L^{finite}(\mathcal{N}')$ is undecidable.*

Proposition 2 *(1) Given two arbitrary λ-free labeled PN $\mathcal{N}$ and $\mathcal{N}'$*
$L^{finite}(\mathcal{N}) = L^{finite}(\mathcal{N}')$ iff $L^{extended}(\mathcal{N}) = L^{extended}(\mathcal{N}')$; *and*
(2) There exists a transformation Γ_1, which, given arbitrary PN $\mathcal{N}_1$ and $\mathcal{N}_2$, constructs nets $\mathcal{N}_1'$ and $\mathcal{N}_2'$ such that $L^{finite}(\mathcal{N}_1) = L^{finite}(\mathcal{N}_2)$ iff $L^{proper}(\mathcal{N}_1') = L^{proper}(\mathcal{N}_2')$.

Proof: The proof for (1) follows from Konig's Lemma and is given in Appendix B. For (2), consider the following transformation Γ_1 which eliminates proper infinite firing sequences in the original PN by mapping such sequences to infinite firing sequences that are not proper in the new PN. In fact, the new net has no proper infinite firing sequences. Finite firing sequences of the original net are unaltered in the new net.

The transformation $\Gamma_1 : \mathcal{N} \rightarrow \mathcal{N}'$ appends one additional place to $\mathcal{N}$. Arcs out of this place are labeled 0, while arcs into this place are labeled 1. Thus whenever a transition fires, it increases the value of this place by 1. Hence no infinite firing sequence w is proper. □

See Figure 1 for an example. Using Proposition 2 and Theorem 1 we get the following result:

Corollary 1 *The equality problem for languages $L^{\epsilon}(N)$ and $L^{\epsilon}(N')$ for two arbitrary λ-free labeled PN N and N' is undecidable, where "ϵ" is one of "finite", "proper", or "extended".*

2.2 Labeled Vector Addition System with States — The VASS Model

An r-dimensional Labeled VASS $V = (S, E, D, q_0, v_0, \sum, h)$ is a 7-tuple where $S = \{q_0, q_1, \ldots, q_m\}$ is a finite set of states, $E : S \times S$ is a set of directed edges $\epsilon_1, \epsilon_2, \ldots \epsilon_k$, $\mathcal{I}$ is the set of integers, $D : E \rightarrow \mathcal{I}^r$ associates with each edge an r-dimensional "direction" vector with integer coordinates, q_0 is the initial state of V, $v_0 \in \mathcal{N}^r$ is the *initial* vector, $\sum$ is a finite set of symbols, and $h : S \rightarrow \sum$ is a function that associates with each state a symbol from $\sum$. As before, h is a homomorphism from sequences of states to words in $\sum$.

A configuration of V is of the form (q, v) where $q \in S$ and v is an r-dimensional vector with integer coordinates. Starting with the initial configuration (q_0, v_0), a path in V induces a sequence of configurations which are obtained using E and D as follows: if the current configuration is (q, v), and the edge $\epsilon = (q, q')$ is traveled, then the next configuration is (q', v') where v' is obtained by adding the vector v to the direction vector specified on ϵ, $D(\epsilon)$. This path is an R-path if all vectors along the path have nonnegative integer coordinates. We represent an R-path as a sequence of configurations starting with (q_0, v_0). For a finite R-path $p = (q_0, v_0), (q_{i_1}, v_{i_1}) \ldots (q_{i_k}, v_{i_k})$ define $h(p) = h(q_0)h(q_{i_1}) \ldots h(q_{i_k})$. Similarly, for an infinite R-path $p = (q_0, v_0), (q_{i_1}, v_{i_1}) \ldots$, define $h(p) = h(q_0)h(q_{i_1}) \ldots$ An R-path p is proper if it is finite or if $\exists U > 0$ such that $\forall k \geq 0, v_{i_k} \leq (U, U, \ldots U)$(r times) (for notational convenience, assume $i_0 = 0$).

Definition 4

$$
\begin{aligned}
L^{finite}(V) = &\quad \{h(p) | p \text{ is a finite } R\text{-path}\};\\
L^{proper}(V) = &\quad L^{finite}(V) \bigcup \{h(p) | p \text{ is a proper infinite } R\text{-path}\};\\
L^{extended}(V) = &\quad L^{finite}(V) \bigcup \{h(p) | p \text{ is an infinite } R\text{-path}\};
\end{aligned}
$$

Below we sketch a transformation Γ_2 which maps a λ-free labeled PN $N = (P, T, B, F, M_0, \sum, h)$ to a labeled VASS $V = (S, E, D, q_0, v_0, \sum', h')$. Γ_2 will satisfy the property stated in the first part of Theorem 2. If the net has r places, the VASS is r-dimensional. Γ_2 has the following parts: *(i):* To represent the fact that a transition t is enabled only when each place p has a value $\geq B(p, t)$, and the fact that each place increases in value by $F(p, t) - B(p, t)$ after t is fired, we construct a component corresponding to t as shown in Step 1 of Figure 2. The vector $-B(., t)$ acts as a guard for the edge transition from q_t^1 to q_t^2 to occur. The vector $F(., t)$ on the edge from q_t^2 to q_t^3 increases the vector in the next configuration by $F(., t)$. Lastly, q_t^1, q_t^2, and q_t^3 have the same label as t; *i.e.* $h'(q_t^1) = h'(q_t^2) = h'(q_t^3) = h(t)$. *(ii):* Step 2 in Figure 2 prepares the VASS for the next transition. The notation $\stackrel{0}{\rightarrow}$ denotes the zero vector. *(iii):* Step 3 in Figure 2 provides the initial configuration for the VASS. State q_0 is labeled by a new symbol x not present in $\sum$. The initial vector v_0 equals the initial marking M_0.

With the construction so far, every firing sequence $t_{i_1} t_{i_2} \ldots t_{i_k}$ corresponds to an R-path such that the states along this path (ignoring the vectors) are $q_0 q_{t_{i_1}}^1 q_{t_{i_1}}^2 q_{t_{i_1}}^3 \ldots q_{t_{i_k}}^1 q_{t_{i_k}}^2 q_{t_{i_k}}^3$, and vice-versa. However, Steps 2 and 3 introduce the following problem. Suppose $L^{finite}(N_1) = L^{finite}(N_2)$ for two nets N_1 and N_2. Further, suppose N_1 has a transition t labeled y and suppose that none of the transitions of N_2 are labeled y. Clearly, in this case, t is never enabled. However transformation Γ_2 will create VASS $\Gamma_2(N_1) = V_1$ such that V_1 has R-paths that end in state q_t^1, but which cannot be extended to include states q_t^2 and q_t^3. This happens due to the edges introduced in Steps 2 and 3 of Γ_2. Since N_2 does not have any transition labeled x, VASS $\Gamma_2(N_2) = V_2$ will not have such R-paths. Thus, despite the equality $L^{finite}(N_1) = L^{finite}(N_2)$, the corresponding languages in the VASS domain are not equal. To correct this problem, we introduce the edges and states shown in Step 4 of Figure 2. There is a state q_σ, labeled σ, corresponding to every $\sigma \in \sum$. From each state q_t^3, and from state q_0, we add edges labeled with the zero vector to each of these new states. Figure 3 shows the VASS obtained from the PN in Figure 1(i) using Γ_2.

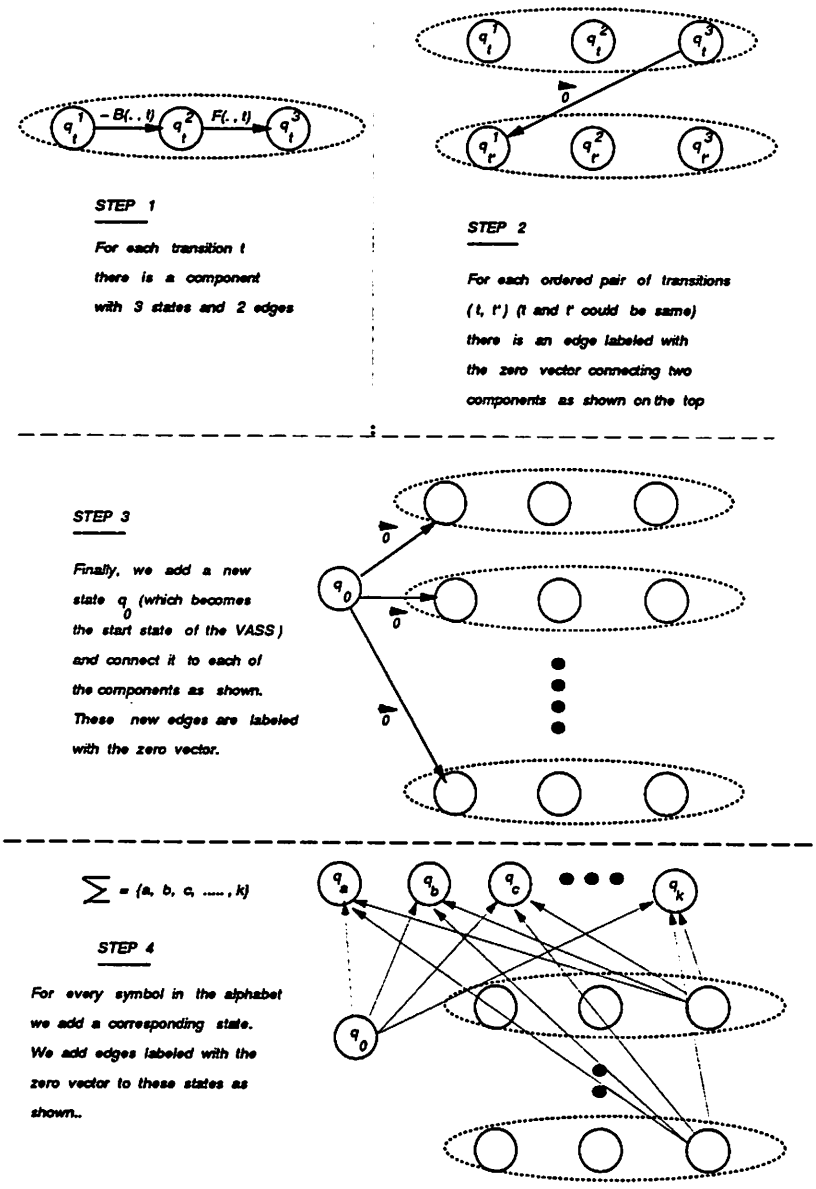


Figure 2: Steps of Γ_2

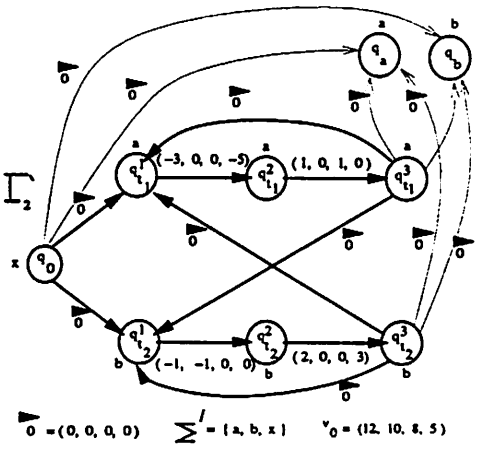


Figure 3: VASS constructed from Γ_2

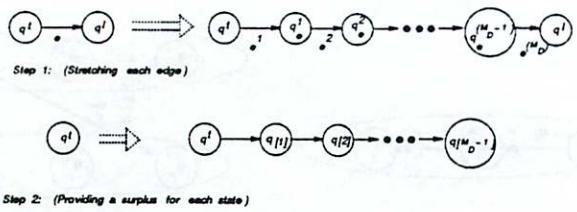


Figure 4: VASS to Restricted VASS – Transformation Γ_3

Part (1) of the theorem below follows from the construction of Γ_2. Part (2) follows from Part (1) and Corollary 1.

Theorem 2 *(1) Let N_1, N_2 be two λ-free labeled PN and let $\Gamma_2(N_1) = V_1$ and $\Gamma_2(N_2) = V_2$. Then $L^\epsilon(N_1) = L^\epsilon(N_2)$ iff $L^\epsilon(V_1) = L^\epsilon(V_2)$ where "ϵ" is one of "finite", "proper", or "extended"; and*
(2) Given 2 arbitrary labeled VASSes V and V', the equality of the languages $L^\epsilon(V)$ and $L^\epsilon(V')$ is undecidable, where "ϵ" is one of "finite", "proper", or "extended".

2.3 Restricted Labeled VASS

We now describe two restrictions on our VASS model. R-paths in a VASS with these restrictions can be "mimicked" with no further alterations by executions in a concurrent system $(P, \mathcal{S})$ constructed using a transformation presented in [9]. This will eventually lead us to the proof of undecidability regarding equality of languages of executions of two CS.

The restrictions are: (1) Each direction vector can have at most one non-zero coordinate, and the value of this coordinate can be ± 1; and (2) the initial vector is the zero vector ($\overline{0}$). We usually write VR, etc. to denote a restricted VASS.

We now describe transformation Γ_3 from a VASS to a restricted VASS. Without loss of generality, we assume that the input VASS's initial vector is the zero vector; otherwise, we perform the following changes to the input VASS: (*i*) Add a new state q_0' to the VASS. This state is the new initial state. The new initial vector of the VASS is the zero vector; (*ii*) Add an edge from q_0' to the previous initial state q_0 and label it with the previous initial vector v_0; and (*iii*) label q_0' with a new symbol x. If the original input VASS is V and the new VASS is V', then, clearly, $w \in L^\epsilon(V)$ iff $xw \in L^\epsilon(V')$, where "e" is one of "finite", "proper", or "extended".

Γ_3 consists of 2 steps as shown in Figure 4. Step 1 first renames state q_i of the input VASS to q^i. Second, each edge of the input VASS is replaced by a sequence of edges and new states that factor the original edge into unit and 0 moves. The number of states in the sequence equals $M_D - 1$, where M_D is any value greater than the maximum of the sum of the absolute value of the coordinates of any direction vector of the input VASS. The M_D edges in the sequence are labeled with direction vectors that satisfy the following conditions: (*i*) each vector is either the zero vector, or it has exactly one non-zero component which is either 1 or -1; and (*ii*) their componentwise vector addition yields the direction vector of the edge in the input VASS being replaced. For example, if the edge being replaced is labeled with direction vector $v = (2, 0, -3)$, and if M_D is 7, then the following 7 vectors $(1,0,0), (1,0,0), (0,0,-1), (0,0,-1), (0,0,-1), (0,0,0)$, and $(0,0,0)$ can be chosen to label the edges in the link. The head and tail states of each link are labeled by the same symbols that appear on the head and state of the edge being replaced. The intermediate $M_D - 1$ states are labeled by a new symbol x not present in the alphabet of the input VASS. For an edge ϵ connecting states q^t and q^l, we denote the new states by q_ϵ^1 thru $q_\epsilon^{(M_D-1)}$ as shown in Figure 4.

From the construction so far, if states $q_{i_1} q_{i_2} \ldots q_{i_k}$ appear along an R-path in the original input VASS, then there is an R-path in the restricted VASS such that states $q^{i_1} q_{\epsilon_{p_1}}^1 \ldots q_{\epsilon_{p_1}}^{(M_D-1)} q^{i_2} q_{\epsilon_{p_2}}^1 \ldots q_{\epsilon_{p_2}}^{(M_D-1)} q^{i_3} \ldots q^{i_k}$ appear along this R-path, where ϵ_{p_j} is the edge that connects states q^{i_j} and $q^{i_{j+1}}$. A similar result holds in the reverse direction.

Step 1 introduces the following problem: an R-path in the restricted VASS to end in an intermediate state of a linear sequence of $M_D - 1$ states. For example, for some R-path p in the original input VASS, suppose we reach a configuration $(q_i, (1,0,2))$, and suppose there is an edge ϵ labeled with direction vector $(-2,0,-2)$ directed out of state q_i. Clearly, p

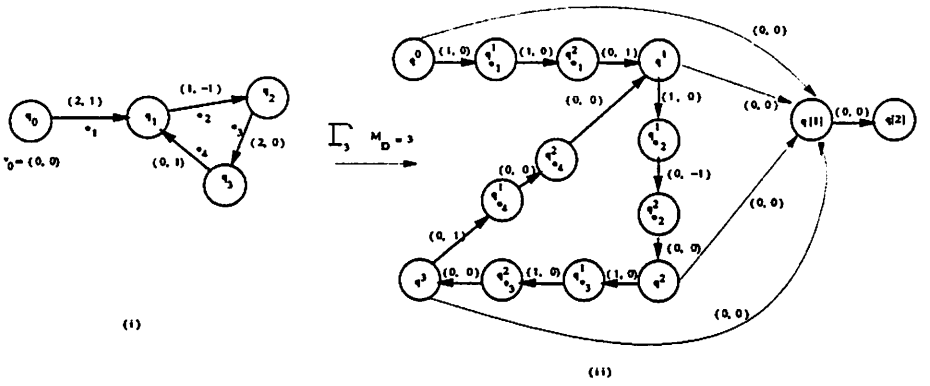


Figure 5: Example – Transformation Γ_3

cannot be extended to include this edge if p is to remain an R–path. Using Step 1 of Γ_3, the R–path p' in the restricted VASS corresponding to p, can be extended from configuration $(q^i, (1, 0, 2))$ to include some of the intermediate states in the chain that replaces edge ϵ. Thus, Step 1 can map VASSes with equal languages to restricted VASSes with distinct languages. Step 2 corrects this problem as follows (see Figure 4): (i) create a chain of $M_D - 1$ states (denoted by $q[1], q[2], \ldots q[M_D - 1]$), where the edges in this chain are labeled with the zero vector and the states are labeled by symbol x; and (ii) add an edge labeled with the zero vector from each state q^i to $q[1]$. This new part of the restricted VASS ensures that any R–path ending in a state q^i can always be augmented to include $M_D - 1$ states which are labeled with symbol x.

Figure 5(ii) shows the restricted VASS obtained by applying the three steps above for the VASS of Figure 5(i).

Theorem 3 *(1) Let V_1 and V_2 be two VASSes which both start from the zero vector. Let M_D be any value greater than the maximum of the sum of the absolute value of the coordinates of the direction vectors in V_1 and V_2. Let $VR_1 = \Gamma_3(V_1)$ and $VR_2 = \Gamma_3(V_2)$. Then, $L^\epsilon(V_1) = L^\epsilon(V_2)$ iff $L^\epsilon(VR_1) = L^\epsilon(VR_2)$, where "$\epsilon$" is one of "finite", "proper", or "extended":*
(2) Given a restricted VASS VR in r dimensions, we can obtain another restricted VASS VR' in $(r + k)$ dimensions for any $k \geq 1$ such that $L^\epsilon(VR') = L^\epsilon(VR)$, where "$\epsilon$" is one of "finite", "proper", or "extended"; and (3) Given 2 arbitrary r–dimensional restricted VASSes VR and VR', the equality of the languages $L^\epsilon(VR)$ and $L^\epsilon(VR')$ is undecidable, where "ϵ" is one of "finite", "proper", or "extended".

Proof: (1) follows from construction Γ_3. To prove (2), we only need to increase the dimensions of the direction vectors in VR to $r + k$. For each vector, these additional k coordinates are zero. We need (2) so that the two concurrent systems constructed can use the same communication alphabet. (3) follows from Parts (1) and (2), and Part (2) of Theorem 2. □

We now use the transformation described in [9] (which we denote by Γ_4) to convert a restricted VASS to a CS. This transformation maps R–paths in the restricted VASS to appropriate execution sequences in the CS. For the sake of completeness, we describe briefly this transformation. The synchronizer state–edge structure is identical to that of the restricted VASS. The start state of the synchronizer is the same as that of the VASS. The set $\mathcal{AP}$ is identical to the set of symbols that annotate the states of the VASS. If the restricted VASS is r–dimensional, then there are r pairs of complementary symbols in the communication alphabet of the CS. For the ith coordinate, the corresponding pair of complementary symbols is denoted by α_i and $\overline{\alpha_i}$. If a direction vector along an edge in the restricted VASS is $+1$ (-1) in the lth coordinate for some l, $1 \leq l \leq r$, it is replaced by α_l $(\overline{\alpha_l})$ in the synchronizer; and if it is the zero vector, it is replaced by ϵ. The user process definition P is as follows: P has $r + 1$ states denoted by $s_0, s_1, \ldots s_r$. s_0 is the start state of P. For every i, $1 \leq i \leq r$, there is an arc from s_0 to s_i labeled with symbol α_i. For every i, $1 \leq i \leq r$, there is an arc from s_i to s_0 labeled with symbol $\overline{\alpha_i}$. Thus, every edge transition labeled with a direction vector that has a $+1$ in the lth coordinate is "simulated" by a communication between a user process in state s_0 and the synchronizer. This causes a transition in the user process to state s_l and increases the number of user processes in state s_l by 1. Similarly, every edge transition labeled with a direction vector that has a -1 in the lth coordinate is simulated by a communication between a user process in state s_l and the synchronizer and also decreases

the number of user processes in state s_l by 1.

Part (1) of the theorem below follows directly from construction Γ_4. Part (2) follows from Part (1) and from Theorem 3.

Theorem 4 *(1) Let VR be a restricted VASS and let $(P,\mathcal{S})$ be the CS obtained using Γ_4. Then $L'(VR) = L'_{\mathcal{S}}(P,\mathcal{S})$, where "$\epsilon$" is one of "finite", "proper", or "extended". Further, let VR and VR' be two r-dimensional restricted VASSes. Let $(P,\mathcal{S}) = \Gamma_4(VR)$ and $(P',\mathcal{S}') = \Gamma_4(VR')$. Then $(P,\mathcal{S})$ and $(P',\mathcal{S}')$ are consistent.*
(2) Given two arbitrary consistent systems, $(P,\mathcal{S})$ and $(P',\mathcal{S}')$, the equality of languages $L'_{\mathcal{S}}(P,\mathcal{S})$ and $L'_{\mathcal{S}'}(P',\mathcal{S}')$ is undecidable, where "ϵ" is one of "finite", "proper", or "extended".

3 Undecidability of Bisimilarity

Theorem 5 *Given two arbitrary consistent systems $(P,\mathcal{S})$ and $(P',\mathcal{S}')$, it is undecidable whether $(P,\mathcal{S}) \equiv^{\epsilon}_{LTL} (P',\mathcal{S}')$, where "$\epsilon$" is one of "finite", "extended", or "proper".*

Proof: For the case when "e" is either "finite" or "extended", the proof is straightforward and outlined in Appendix B. When "e" is proper, we use a transformation Γ_5 similar to Γ_1. Γ_5 is described in Appendix B. In brief, Γ_5 transforms a CS into another such that (a) the latter CS has no infinite proper execution sequences; and (b) for each finite execution sequence in the former CS there is a corresponding finite execution sequence in the latter and vice-versa. Thus, using Γ_5 we map the problem of determining equality of languages $L^{finite}_{\mathcal{S}}(P,\mathcal{S})$ and $L^{finite}_{\mathcal{S}'}(P',\mathcal{S}')$ to that of determining whether $(P_1,\mathcal{S}_1) \equiv^{\epsilon}_{LTL} (P'_1,\mathcal{S}'_1)$ where $(P_1,\mathcal{S}_1)$ (respectively $(P'_1,\mathcal{S}'_1)$) is constructed using Γ_5 from $(P,\mathcal{S})$ (respectively $(P',\mathcal{S}')$). Using part 2 of Theorem 4 we get the required result. □

To prove undecidability of bisimilarity with respect to $LTL/\mathbf{X}$ consider a transformation Γ_6 which changes the synchronizer definition in a CS such that a new symbol x is inserted after every symbol in a word w in the language of the original CS. Thus, if the original CS is $(P,\mathcal{S})$ and the new CS constructed from Γ_6 is $(P',\mathcal{S}')$, then $w \in L^{\epsilon}_{\mathcal{S}}(P,\mathcal{S})$ iff $w' \in L^{\epsilon}_{\mathcal{S}'}(P',\mathcal{S}')$, where "e" is one of "finite", "proper", or "extended", and w' is obtained from w by inserting x after every symbol in w.

Formally, Γ_6 is as follows: (1) The new process definition P' is the same as P; (2) The propositional alphabet of $(P',\mathcal{S}')$, $\mathcal{AP}' = \mathcal{AP} \bigcup \{x\}$ where x is a propositional letter not in $\mathcal{AP}$. The communication alphabet of $(P',\mathcal{S}')$ is the same as that of $(P,\mathcal{S})$. (3) $\mathcal{S}'$ is obtained from $\mathcal{S}$ as follows: every edge ϵ in $\mathcal{S}$ from state q to q' with communication symbol c (ϵ) is replaced by two edges ϵ_1 and ϵ_2, and a new state q_ϵ as follows: ϵ_1 goes from q to q_ϵ, and ϵ_2 goes from q_ϵ to q'; ϵ_1 is labeled with ϵ, and ϵ_2 with c (ϵ). State q_ϵ is labeled with the propositional letter x.

Using a transformation Γ_7, which composes Γ_5 and Γ_6, we get the following result (see Appendix B for proof):

Theorem 6 *Given two arbitrary consistent systems $(P,\mathcal{S})$ and $(P',\mathcal{S}')$, it is undecidable whether $(P,\mathcal{S}) \equiv^{\epsilon}_{LTL/\mathbf{X}} (P',\mathcal{S}')$ where "ϵ" is either "finite", "extended", or "proper".*

We now consider the following restricted bisimulation equivalence problems:
Problem 1: For an arbitrary synchronizer $\mathcal{S}$ and two arbitrary processes P_1 and P_2, is $L^{\epsilon}_{\mathcal{S}}(P_1,\mathcal{S}) = L^{\epsilon}_{\mathcal{S}}(P_1,\mathcal{S})$, where ϵ is one of "*finite*", "*proper*", or "*extended*".
Problem 2: For an arbitrary synchronizer $\mathcal{S}$ and two arbitrary processes P_1 and P_2, is $(P_1,\mathcal{S}) \equiv^{\epsilon}_{F} (P_2,\mathcal{S})$, where F stands for either LTL or $LTL/\mathbf{X}$ and ϵ stands for one of *finite*, *proper*, or *extended*.

We sketch a transformation Γ_8 which constructs from two arbitrary consistent CS $(P,\mathcal{S})$ and $(P',\mathcal{S}')$ a synchronizer $\mathcal{S}_a$ and two process definitions P_1 and P_2 such that the property stated below in Proposition 3 holds. Using this proposition and Theorems 4, 5 and 6 we conclude that the above two problems are undecidable (Theorem 7).

Note that the CS obtained using transformations Γ_4, Γ_5, Γ_6, and Γ_7, are such that the user processes are ϵ-free. Thus, the undecidability results stated in Theorems 4, 5, and 6 apply to CS in which P and P' are arbitrary ϵ-free user process definitions. The input to transformation Γ_8 are two CS $(P,\mathcal{S})$ and $(P',\mathcal{S}')$, where both P and P' are ϵ-free.

The first step of Γ_8 eliminates ϵ transitions in $\mathcal{S}$ and $\mathcal{S}'$. This is done for $\mathcal{S}$ as follows. We introduce a new communication symbol c_{new}, and its complementary symbol $\overline{c_{\text{new}}}$. Every ϵ labeled transition of $\mathcal{S}$ is now labeled with $\overline{c_{\text{new}}}$. P is changed as follows: For every state of P, add an edge from that state to itself; this edge is labeled with c_{new}. Let this user process definition be P_1. We do a similar change to P' and $\mathcal{S}'$, except that we use another complementary communication symbol pair $c_{\text{new}'}$, and $\overline{c_{\text{new}'}}$ distinct from any of the other communication symbols. Let this user process definition be P_2.

At the second step, we join $\mathcal{S}$ and $\mathcal{S}'$ to construct a synchronizer $\mathcal{S}_a$ as follows: We add a new state q_{new}, and add edges from q_{new} to all the start states of $\mathcal{S}$, and edges from q_{new} to all the start states of $\mathcal{S}'$. An edge from q_{new} to a start state of $\mathcal{S}$ ($\mathcal{S}'$) is labeled by communication symbol $\overline{c_{\text{new}}}$ ($\overline{c_{\text{new}'}}$). q_{new} is the new start state of $\mathcal{S}_a$; it is labeled by a new propositional symbol x.

Proposition 3 *$L^{\epsilon}_{\mathcal{S}}(P,\mathcal{S}) = L^{\epsilon}_{\mathcal{S}'}(P',\mathcal{S}')$ iff $L^{\epsilon}_{\mathcal{S}_a}(P_1,\mathcal{S}_a) = L^{\epsilon}_{\mathcal{S}_a}(P_2,\mathcal{S}_a)$, where "$\epsilon$" is "finite", "extended", or "proper".*

Proof: From construction Γ_8. □

Part (1) of the theorem below follows from Theorem 4 and Proposition 3. Part (2) follows by using transformations similar to Γ_5 and Γ_6 and from Theorems 5 and 6.

Theorem 7 *(1) Given two arbitrary process definitions P and P' and an arbitrary synchronizer $\mathcal{S}$, the problem of determining if $L^{\epsilon}_{\mathcal{S}}(P,\mathcal{S}) = L^{\epsilon}_{\mathcal{S}}(P',\mathcal{S})$ is undecidable, where "ϵ" is one of "finite", "proper", or "extended". (2) Given two arbitrary process definitions P and P' and an arbitrary synchronizer $\mathcal{S}$, it is undecidable whether $(P,\mathcal{S}) \equiv^{\epsilon}_{F} (P',\mathcal{S})$, where "$\epsilon$" is one of "finite", "proper", or "extended", and F is either LTL or $LTL/\boldsymbol{X}$.*

Using similar methods, we can prove the following result for another restricted variant of the bisimilarity problem:

Theorem 8 *Given two arbitrary synchronizer definitions $\mathcal{S}$ and $\mathcal{S}'$ and an arbitrary process definition P, the problem of determining equality of languages $L^{\epsilon}_{\mathcal{S}}(P,\mathcal{S})$ and $L^{\epsilon}_{\mathcal{S}'}(P,\mathcal{S}')$ is undecidable, where "ϵ" is one of "finite", "proper", or "extended". Similarly, the problem of determining $(P,\mathcal{S}) \equiv^{\epsilon}_{F} (P',\mathcal{S})$, where "$\epsilon$" is one of "finite", "proper", or "extended", and F is either LTL or $LTL/\boldsymbol{X}$ is also undecidable.*

4 Conclusion

In this paper, we have shown that bisimulation equivalence is undecidable for CS of the type presented in [9]. We have also shown that certain restricted variants of the bisimilarity problem (single synchronizer and two user processes, and, single user process and two synchronizers) are also undecidable. These negative results indicate that it is not possible to compare two such systems with respect to LTL and $LTL/\mathbf{X}$. One important direction for future research is to identify simpler notions of equivalences for such CS which are decidable. These "simpler" notions of equivalences, however, should expose "important" similarities (or properties of interest) between such CS.

References

[1] Antii Valmari and Matthew Clegg. Reduced Labelled Transition Systems Save Verification Effort. *CONCUR*, 1991.

[2] K. Ramamritham. Synthesizing Code For Resource Controllers. *IEEE Transactions on Software Engg.* August 1985.

[3] M. C. Browne, E. M. Clarke, and O. Grumberg. Characterizing Finite Kripke Structures in Propositional Temporal Logic. *Theoretical Computer Science*, 1988.

[4] M. Lamport. What good is Temporal Logic ? *Proc. International Federation for Information Processing*, 1983.

[5] Mahesh Girkar and Robert Moll. Stability Analysis of Concurrent Systems with an Indefinite Number of Processes. *CS Technical Report 92-72, Dept. of Computer Science, Univ. of Massachusetts*, November 1992.

[6] R. Kaivola and A. Valmari. Using Truth-Preserving Reductions to Improve the Clarity of Kripke-models. *CONCUR*. 1991.

[7] R. Kaivola and A. Valmari. The Weakest Compositional Semantic Equivalence Preserving Nexttime-less Linear Temporal Logic. *CONCUR*. 1992.

[8] Rudiger Valk and Guy Vidal-Naquet. Petri Nets And Regular Languages. *Journal of Computer and System Sciences*. 1981.

[9] S. German and A. P. Sistla. Reasoning About Systems With Many Processes. *JACM*. July 1992.

[10] S. Rao. Kosaraju. Decidability Of Reachability In Vector Addition Systems. *Proceedings of the 16th ACM Symposium on Theory of Computing*. May 1982.

[11] Ugo Buy and Robert Moll. Analysis And Synthesis Of Inter-process Communication Code. To appear.

Appendix A — Concurrent System and Temporal Logic

In this appendix we describe our model for CS with an indefinite number of identical user processes. The description and notation is from [9]. We begin with a description of LTL and $LTL/\mathbf{X}$.

LTL uses $\mathcal{AP}$ — a finite set of atomic propositions, the constant **True**, the connectives $\neg$, $\vee$, and the temporal operators $\mathbf{X}$ (nexttime) and $\mathbf{U}$ (until). LTL is the smallest set of formulas obtained as follows: (a) the constant **True** and the atomic propositions are formulas; (b) if f and g are formulas, then $\neg f$, $f \vee g$, $\mathbf{X}f$, and $f\mathbf{U}g$ are formulas. $LTL/\mathbf{X}$ is the set of LTL formulas obtained without using the nexttime operator.

An *interpretation* is a pair (t, i) where $t = t_0, t_1, \ldots$ is a finite or infinite sequence of subsets of $\mathcal{AP}$ and i is a nonnegative integer. $(t, i) \models f$ is used to denote that the interpretation satisfies a formula f. $\models$ is defined inductively as follows:

1. $(t, i) \models$ **True**
2. $(t, i) \models A$, where $A \in \mathcal{AP}$, *iff* $A \in t_i$.
3. $(t, i) \models \neg f$ *iff* $(t, i) \not\models f$.
4. $(t, i) \models f \vee g$ *iff* $(t, i) \models f$ or $(t, i) \models g$.
5. $(t, i) \models \mathbf{X}f$ *iff* $(t, i+1) \models f$.
6. $(t, i) \models (f\mathbf{U}g)$ *iff* there exists $k \geq i$ such that $(t, k) \models g$, and for all $j, i \leq j < k, (t, j) \models f$.

Finally, we say that $t \models f$ *iff* $(t, 0) \models f$. In words, if $t \models f$, we say that the temporal formula f is *valid* for the sequence of subsets of atomic propositions that appear in t. We use the words "sequence of subsets of atomic propositions" and "interpretation" interchangeably.

Our model for concurrency consists of a unique *synchronizer* process and many identical *user* processes. We first describe the alphabet used by the processes to interact with each other. Intuitively, the alphabet provides a mechanism for synchronization between two processes. It is thus useful to model operations which request/release a resource. A communication alphabet $\sum$ is a union of mutually disjoint sets $\sum^+, \sum^-$, and ϵ. ϵ is used to represent internal process transitions. $\sum^+ (\sum^-)$ is the set of action (complements of the action) symbols. Thus, if $c \in \sum^+$, then its complement $\overline{c} \in \sum^-$. Each member of $\sum$ is a communication symbol. Two symbols c, d are complementary if $c = \overline{d}$.

A synchronizer $\mathcal{S}$ is a 4-tuple (S_S, R_S, I_S, Φ_S) where S_S is a finite set of states, $R_S \subseteq S_S \times S_S \times \sum$ is a set of transitions, $I_S \subseteq S_S$ is the set of initial states, and $\Phi : S_S \rightarrow 2^{\mathcal{AP}}$ is a function that labels each state in S_S with atomic propositions that are true in that state. A user process is a 4-tuple (S_P, R_P, I_P, Φ_P). S_P, R_P, I_P and Φ_P are similar to S_S, R_S, I_S and Φ_S.

We now formalize the notion of an "execution" in a CS with k user processes and a synchronizer. Such a system is denoted by $(P^k, \mathcal{S})$. Subsequently, we formalize the notion of "execution" in a CS with an indefinite number of user processes and a synchronizer (such a system is denoted by $(P, \mathcal{S})$). Using these formalizations, we define three different sets of executions, namely the set of finite executions, the set of proper executions, and, the set of extended executions.

For $k \geq 1$, a *global state* of $(P^k, \mathcal{S})$ is an element of $(S_{\mathcal{S}} \times (S_P)^k)$. A global state $\delta = (s_1, s_2, \ldots s_{k+1})$ is *initial* iff $s_1 \in I_{\mathcal{S}}$, and for $2 \leq i \leq k+1$, $s_i \in I_P$. Let $S_1 = S_{\mathcal{S}}$, and for $2 \leq i \leq k+1$, let $S_i = S_P$. Also, let R_1, Φ_1 be the same as $R_{\mathcal{S}}$ and $\Phi_{\mathcal{S}}$ respectively, and for each $i, 2 \leq i \leq k+1$, let R_i be the same as R_P and Φ_i be the same as Φ_P. We refer to the synchronizer as Process 1, and the k user processes as Process 2, ..., Process $k+1$.

Distinct processes i and j $(1 \leq i, j \leq k+1)$ can *communicate* in a global state $(s_1, s_2, \ldots s_{k+1})$ *iff* there exist $c \in \sum$, $c \neq \epsilon$, $s \in S_i$, $s' \in S_j$ such that $(s_i, s, c) \in R_i$ and $(s_j, s', \bar{c}) \in R_j$. Process i is *enabled* in a global state δ if either it can communicate with another process in δ, or for some $s \in S_i$, $(s_i, s, \epsilon) \in R_i$.

Starting with an initial state, we obtain the next global state in an execution sequence of the system by choosing one of the enabled transitions. To make this more precise, let us define $\delta[i]$ for each i, $1 \leq i \leq k+1$, to be the state s_i where $\delta = (s_1, s_2, \ldots s_{k+1})$. We say that a global state γ can be *reached* from global state δ by an internal transition of process i if $(\delta[i], \gamma[i], \epsilon) \in R_i$ and for $1 \leq j \leq k+1, j \neq i, \delta[j] = \gamma[j]$. Similarly γ can be reached from δ by communication between processes i and j, $i \neq j$, if for some $c \in \sum, (\delta[i], \gamma[i], c) \in R_i, (\delta[j], \gamma[j], \bar{c}) \in R_j$, and for $l \neq i, j, 1 \leq l \leq k+1, \delta[l] = \gamma[l]$. We say that γ can be reached from δ by one computation step if γ can be reached from δ by an internal transition or by communication between a pair of processes.

An *execution sequence* ϵ in a system $(P^k, \mathcal{S})$ is a finite or infinite sequence of global states $\sigma_0, \sigma_1, \ldots$ where σ_0 is an initial state, and for each $i \geq 0$, σ_{i+1} can be reached from σ_i by one computation step.

An execution sequence in the system $(P, \mathcal{S})$ is defined similarly, except that now a global state δ is an element from $(S_{\mathcal{S}} \times S_P \times S_P \ldots)$. A global state $(s_1, s_2, \ldots)$ is initial iff $s_1 \in I_{\mathcal{S}}$ and $\forall i \geq 2, s_i \in I_P$. Thus, an arbitrarily large number of user processes can participate in an execution.

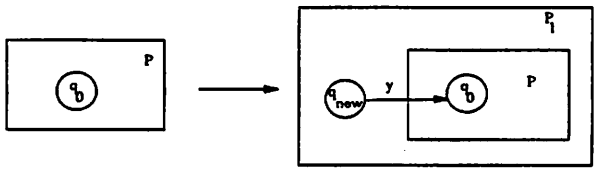
For an execution sequence ϵ (either from $(P^k, \mathcal{S})$ or $(P, \mathcal{S})$), let ϵ_i denote the sequence ϵ restricted to the states of the ith process. Thus, ϵ_1 denotes the sequence of synchronizer states that appear in ϵ. Let $\Phi_i(\epsilon_i)$ denote the sequence of subsets of propositional symbols obtained by applying Φ_i to the states in ϵ_i. We define the following sets:

Definition 5

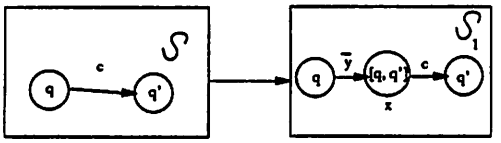
$$
\begin{aligned}
L_{\mathcal{S}}^{finite}(P^k, \mathcal{S}) &= \{\Phi_1(\epsilon_1) | \epsilon \textit{ is a finite execution of } (P^k, \mathcal{S})\} \\
L_{\mathcal{S}}^{finite}(P, \mathcal{S}) &= \bigcup_{k \geq 0} L_{\mathcal{S}}^{finite}(P^k, \mathcal{S}) \\
L_{\mathcal{S}}^{proper}(P, \mathcal{S}) &= L_{\mathcal{S}}^{finite}(P, \mathcal{S}) \cup \\
&\quad \{\Phi_1(\epsilon_1) | \epsilon \textit{ is an infinite execution of } (P^k, \mathcal{S}) \textit{ for some } k \geq 0\} \\
L_{\mathcal{S}}^{extended}(P, \mathcal{S}) &= L_{\mathcal{S}}^{finite}(P, \mathcal{S}) \cup \{\Phi_1(\epsilon_1) | \epsilon \textit{ is an infinite execution of } (P, \mathcal{S})\}
\end{aligned}
$$

Appendix B — Detailed Proofs

Proof of Part (1) of Proposition 2: For a PN Y, $L^{finite}(Y) \subseteq L^{extended}(Y)$. Therefore, if $L^{extended}(N) = L^{extended}(N')$ then $L^{finite}(N) = L^{finite}(N')$. Conversely, suppose $L^{finite}(N) = L^{finite}(N')$, and let $w \in L^{extended}(N)$. We will show that $w \in L^{extended}(N')$. Since $L^{finite}(N) = L^{finite}(N')$, if w is of finite length, $w \in L^{extended}(N')$. Suppose w is infinite. Since $L^{finite}(N) = L^{finite}(N')$, every finite prefix of w is in $L^{extended}(N')$. Starting with the initial marking M_0', consider a tree representation Φ of the firing sequences in N' as follows: The root node is labeled with M_0'; every node labeled with a marking M in Φ, has k children labeled with markings $M_1, M_2, \ldots, M_k$ respectively iff for each $i, 1 \leq i \leq k, \exists t \in T' : M(t > M_i$. We annotate every edge between two nodes by $h'(t)$, where t was the transition "taken" along that edge. It is easy to see that every word (finite or infinite) obtained by concatenating the annotations on



(i) Process Definition Change



For every edge between states q and q', a new state denoted by [q, q'] is introduced and the original edge replaced by two edges as shown. In [q, q'], "new" propositional symbol x is true.

(ii) Synchronizer Definition Change

Figure 6: Transformation Γ_5

the edges along a path in Φ is in $L^{extended}(\mathcal{N}')$. Also, Φ is "finite" branching because every node can have only a maximum of $|T'|$ immediate children nodes. Since every finite prefix of w is in $L^{extended}(\mathcal{N}')$, for each such prefix, there must be path in Φ starting with the root node. Thus, by Konig's Tree Lemma, there is a word $w' \in L^{extended}(\mathcal{N}')$ such that $w = w'$. □

Proof of Theorem 5: For the case when "e" is either "finite" or "extended", we first prove the equivalence of the following two statements: (1): $L^{\epsilon}_{\mathcal{S}}(P, \mathcal{S}) = L^{finite}_{\mathcal{S}'}(P', \mathcal{S}')$; and (2): $(P, \mathcal{S}) \equiv^{\epsilon}_{LTL} (P', \mathcal{S}')$. When ϵ is $finite\epsilon$, (1) $\rightarrow$ (2) is easy. To prove (2) $\rightarrow$ (1), suppose $(P, \mathcal{S}) \equiv^{finite}_{LTL} (P', \mathcal{S}')$. Let the propositional alphabet used to label the states of the two systems be $\mathcal{AP}$. Using the nexttime operator **X**, construct an LTL formula f for every finite sequence (say w) of subsets of $\mathcal{AP}$ such that if w' is any finite sequence of subsets of $\mathcal{AP}$ that models f (*i.e.* $w' \models f$), then $w = w'$. For example, suppose $w = a_1 a_2 \ldots a_k$, where each $a_i \in 2^{\mathcal{AP}}$. Consider the following formula f

$$(\bigwedge_{p \in a_1} p) \bigwedge \mathbf{X}(\bigwedge_{p \in a_2} p) \ldots \bigwedge \mathbf{XX}\ldots k \text{ times } (\bigwedge_{p \in a_k} p)$$

Clearly, for any w', $w' \models f$ *iff* $w = w'$. Since such a formula either simultaneously holds in both the systems or fails in both the systems, it follows that if $w \in L^{finite}_{\mathcal{S}}(P, \mathcal{S})$ then it is also in $L^{finite}_{\mathcal{S}'}(P', \mathcal{S}')$ and vice-versa.

When ϵ is $extended$, once again, (1) $\rightarrow$ (2) is easy. To prove (2) $\rightarrow$ (1) observe that $L^{\mathcal{S}}_{extended}(P, \mathcal{S}) = L^{extended}_{\mathcal{S}'}(P', \mathcal{S}')$ iff $L^{finite}_{\mathcal{S}}(P, \mathcal{S}) = L^{finite}_{\mathcal{S}'}(P', \mathcal{S}')$. (This can be proved using Konig's Tree Lemma argument similar to the proof of Proposition 2.) Thus, if $L^{extended}_{\mathcal{S}}(P, \mathcal{S})$ and $L^{extended}_{\mathcal{S}'}(P', \mathcal{S}')$ are not equal, then, without loss of generality, there is a finite word w such that (a) w is a prefix of some word w' in $L^{extended}_{\mathcal{S}}(P, \mathcal{S})$; and (b) and no finite prefix of any word in $L^{extended}_{\mathcal{S}'}(P', \mathcal{S}')$ equals w. As in the previous claim, we can write an LTL formula f such that for any word w', $w' \models f$ *iff* $w = w'$. Clearly, $w' \models f$ in $(P, \mathcal{S})$; however, there is no word in $L^{extended}_{\mathcal{S}'}(P', \mathcal{S}')$ that models f. Hence, (2) does not hold.

When "e" is *proper*, we show that an instance of the equality problem of the language of finite executions for two arbitrary consistent systems $(P, \mathcal{S})$ and $(P', \mathcal{S}')$ can be transformed to the problem of checking whether $(P_1, \mathcal{S}_1) \equiv^{proper}_{LTL} (P_1', \mathcal{S}_1')$, where $(P_1, \mathcal{S}_1)$ and $(P_1', \mathcal{S}_1')$ are obtained from $(P, \mathcal{S})$ and $(P', \mathcal{S}')$ respectively using a transformation Γ_5. We now describe Γ_5 (see Figure 6). Γ_5 is similar to Γ_1 in principle. The main idea is to eliminate infinite proper execution sequences in the original CS.

The communication alphabet of $(P_1, \mathcal{S}_1)$ has a new pair of complementary symbols y and $\bar{y}$ in addition to the symbols in the communication alphabet of $(P, \mathcal{S})$. The propositional alphabet of $(P_1, \mathcal{S}_1)$ has a new symbol x in addition to the propositional symbols of $(P, \mathcal{S})$.

The definition of P_1 is the same as P except for the addition of a new state q_{new} and an arc from q_{new} to the initial state q_0 of P. This arc is labeled by the communication symbol y (see Figure 6(i)). q_{new} is the initial state of P_1.

$\mathcal{S}_1$ is obtained from $\mathcal{S}$ as shown in Figure 6(ii). The first step replaces every edge from state q to q' (q and q' can be identical) labeled with a communication symbol c (respectively ϵ) by two edges. The first edge is from q to a "new" state (denoted by $[q, q']$) labeled with communication symbol $\overline{y}$. The second edge is from $[q, q']$ to q' labeled with communication symbol c (respectively ϵ). New states of the form $[q, q']$ are labeled x. The remaining "old" states are labeled as in $\mathcal{S}$. $(P_1', \mathcal{S}_1')$ is obtained similarly from $(P', \mathcal{S}')$.

From Γ_5 it is easy to observe the following:

(1) If $(P, \mathcal{S})$ and $(P', \mathcal{S}')$ are consistent, then so are $(P_1, \mathcal{S}_1)$ and $(P_1', \mathcal{S}_1')$;

(2) $L_{\mathcal{S}}^{finite}(P, \mathcal{S}) = L_{\mathcal{S}'}^{finite}(P', \mathcal{S}')$ *iff* $L_{\mathcal{S}_1}^{proper}(P_1, \mathcal{S}_1) = L_{\mathcal{S}_1'}^{proper}(P_1', \mathcal{S}_1')$; and

(3) $L_{\mathcal{S}_1}^{proper}(P_1, \mathcal{S}_1) = L_{\mathcal{S}_1'}^{proper}(P_1', \mathcal{S}_1')$ *iff* $L_{\mathcal{S}_1}^{finite}(P_1, \mathcal{S}_1) = L_{\mathcal{S}_1'}^{finite}(P_1', \mathcal{S}_1')$. Using these observations, and the fact that for arbitrary consistent systems $(P, \mathcal{S})$ and $(P', \mathcal{S}')$ $(P, \mathcal{S}) \equiv_{LTL}^{e} (P', \mathcal{S}')$ is undecidable when "e" is "finite", we get the required result. □

Proof of Theorem 6: The proof is similar to the proof of Theorem 5. Using Γ_6 (described in Section 3), we first prove this theorem for the case when ϵ is "finite" or "extended". We make the following observations regarding Γ_6:

1. Let $(P, \mathcal{S})$ be any CS and let $(P', \mathcal{S}')$ be the CS obtained using Γ_6. Then, for every finite word $w = a_1 a_2 \ldots a_k$, $w \in L_{\mathcal{S}}^{finite}(P, \mathcal{S})$ *iff* $w' = a_1 x a_2 x a_3 \ldots x a_k \in L_{\mathcal{S}'}^{finite}(P', \mathcal{S}')$. Similarly, for every infinite word $w = a_1 a_2 \ldots$, $w \in L_{\mathcal{S}}^{extended}(P, \mathcal{S})$ *iff* $w' = a_1 x a_2 x a_3 \ldots \in L_{\mathcal{S}'}^{extended}(P', \mathcal{S}')$.

2. Let $(P_1, \mathcal{S}_1)$ and $(P_2, \mathcal{S}_2)$ be any two CS. Let $(P_1', \mathcal{S}_1')$ and $(P_2', \mathcal{S}_2')$ be obtained from $(P_1, \mathcal{S}_1)$ and $(P_2, \mathcal{S}_2)$ respectively using Γ_6. Then, $L_{\mathcal{S}_1}^{e}(P_1, \mathcal{S}_1) = L_{\mathcal{S}_2}^{e}(P_2, \mathcal{S}_2)$ iff $L_{\mathcal{S}_1'}^{e}(P_1', \mathcal{S}_1') = L_{\mathcal{S}_2'}^{e}(P_2', \mathcal{S}_2')$ where "e" is one of "finite", "proper" or "extended".

3. Suppose $(P_1, \mathcal{S}_1), (P_2, \mathcal{S}_2), (P_1', \mathcal{S}_1')$, and $(P_2', \mathcal{S}_2')$ are as stated in Observation 2 above. Then the following two statements are equivalent: (*i*): $L_{\mathcal{S}_1'}^{e}(P_1', \mathcal{S}_1') = L_{\mathcal{S}_2'}^{e}(P_2', \mathcal{S}_2')$; and (*ii*) $(P_1', \mathcal{S}_1') \equiv_{LTL/\mathbf{X}}^{e} (P_2', \mathcal{S}_2')$.

Observations 1 and 2 are straightforward to prove. We now prove Observation 3. When ϵ is "finite" (*i*) $\rightarrow$ (*ii*) is easy. Suppose (*i*) does not hold. Without loss of generality, let $w = a_1 x a_2 x \ldots x a_k$ be a finite word in $L_{\mathcal{S}_1'}^{finite}(P_1', \mathcal{S}_1')$ but $w \notin L_{\mathcal{S}_2'}^{finite}(P_2', \mathcal{S}_2')$. Let $f(a_i)$ stand for the formula $(\bigwedge_{p \in a_i} p)$. Consider the following formula f

$$f(a_1) \bigwedge (f(a_1) \mathbf{U} (x \bigwedge (x \mathbf{U} (f(a_2) \bigwedge \ldots f(a_k)))))$$

Clearly, if any word $w' \in L_{\mathcal{S}_2'}^{finite}(P_2', \mathcal{S}_2')$ is such that $w' \models f$, then $w = w'$. Thus, if (*i*) does not hold, then (*ii*) does not hold. For the case when ϵ is "extended", we use arguments similar to those presented for the same case in Theorem 5.

For the case when ϵ is "proper", we show that an instance of the equality problem of the language of finite executions for two consistent systems $(P, \mathcal{S})$ and $(P', \mathcal{S}')$ can be transformed to the problem of checking whether $(P_1, \mathcal{S}_1) \equiv_{LTL/\mathbf{X}}^{proper} (P_1', \mathcal{S}_1')$, where $(P_1, \mathcal{S}_1)$ and $(P_1', \mathcal{S}_1')$ is obtained from $(P, \mathcal{S})$ and $(P', \mathcal{S}')$ respectively using a transformation Γ_7.

Γ_7 does the work of transformation Γ_5 followed by the work of transformation Γ_6. Given two arbitrary systems $(P, \mathcal{S})$ and $(P', \mathcal{S}')$, let $(P_1, \mathcal{S}_1)$ and $(P_1', \mathcal{S}_1')$ be obtained using Γ_7 on $(P, \mathcal{S})$ and $(P', \mathcal{S}')$ respectively. We make the following observations regarding Γ_7.

(1) If $(P, \mathcal{S})$ and $(P', \mathcal{S}')$ are consistent, then so are $(P_1, \mathcal{S}_1)$ and $(P_1', \mathcal{S}_1')$;

(2) $L_{\mathcal{S}}^{finite}(P, \mathcal{S}) = L_{\mathcal{S}'}^{finite}(P', \mathcal{S}')$ *iff* $L_{\mathcal{S}_1}^{proper}(P_1, \mathcal{S}_1) = L_{\mathcal{S}_1'}^{proper}(P_1', \mathcal{S}_1')$; and

(3) $L^{proper}_{\mathcal{S}_1}(P_1, \mathcal{S}_1) = L^{proper}_{\mathcal{S}_1'}(P_1', \mathcal{S}_1')$ *iff* $L^{finite}_{\mathcal{S}_1}(P_1, \mathcal{S}_1) = L^{finite}_{\mathcal{S}_1'}(P_1', \mathcal{S}_1')$. Using these observations, we get the required result. □