# Synthesis of Concurrent Systems with Many Similar Processes

PAUL C. ATTIE
Florida International University
and
E. ALLEN EMERSON
The University of Texas at Austin

Methods for synthesizing concurrent programs from temporal logic specifications based on the use of a decision procedure for testing temporal satisfiability have been proposed by Emerson and Clarke and by Manna and Wolper. An important advantage of these synthesis methods is that they obviate the need to manually compose a program and manually construct a proof of its correctness. One only has to formulate a precise problem specification; the synthesis method then mechanically constructs a correct solution. A serious drawback of these methods in practice, however, is that they suffer from the state explosion problem. To synthesize a concurrent system consisting of $K$ sequential processes, each having $N$ states in its local transition diagram, requires construction of the global product-machine having at least $N^K$ global states in general. This exponential growth in $K$ makes it infeasible to synthesize systems composed of more than 2 or 3 processes. In this article, we show how to synthesize concurrent systems consisting of many (i.e., a finite but arbitrarily large number $K$ of) similar sequential processes. Our approach avoids construction of the global product-machine for $K$ processes; instead, it constructs a two-process product-machine for a single pair of generic sequential processes. The method is uniform in $K$, providing a simple template that can be instantiated for each process to yield a solution for any fixed $K$. The method is also illustrated on synchronization problems from the literature.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.1.2 [**Programming Techniques**]: Automatic Programming; D.1.3 [**Programming Techniques**]: Concurrent Programming; D.2.4 [**Software Engineering**]: Program Verification; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*mechanical verification*; I.2.2 [**Artificial Intelligence**]: Automatic Programming—*program synthesis*

---

A preliminary version of some of these results was presented at the ACM Symposium on Principles of Programming Languages, Austin, Texas, 1989, under the title "Synthesis of Concurrent Systems with Many Similar Sequential Processes." P.C. Attie was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-96-1-0221. E.A. Emerson was supported in part by NSF Grant CCR-9415496 and by SRC Contract 97-DP-388. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.
Authors' addresses: P.C. Attie, School of Computer Science, Florida International University, University Park, Miami, FL 33199; E.A. Emerson, Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712.

General Terms: Theory, Verification

Additional Key Words and Phrases: Concurrent programs, program synthesis, temporal logic

## 1. INTRODUCTION

We exhibit a method of synthesizing a system of $K$ *similar sequential processes* executing in parallel from a temporal logic specification, where $K$ is an arbitrarily large natural number. The method can be automated. It may be seen as an extension of the synthesis method of Emerson and Clarke [1982] and the related method of Manna and Wolper [1984]. The basic idea underlying both these earlier approaches to synthesis was to exploit the finite-model property for the temporal logic serving as a specification language: if a formula is satisfiable then it is satisfiable in a finite model. This makes it possible to develop a decision procedure for testing satisfiability of formulas in the specification logic. Given a formula, the decision procedure determines whether there exists a model of the formula. If such a model does exist, the procedure constructs a model which is a finite, labeled, directed graph that may be viewed as the global state transition diagram of a concurrent system meeting the specification. From this global state transition diagram, the *synchronization skeletons* of the individual sequential processes may be obtained by projecting out onto the coordinates corresponding to each of the individual sequential processes.

In principle, these earlier model-theoretic methods of program synthesis would have permitted synthesis of concurrent systems consisting of many (i.e., a finite but arbitrarily large number $K$ of) similar sequential processes running in parallel. In practice, however, the methods were not feasible for systems with many processes because of the state explosion problem: the global state transition diagram is essentially the automata-theoretic product of the sequential processes (these processes being viewed as state-machines), and is therefore of size exponential in $K$, in general. Thus, even if the individual sequential processes had only two local states each, the global state transition diagram would contain on the order of $2^K$ global states. More recent synthesis methods [Anuchitanukul and Manna 1994; Pnueli and Rosner 1989a; 1989b] suffer from the same intractability problem.

In this article, we propose a method (which we refer to as MP-synthesis, i.e., many-process synthesis, in the sequel) to overcome the synthesis state explosion problem. We show how to reduce the problem of synthesizing a concurrent system of $K$ similar sequential processes, each with $N$ local states—which would naively require building the global state transition diagram of size on the order of $N^K$—to the problem of constructing the product of small numbers of sequential processes, and in particular, the product of a pair of sequential processes (which we call a *pair-system*), thereby avoiding the exponential complexity in $K$.

One other highly desirable feature of MP-synthesis is that it is uniform in $K$. Applied to a "generic" pair of representative processes $i, j$ (since all processes are similar, any pair is representative, so we can select an arbitrary pair of *process indices* $i, j$, which we use as names for our representative pair) MP-synthesis yields a template for process $i$ that can be instantiated for each $K$ to get a synchronization skeleton for process $i$ in the $K$-process solution. The method is uniform so that the

$(K+1)$-process solution is analogous to the $K$-process solution. Furthermore, our method can generate systems under arbitrary *process interconnection* schemes, e.g., fully connected, ring, star. In our model of parallel computation, two processes are interconnected if and only if either (1) one process can inspect the local state of the other process or (2) both processes read and/or write a common variable, or both.

The method requires the pair-systems to satisfy certain technical assumptions, which are given in the sequel. Thus it is not completely general. Nevertheless, it is applicable in many interesting cases. In particular, we illustrate MP-synthesis by applying it to the $K$-process mutual exclusion problem, the $K$-process dining philosophers problem, and the $K$-process drinking philosophers problem [Chandy and Misra 1984; 1988] for arbitrarily large finite $K$. This is a significant improvement over the previous literature. For example, Wolper [1982] considers the dining philosophers problem, but only for three philosophers; even five philosophers made the synthesis problem computationally intractable. Similarly, the solutions synthesized in Emerson and Clarke [1982] and Manna and Wolper [1984] for the mutual exclusion problem were only for two processes. Consideration of just three processes made the problem infeasible for hand computation, while an actual implementation automating the Emerson and Clarke [1982] synthesis method [Inaba 1984] was unable to handle more than four processes. Thus, we are able to overcome the severe limitations previously imposed by the state explosion problem on the applicability of automatic temporal logic synthesis methods.

A crucial aspect of our method is its soundness: what correctness properties of the pair-systems are preserved by the $K$-process systems? We demonstrate, using the technical assumptions mentioned above, that (1) a propositional invariant of any pair-system is also an invariant of the $K$-process system and (2) a temporal leads-to property (expressed over a single process) satisfied by a pair-system is also satisfied by the $K$-process system. (A temporal leads-to property is of the following form: if condition 1 is true now, then condition 2 will eventually be true.) Part (1) means that safety properties which can be expressed pairwise are preserved. For example, mutual exclusion of $K$ processes from a critical section can be expressed pairwise by stating that no two processes are in the critical section at the same time. Part (2) means that typical progress properties *of a single process* are preserved. For example, absence of starvation in the mutual exclusion problem can be expressed as "if a process requests entry to the critical section, then it eventually gains entry to the critical section."

We do not consider other correctness properties, e.g., liveness properties expressed over a pair of processes. Such properties arise, for example, when expressing data dependencies in distributed or heterogeneous databases [Attie et al. 1996; Elmagarmid 1992; Klein 1991], e.g., "if transaction 1 commits then transaction 2 must also commit." There is no reason in principle that MP-synthesis would not preserve such properties; however, different technical assumptions than the ones we make here would be required. We defer this topic for future work.

The rest of this article is organized as follows. Section 2 defines the model of computation, and Section 3 presents the specification language. Section 4 presents our syntax for concurrent programs and other notation. Section 5 describes MP-synthesis, and Section 6 gives a proof of its soundness. Section 7 presents a way of implementing the programs we synthesize in terms of interprocess communication and synchronization primitives that are available in practice, namely asynchronous message passing and broadcast. Section 8 discusses some important aspects of the synthesis method: its complexity and scope of applicability. Finally, Section 9 discusses related work by others; Section 10 discusses possible extensions of the results presented here; and Section 11 presents our conclusions. Appendices B and C present those proofs that are omitted from the main text for reasons of space.

## 2. MODEL OF PARALLEL COMPUTATION

We consider concurrent programs of the form $P = P_1 \| \cdots \| P_K$ that consist of a finite number of fixed sequential processes $P_1, \ldots, P_K$ running in parallel. With every process $P_i$, we associate a single, unique index, namely $i$. We then say that two processes are *similar* if and only if one can be obtained from the other by swapping their indices (this is formalized in Section 6.2). Intuitively, this corresponds to concurrent algorithms where a single "generic" indexed piece of code gives the code body for all processes.

We observe that for most actual concurrent programs the portions of each process responsible for interprocess synchronization can be cleanly separated from the sequential applications-oriented computations performed by the process. This suggests that we focus our attention on *synchronization skeletons* which are abstractions of actual concurrent programs where detail irrelevant to synchronization is suppressed.

We may view the synchronization skeleton of an individual process $P_i$ as a state-machine where each state represents a region of code intended to perform some sequential computation and where each arc represents a conditional transition (between different regions of sequential code) used to enforce synchronization constraints. For example, there may be a node labeled $C_i$ representing the critical section of process $P_i$. While in $C_i$, the process $P_i$ may simply increment a single variable, or it may perform an extensive series of updates on a large database. In general, the internal structure and intended application of the regions of sequential code in an actual concurrent program are unspecified in the synchronization skeleton. By virtue of the abstraction to synchronization skeletons, we thus eliminate all steps of the sequential computation from consideration.

Formally, the synchronization skeleton of each process $P_i$ is a directed graph where each node is labeled by a unique name ($s_i$) which represents a *local state* of $P_i$, and where each arc is labeled with a synchronization command $B \to A$ consisting of an enabling condition (i.e., guard) $B$ and corresponding action $A$ to be performed (i.e., a guarded command [Dijkstra 1976]). Self-loops, where there is an arc from a node to itself, are disallowed. Also, each node must have at least one outgoing arc, i.e., a skeleton contains no "dead ends." A *global state* is a tuple of the form $(s_1, \ldots, s_K, v_1, \ldots, v_m)$ where each node $s_i$ is the current local state of $P_i$ and where $v_1, \ldots, v_m$ is a list giving the current values of the shared variables $x_1, \ldots, x_m$ (we assume that these are ordered in some fixed way, so that $v_1, \ldots, v_m$ specifies a unique value for each shared variable). A guard $B$ is a predicate on states, and an action $A$ is a parallel assignment statement that updates the values of the shared variables. If the guard $B$ is omitted from a command, it is interpreted as *true*, and we simply write the command as $A$. If the action $A$ is omitted, the shared variables are unaltered, and we write the command as $B$. For example, Figure 1 displays the

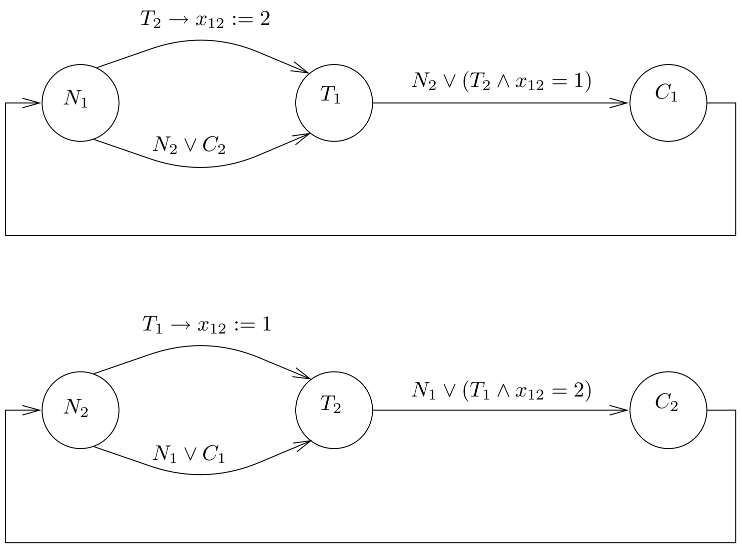



Fig. 1. Two-process mutual exclusion example.

synchronization skeletons (taken from Emerson and Clarke [1982]) synthesized by the synthesis method of Emerson and Clarke for the two-process mutual exclusion problem.

We model parallelism in the usual way by the nondeterministic interleaving of the "atomic" transitions of the individual synchronization skeletons of the processes $P_i$. Hence, at each step of the computation, some process with an "enabled" arc is nondeterministically selected to be executed next. Assume that the current state is $s = (s_1, \ldots, s_i, \ldots, s_K, v_1, \ldots, v_m)$ and that process $P_i$ contains an arc from node $s_i$ to $s'_i$ labeled by the command $B \rightarrow A$. If $B$ is true in the current state then a permissible next state is $(s_1, \ldots, s'_i, \ldots, s_K, v'_1, \ldots, v'_m)$ where $v'_1, \ldots, v'_m$ is the list of updated values for the shared variables resulting from action $A$. The arc from node $s_i$ to $s'_i$ is said to be *enabled* in state $s$. An arc that is not enabled is *disabled*, or *blocked*. A *computation path* is any sequence of states where each successive pair of states is related by the above next-state relation.

## 3. THE SPECIFICATION LANGUAGE MPCTL*

Our specification language is an extension of the temporal logic CTL* (Computation Tree Logic* [Emerson 1990]). CTL* is a propositional branching-time temporal logic. The basic modalities of CTL* consist of a path quantifier, either $A$ (for all paths) or $E$ (for some path) followed by a linear-time formula, which is built up from atomic propositions, the boolean operators $\wedge, \vee, \neg$, and the linear-time modalities $G$ (always), $F$ (sometime), $X_j$ (strong nexttime), $Y_j$ (weak nexttime) and $U$ (until). CTL* formulae are built up from atomic propositions, the boolean operators $\wedge, \vee, \neg$, and the basic modalities. We remark that since synchronization skeletons are in general finite state, the propositional version of temporal logic can be used to specify their properties.

### 3.1 CTL*

We have the following syntax for CTL*. We inductively define a class of state formulae (true or false of states) using rules (S1)–(S3) below and a class of path formulae (true or false of paths) using rules (P1)–(P3) below:

(S1) Each atomic proposition $p$ is a state formula;

(S2) if $f, g$ are state formulae, then so are $f \wedge g$, $\neg f$ (where $\wedge$, $\neg$ indicate conjunction and negation, respectively); and

(S3) if $f$ is a path formula, then $Ef$ and $Af$ are state formulae.

(P1) Each state formula is also a path formula;

(P2) if $f, g$ are path formulae, then so are $f \wedge g$, $\neg f$; and

(P3) if $f, g$ are path formulae, then so are $X_j f$, $fUg$.

$Ef$ is a formula which intuitively means that there is some maximal path for which $f$ holds, and $Af$ is a formula which intuitively means that $f$ holds of every maximal path. $X_j f$ (process indexed strong nexttime) is a formula which intuitively means that the immediate successor state along the maximal path under consideration is reached by executing one step of process $P_j$, and formula $f$ holds in the immediate successor state.[1] $fUg$ is a formula which intuitively means that there is some state along the maximal path under consideration where $g$ holds, and $f$ holds at every state along this path up to, but not necessarily including, that state.

CTL* consists of the set of state formulae generated by the above rules. The linear-time temporal logic PTL [Manna and Wolper 1984] consists of the set of path formulae generated by rules (S1) and (P1)–(P3). The logic CTL [Emerson 1990; Emerson and Clarke 1982] is obtained by replacing rules (P1)–(P3) by (P0):

(P0) If $f, g$ are state formulae, then $X_j f$, $fUg$ are path formulae.

The set of state formulae generated by rules (S1)–(S3) and (P0) forms CTL.

Formally, we define the semantics of CTL* formulae with respect to a ($K$-process) structure $M = (S, R_{i_1}, \ldots, R_{i_K})$ consisting of

—$S$, a countable set of states. Each state is a mapping from the set of atomic propositions into $\{true, false\}$ and

—$R_i \subseteq S \times S$, a binary relation on $S$ giving the transitions of sequential process $i$.

Let $R = R_{i_1} \cup \cdots \cup R_{i_K}$. A *path* is a sequence of states $(s_1, s_2 \ldots)$ such that $\forall i, (s_i, s_{i+1}) \in R$, and a *fullpath* is a maximal path. A fullpath $(s_1, s_2, \ldots)$ is infinite unless for some $s_k$ there is no $s_{k+1}$ such that $(s_k, s_{k+1}) \in R$. We use the convention (1) that $\pi = (s_1, s_2, \ldots)$ denotes a fullpath and (2) that $\pi^i$ denotes the suffix $(s_i, s_{i+1}, s_{i+2}, \ldots)$ of $\pi$, provided $i \leq |\pi|$, where $|\pi|$, the length of $\pi$, is $\omega$ when $\pi$ is infinite and $k$ when $\pi$ is finite and of the form $(s_1, \ldots, s_k)$; otherwise $\pi^i$ is undefined. We also use the usual notation to indicate truth in a structure:

[1]Note that we follow the presentation in Emerson and Clarke [1982], which introduced the process indexed nexttime operator, rather than the more conventional presentation of Emerson [1990], where the nexttime operator is unindexed.

$M, s_1 \models f$ (respectively $M, \pi \models f$) means that $f$ is true in structure $M$ at state $s_1$ (respectively of fullpath $\pi$). In addition, we use $M, S \models f$ to mean $\forall s \in S \,.\, (M, s \models f)$, where $S$ is a set of states. We define $\models$ inductively:

(S1) $M, s_1 \models p$ iff $s_1(p) = true$

(S2) $M, s_1 \models f \wedge g$ iff $M, s_1 \models f$ and $M, s_1 \models g$
$M, s_1 \models \neg f$ iff not$(M, s_1 \models f)$

(S3) $M, s_1 \models Ef$ iff there exists a fullpath $\pi = (s_1, s_2, \ldots)$ in $M$ such that $M, \pi \models f$
$M, s_1 \models Af$ iff for every fullpath $\pi = (s_1, s_2, \ldots)$ in $M$: $M, \pi \models f$

(P1) $M, \pi \models f$ iff $M, s_1 \models f$

(P2) $M, \pi \models f \wedge g$ iff $M, \pi \models f$ and $M, \pi \models g$
$M, \pi \models \neg f$ iff not$(M, \pi \models f)$

(P3) $M, \pi \models X_j f$ iff $\pi^2$ is defined and $(s_1, s_2) \in R_j$ and $M, \pi^2 \models f$
$M, \pi \models fUg$ iff there exists $i \in [1 : |\pi|]$ such that $M, \pi^i \models g$ and for all $j \in [1 : (i-1)]$: $M, \pi^j \models f$

We write $\models f$ to indicate that $f$ is *valid*, i.e., true at all states in all structures. When the structure $M$ does not affect the truth of the formula, (e.g., $M, s_1 \models p$ depends only on $s_1$ and $p$), it may be omitted (e.g., $M, s_1 \models p$ is written as $s_1 \models p$).

We introduce the abbreviations $f \vee g$ for $\neg(\neg f \wedge \neg g)$, $f \Rightarrow g$ for $\neg f \vee g$, and $f \equiv g$ for $(f \Rightarrow g) \wedge (g \Rightarrow f)$, which denote logical disjunction, implication, and equivalence, respectively. We also introduce a number of additional modalities as abbreviations: $Y_j f$ for $\neg X_j \neg f$, $Ff$ for $[trueUf]$, $Gf$ for $\neg F \neg f$, $fU_w g$ for $[fUg] \vee Gf$, $\overset{\infty}{F} f$ for $GFf$, and $\overset{\infty}{G} f$ for $FGf$. $Y_j$ is the "process indexed weak nexttime" modality. $Y_j f$ is a formula which intuitively means that if the immediate successor state along the maximal path exists, and is reached by executing one step of process $P_j$, then formula $f$ holds in the immediate successor state. $F$ is the "eventually" modality. $Ff$ is a formula which intuitively means that there is some state along the maximal path where $f$ holds. $G$ is the "always" modality. $Gf$ is a formula which intuitively means that $f$ holds at every state along the maximal path. $U_w$ is the "weak until" (or "unless") modality. $fU_w g$ is a formula which intuitively means that either $g$ holds at some state along the maximal path, and $f$ holds at every state along the maximal path up to (but not necessarily including) that state; or $g$ never holds along the maximal path, and $f$ holds at all states of the maximal path. Finally, $\overset{\infty}{F}$ and $\overset{\infty}{G}$ are the infinitary modalities "infinitely often" and "almost everywhere," respectively. $\overset{\infty}{F} f$ and $\overset{\infty}{G} f$ are formulae which intuitively mean respectively that $f$ holds in infinitely many states along the maximal path and that $f$ holds in all but a finite number of states along the maximal path. Particularly useful modalities are $AFf$, which means that for every maximal path there exists a state on the path where $f$ holds, and $AGf$, which means that $f$ holds at every state along every path. Note that the unindexed nexttime operators can be viewed as abbreviations:

$$Xf \equiv (X_{i_1} f \vee \ldots \vee X_{i_K} f)$$
$$Yf \equiv (Y_{i_1} f \wedge \ldots \wedge Y_{i_K} f)$$

### 3.2 Sublogics of CTL*

In the sequel, we define various sublogics of CTL* and PTL. We denote a sublogic by $\mathcal{L}$ followed by a list (enclosed in parentheses) of all the

—sets of atomic propositions,

—boolean operators, and

—temporal modalities

that can be used in constructing formulae of the sublogic. For example, $\mathcal{L}(\{Q_1, R_1\}, \{Q_2, R_2\}, \neg, \wedge, U)$ denotes the sublogic of PTL consisting of all those formulae which contain only the atomic propositions $Q_1, Q_2, R_1, R_2$ and which contain no next-time $(X, Y)$ modalities, while $\mathcal{L}(\wedge, \vee, AG(q_i \Rightarrow AFr_i), AG(q_i \Rightarrow AXr_i))$ is the sublogic of CTL* consisting of positive boolean combinations of formulae of the form $AG(q_i \Rightarrow AFr_i)$ or $AG(q_i \Rightarrow AXr_i)$.

### 3.3 The Interconnection Relation

As stated in the introduction, we present a method of synthesizing a $K$-process system from a pair-system. We also stated that our method can deal with arbitrary process interconnection schemes. The desired process interconnection scheme is given by the *interconnection relation* $I$, which is, therefore, part of the problem specification. The domain of $I$ is the set $\{i_1, \ldots, i_K\}$ of indices of the processes in the $K$-process system being specified. Thus, $I \subseteq \{i_1, \ldots, i_K\} \times \{i_1, \ldots, i_K\}$, and $i\,I\,j$ iff processes $i$ and $j$ are interconnected. In the sequel, we say that $i$ and $j$ are *neighbors* when $i\,I\,j$. Since our definition of interconnection (Section 1) is symmetric, we stipulate that $I$ is symmetric. We also stipulate that $I$ is irreflexive, since a process cannot be interconnected to itself. Furthermore, we assume that $I$ is a total relation: $\bigwedge i \in \{i_1, \ldots, i_K\} \,.\, (\bigvee j \,.\, (i\,I\,j))$. Thus every process is interconnected to at least one other process.

Since there are many possible interconnection schemes for a given number $K$ of processes, we henceforth use the more precise term $I$*-system* in place of the (up till now used) term "$K$-process system." We also introduce the following abbreviations. $dom(I)$ denotes the domain of $I$ (i.e., the set $\{i_1, \ldots, i_K\}$); $I(i)$ denotes the set $\{j \mid i\,I\,j\}$; and $\hat{I}(i)$ denotes the set $\{i\} \cup \{j \mid i\,I\,j\}$.

### 3.4 MPCTL*

An MPCTL* (Many-Process CTL*) formula consists of a spatial modality followed by a CTL* state formula over a "uniformly" indexed family $\mathcal{AP} = \{\mathcal{AP}_{i_1}, \ldots, \mathcal{AP}_{i_K}\}$ of atomic propositions. The propositions in $\mathcal{AP}_i$ are the same as those in $\mathcal{AP}_j$ except for their subscripts; for example, we could have $\mathcal{AP}_1 = \{Q_1, R_1\}$ and $\mathcal{AP}_2 = \{Q_2, R_2\}$. $Q_1$ corresponds to $Q_2$, $R_1$ to $R_2$, etc. A spatial modality is of the form $\bigwedge_i$, or $\bigwedge_{ij}$. $\bigwedge_i$ quantifies the process index $i$ which ranges over $\{i_1, \ldots, i_K\}$. $\bigwedge_{ij}$ quantifies the process indices $i, j$ which range over the elements of $\{i_1, \ldots, i_K\}$ which are related by $I$. If the spatial modality is $\bigwedge_i$, then only atomic propositions in $\mathcal{AP}_i$ are allowed in the CTL* formula, and if the spatial modality is $\bigwedge_{ij}$, then only atomic propositions in $\mathcal{AP}_i \cup \mathcal{AP}_j$ are allowed in the CTL* formula. In other words, every process index within an MPCTL* formula must be "bound" by the spatial modality.

Thus mutual exclusion over all pairs of processes that are related by $I$ is expressed by $\bigwedge_{ij} AG(\neg(C_i \wedge C_j))$, (no two processes related by $I$ are in their critical regions simultaneously), while absence of starvation for each process is expressed by $\bigwedge_i AG(T_i \Rightarrow AFC_i)$ (every process which enters its trying region will eventually enter its critical region). The semantics of these assertions over a structure depends on the interconnection relation $I$ associated with the structure. For example, the semantics of the mutual exclusion specification over a structure is given by viewing the spatial modalities as abbreviations and expanding them in the obvious way: $AG(\neg(C_1 \wedge C_2))$ for a two-process structure, with $I = \{\{1,2\}\}$, and $AG(\neg(C_1 \wedge C_2)) \wedge AG(\neg(C_1 \wedge C_3)) \wedge AG(\neg(C_2 \wedge C_3))$ for a three-process structure, with $I = \{\{1,2\},\{1,3\},\{2,3\}\}$, and so forth. Similarly, the semantics of the absence of starvation assertion is given by $AG(T_1 \Rightarrow AFC_1) \wedge AG(T_2 \Rightarrow AFC_2)$ over a two-process structure with $I = \{\{1,2\}\}$, and by $AG(T_1 \Rightarrow AFC_1) \wedge AG(T_2 \Rightarrow AFC_2) \wedge AG(T_3 \Rightarrow AFC_3)$ over a three-process structure with $I = \{\{1,2\},\{1,3\},\{2,3\}\}$.

In general, an MPCTL* formula $q$ is interpreted over a $K$-process structure $M = (S, R_{i_1}, \ldots, R_{i_K})$ where $S$ and $R_{i_1}, \ldots, R_{i_K}$ are as before. We say that $K$ is the *arity* of the structure $M$. The definition of truth in structure $M$ at state $s$ of formula $q$ is given by $M, s \models q$ iff $M, s \models q'$ where $q'$ is the CTL* formula obtained from $q$ by viewing $q$ as an abbreviation and expanding it as shown above, i.e.,

—$M, s \models \bigwedge_i f_i$ iff $M, s \models \bigwedge i \in \{i_1, \ldots, i_K\} \,.\, (f_i)$
—$M, s \models \bigwedge_{ij} f_{ij}$ iff $M, s \models \bigwedge (i,j) \in I \,.\, (f_{ij})$

where $f_i$ is a CTL* formula containing atomic propositions from $\mathcal{AP}_i$ only, and $f_{ij}$ is a CTL* formula containing atomic propositions from $\mathcal{AP}_i \cup \mathcal{AP}_j$ only. Note that we use $\bigwedge v \in R \,.\, (f)$, $\bigvee v \in R \,.\, (f)$, to denote $f(v_1) \wedge \ldots \wedge f(v_n)$, $f(v_1) \vee \ldots \vee f(v_n)$ respectively, where $R = \{v_1, \ldots, v_n\}$. Sometimes we will also use $\bigwedge_{v \in R} f$, $\bigvee_{v \in R} f$. We also use "$\bigwedge (i,j) \in I$" to denote "$\bigwedge i, j \in dom(I)$ such that $i \, I \, j$." In MPCTL*, $\bigwedge_{ij} f_i$ is permitted, and is equivalent to $\bigwedge_i f_i$. MPCTL* is similar to the Indexed CTL* of Clarke et al. [1986], but allows quantification over pairs of processes (via the quantifier $\bigwedge_{ij}$) as well as over single processes. Furthermore, if the CTL* subformula of the MPCTL* formula is restricted to be in CTL [Emerson 1990; Emerson and Clarke 1982], then the resulting logic is called MPCTL.

An example of an MPCTL* specification is given below for the mutual exclusion problem. In this specification, process $P_i$ is always in exactly one of three states $N_i$, $T_i$, $C_i$, which represent, respectively, the Noncritical, Trying, and Critical code regions. A process in its Noncritical state does not require the use of critical (shared) resources, and it performs computations that can proceed in parallel with computations by the other processes. A process requiring a critical resource moves into the Trying state. From there, it enters the Critical state as soon as possible, provided that all constraints on the use of the critical resource are met. In the Critical state, $P_i$ has access to the critical resource. This specification is given by the following set of MPCTL* formulae (which are implicitly conjoined):

(1) initial state (every process is initially in its noncritical region):

$$\bigwedge_i N_i$$

(2) it is always the case that any move $P_i$ makes from its noncritical region is into

its trying region, and such a move is always possible:

$$\bigwedge_i AG(N_i \Rightarrow (AY_iT_i \wedge EX_iT_i))$$

(3) it is always the case that any move $P_i$ makes from its trying region is into its critical region:

$$\bigwedge_i AG(T_i \Rightarrow (AY_iC_i))$$

(4) it is always the case that any move $P_i$ makes from its critical region is into its noncritical region, and such a move is always possible:

$$\bigwedge_i AG(C_i \Rightarrow (AY_iN_i \wedge EX_iN_i))$$

(5) $P_i$ is always in exactly one of the states $N_i$, $T_i$, or $C_i$:

$$\bigwedge_i AG(N_i \equiv \neg(T_i \vee C_i))$$
$$\bigwedge_i AG(T_i \equiv \neg(N_i \vee C_i))$$
$$\bigwedge_i AG(C_i \equiv \neg(N_i \vee T_i))$$

(6) $P_i$ does not starve:

$$\bigwedge_i AG(T_i \Rightarrow AFC_i)$$

(7) a transition by one process cannot cause a transition by another:

$$\bigwedge_{ij} AG((N_i \Rightarrow AY_jN_i) \wedge (N_j \Rightarrow AY_iN_j))$$
$$\bigwedge_{ij} AG((T_i \Rightarrow AY_jT_i) \wedge (T_j \Rightarrow AY_iT_j))$$
$$\bigwedge_{ij} AG((C_i \Rightarrow AY_jC_i) \wedge (C_j \Rightarrow AY_iC_j))$$

(8) no two processes access critical resources together:

$$\bigwedge_{ij} AG(\neg(C_i \wedge C_j))$$

If $I = dom(I) \times dom(I) - \{(i,i) \mid i \in dom(I)\}$, i.e., all pairs of processes are $I$-related, then the specification is for the $K$-process mutual exclusion problem, where at most one process is in its critical section at any time. On the other hand, by using an appropriate $I$, we can also specify the following problems in MPCTL$^*$:

Five dining philosophers arranged in a ring:

$$I = \{\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{5,1\}\}.$$

One thousand dining philosophers arranged in a ring:

$$I = \{\{1,2\},\{2,3\},\ldots,\{999,1000\},\{1000,1\}\}.$$

Readers-Writers (with $P_1, P_2$ as readers, $P_3, P_4$ as writers, and no writer priority):

$$I = \{\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\}\}.$$

These examples generalize the mutual exclusion example: in dining philosophers, only neighboring ($I$-related) philosophers are mutually excluded from simultaneous access to their critical sections, and in readers-writers, only reader/writer and writer/writer pairs are mutually excluded.

## 4. NOTATION AND SYNTAX

In the sequel, unless otherwise indicated (e.g., by explicit quantification) $i$ and $j$ will denote arbitrary elements of $dom(I)$ such that $i\,I\,j$. Consider the pair-system consisting of processes $i$ and $j$, which execute concurrently and interact by reading each other's local state and by reading/writing shared variables (when we need to specify the component processes, we call this pair-system the *ij-system*). We call (the synchronization skeletons for)[2] processes $i, j$ in this pair-system $P_i^j, P_j^i$, respectively. In the sequel, we call a process of a pair-system a *pair-process*. As elaborated in Section 2, (a synchronization skeleton for) a process is a directed graph, where the nodes are local states of the process, and the arcs between nodes are labeled with synchronization commands, whose execution is required for the arc to be "traversed" by the process. These synchronization commands test the local state of other processes, and they test/modify shared variables. Let $\mathcal{SH}_{ij}$ be the set of all the variables shared by $P_i^j$ and $P_j^i$. We restrict $P_i^j, P_j^i$ to the following syntax. Each shared variable $x_{ij}$ assumes a value in $\{i, j\}$, i.e., all shared variables are binary. Shared variables may optionally be superscripted, which takes the place of the usual "subscript" notation, since we use subscripts here to denote the relevant pair-system. The guards of the synchronization commands that label arcs in $P_i^j$ are built up from the standard propositional logic connectives, the atomic propositions in $\mathcal{AP}_j$,[3] and expressions of the form $x_{ij} = i$ or of the form $x_{ij} = j$, where $x_{ij}$ is in $\mathcal{SH}_{ij}$. The actions of the synchronization commands that label arcs in $P_i^j$ are multiple assignments of the form $x_{ij}, y_{ij}, z_{ij}, \ldots := h, k, \ell, \ldots$ where the left-hand side is a list of variables in $\mathcal{SH}_{ij}$, and the right-hand side is an equally long list of constants, all of whom have value either $i$ or $j$. Each node of $P_i^j$ assigns either $true$ or $false$ to each atomic proposition in $\mathcal{AP}_i$ (i.e., is a mapping of $\mathcal{AP}_i$ into $\{true, false\}$). We refer to the nodes of $P_i^j$ as ***i*-states** (i.e., local states of $P_i^j$). For example, in Figure 1, the lower $N_1$-to-$T_1$ transition has a label with guard $N_2 \vee C_2$ and assignment *skip*, and the upper $N_1$-to-$T_1$ transition has a label with guard $T_2$ and assignment $x_{12} := 2$. The $T_1$-to-$C_1$ transition has a label whose guard is $N_2 \vee (T_2 \wedge x_{12} = 1)$ and whose assignment is *skip*. The node $N_1$ maps $N_1$ to $true$ and both $T_1$ and $C_1$ to $false$. Likewise, the node $T_1$ maps $T_1$ to $true$ and both $N_1$ and $C_1$ to $false$, and the node $C_1$ maps $C_1$ to $true$ and both $N_1$ and $T_1$ to $false$. In the sequel, we use $S_i$ to denote the set of $i$-states in $P_i^j$, and we use $(s_i, B \to A, t_i)$ to denote an arc from $i$-state $s_i$ to $i$-state $t_i$ with label $B \to A$.

Since we adopt an interleaving model of parallel computation, we need to define the global state of the pair-system $P_i^j \| P_j^i$. A global state of this pair-system is a tuple $(s_i, s_j, v_{ij}^1, \ldots, v_{ij}^m)$ where $s_i, s_j$ are the current $i$-state, $j$-state, respectively, and $v_{ij}^1, \ldots, v_{ij}^m$ give the values of all the variables in $\mathcal{SH}_{ij}$. We refer to global states of $P_i^j \| P_j^i$ as ***ij*-states**. When viewed as a mapping of $\mathcal{AP}_i \cup \mathcal{AP}_j$ into

[2] In our framework, there is no technical distinction between a process and a synchronization skeleton; a process is a synchronization skeleton. We therefore, for the most part, use the term "process" from now on instead of "skeleton," but will sometimes use the term skeleton when we wish to emphasize the syntactic aspects of a process.

[3] We remind the reader that the atomic propositions in $\mathcal{AP}_j$ can only be written by $P_j^i$, but may be read by $P_i^j$; see Section 3.4.

$\{true, false\}$ (as required by Section 3.1 to evaluate CTL* formulae), an $ij$-state inherits the assignments defined by its component $i$- and $j$-states: $s_{ij}(Q_i) = s_i(Q_i)$, $s_{ij}(Q_j) = s_j(Q_j)$, where $s_{ij} = (s_i, s_j, v^1_{ij}, \ldots, v^m_{ij})$, and $Q_i, Q_j$ are arbitrary atomic propositions in $\mathcal{AP}_i$, $\mathcal{AP}_j$, respectively. In addition to the skeletons for $P^j_i, P^i_j$, a complete description of the pair-system $P^j_i \| P^i_j$ requires that the initial states be given. We therefore include a set $S^0_{ij}$ of $ij$-states which gives the permissible initial states, and our complete notation for the pair-system consisting of processes $i$ and $j$ is $(S^0_{ij}, P^j_i \| P^i_j)$.

We now turn to $I$-systems. For a given interconnection relation $I$, the $I$-system consists of processes $i_1, \ldots, i_K$ executing concurrently, where $\{i_1, \ldots, i_K\}$ is the domain of $I$. We call the synchronization skeleton for process $i$ ($i \in \{i_1, \ldots, i_K\}$) in the $I$-system $P^I_i$, and we call a process of the $I$-system an *$I$-process*. As in the case for pair-systems, the nodes of such processes are $i$-states, i.e., mappings of $\mathcal{AP}_i$ into $\{true, false\}$. A global state of the $I$-system is a tuple $(s_{i_1}, \ldots, s_{i_K}, v^1, \ldots, v^n)$, where $s_i$, ($i \in \{i_1, \ldots, i_K\}$) is the current local state of $P^I_i$, and $v^1, \ldots, v^n$ give the values of all the shared variables of the $I$-system (once again we assume some fixed ordering of these variables, so that the values assigned to them are uniquely determined by the list $v^1, \ldots, v^n$). We refer to global states of the $I$-system as **$I$-states**. When viewed as a mapping of $\mathcal{AP}_{i_1} \cup \cdots \cup \mathcal{AP}_{i_K}$ into $\{true, false\}$ (as required by Section 3.1 to evaluate CTL* formulae), an $I$-state inherits the assignments defined by its component $i$-states ($i \in \{i_1, \ldots, i_K\}$): $s_{ij}(Q_i) = s_i(Q_i)$, where $s = (s_{i_1}, \ldots, s_{i_K}, v^1, \ldots, v^n)$, and $Q_i$ is an arbitrary atomic proposition in $\mathcal{AP}_i$ ($i \in \{i_1, \ldots, i_K\}$). We usually use $s_I, t_I, u_I$ to denote $I$-states. When the interconnection relation $I$ is clear from context, it may be omitted, e.g., $s_I, P^I_i$ are rendered as $s, P_i$, respectively. We use $S^0_I$ to denote the set of permissible initial $I$-states and $\mathcal{SH}$ to denote the set of shared variables of the $I$-system. The $I$-system overall is denoted as $(S^0_I, P^I_{i_1} \| \ldots \| P^I_{i_K})$. The next section shows how $(S^0_I, P^I_{i_1} \| \ldots \| P^I_{i_K})$ is (uniquely) determined from $(S^0_{ij}, P^j_i \| P^i_j)$ and $I$.

Let $s_i$ be an $i$-state. We define a state-to-formula operator $\{\!|s_i|\!\}$ that takes an $i$-state $s_i$ as an argument and returns a propositional formula that characterizes $s_i$ in that $s_i \models \{\!|s_i|\!\}$, and $s'_i \not\models \{\!|s_i|\!\}$ for all $i$-states $s'_i$ such that $s'_i \neq s_i$: $\{\!|s_i|\!\} = (\bigwedge_{s_i(Q_i)=true} Q_i) \wedge (\bigwedge_{s_i(Q_i)=false} \neg Q_i)$, where $Q_i$ ranges over the members of $\mathcal{AP}_i$.

## 5. DESCRIPTION OF THE METHOD

### 5.1 Synthesis of Many-Process Systems from Pair-Systems

As stated earlier, our aim is to synthesize an $I$-system without explicitly generating its global state transition diagram (and thereby incurring a time and space complexity exponential in $|dom(I)|$). We achieve this by breaking the problem down into two steps:

(1) Synthesize a pair-system (from the given temporal logic specification)
(2) Generalize the pair-system to an $I$-system

Since our focus in this article is on exploiting the similarity of processes, we shall not address the first step explicitly. We note in passing that any method for the derivation of concurrent programs can be used to generate the pair-system. In particular, if the problem specification can be written in MPCTL, then we can extract a two-process specification (say for processes $i$ and $j$) by expanding the MPCTL specification (as shown in Section 3) for $I = \{\{i, j\}\}$. We then apply the synthesis method of Emerson and Clarke [1982] to the two-process specification, resulting in the pair-system. For example, the specification for $K$-process mutual exclusion given in Section 3.4 is actually in MPCTL, and so the pair-system for mutual exclusion (Figure 1) can be generated automatically from this specification. Since a pair-system has only $O(N^2)$ states (where $N$ is the size of each sequential process), the problem of deriving a pair-system from a specification is considerably easier than that of deriving an $I$-system from the specification. Hence, the contribution of this article, namely the second step above, is to reduce the more difficult problem (deriving the $I$-system) to the easier problem (deriving the pair-system). We proceed as follows.

Consider the pair-system $(S^0_{ij}, P^j_i \| P^i_j)$. We take this pair-system and generalize it in a natural way to an $I$-system. We show that our generalization preserves a large class of correctness properties. Roughly the idea is as follows. Consider first the generalization to three pairwise interconnected processes $i, j, k$, i.e., $I = \{\{i, j\}, \{j, k\}, \{k, i\}\}$. With respect to process $i$, the proper interaction (i.e., the interaction required to satisfy the specification) between process $i$ and process $j$ is captured by the synchronization commands that label the arcs of $P^j_i$. Likewise, the proper interaction between process $i$ and process $k$ is captured by the arc labels of $P^k_i$. Therefore, in the three-process system consisting of processes $i, j, k$ executing concurrently, (and where process $i$ is interconnected to both process $j$ and process $k$), the proper interaction for process $i$ with processes $j$ and $k$ is captured as follows: when process $i$ traverses an arc, the synchronization command which labels that arc in $P^j_i$ is executed "simultaneously" with the synchronization command which labels the corresponding arc in $P^k_i$. For example, taking as our specification the mutual exclusion problem, if $P_i$ executes the mutual exclusion protocol with respect to both $P_j$ and $P_k$, then, when $P_i$ enters its critical section, both $P_j$ and $P_k$ must be outside their own critical sections.

Based on the above reasoning, we determine that the synchronization skeleton for process $i$ in the aforementioned three-process system (call it $P^{jk}_i$) has the same basic graph structure as $P^j_i$ and $P^k_i$, and a arc label in $P^{jk}_i$ is a "composition" of the labels of the corresponding arcs in $P^j_i$ and $P^k_i$. In addition, the initial states $S^0_{ijk}$ of the three-process system are exactly those states that "project" onto initial states of all three pair-systems ($(S^0_{ij}, P^j_i \| P^i_j)$, $(S^0_{ik}, P^k_i \| P^i_k)$, and $(S^0_{jk}, P^k_j \| P^j_k)$). More formally, $s^0_{ijk} \in S^0_{ijk}$ if and only if $s^0_{ijk} = (s_i, s_j, s_k, v^1_{ij}, \ldots, v^m_{ij}, v^1_{ik}, \ldots, v^m_{ik}, v^1_{jk}, \ldots, v^m_{jk})$, where $(s_i, s_j, v^1_{ij}, \ldots, v^m_{ij}) \in S^0_{ij}$, $(s_i, s_k, v^1_{ik}, \ldots, v^m_{ik}) \in S^0_{ik}$, and $(s_j, s_k, v^1_{jk}, \ldots, v^m_{jk}) \in S^0_{jk}$.

Generalizing this to the case of an arbitrary interconnection relation $I$, we see that the skeleton for process $i$ in the $I$-system, $P^I_i$, has the same basic graph structure as $P^j_i$, and a transition label in $P^I_i$ is a "composition" of the labels of the corresponding transitions in $P^{j_1}_i, \ldots, P^{j_n}_i$, where $\{j_1, \ldots, j_n\} = I(i)$, i.e., processes $j_1, \ldots, j_n$ are all the $I$-neighbors of process $i$. Likewise the set $S^0_I$ of initial states of the $I$-system is exactly those states all of whose "projections" onto all the pairs in $I$ give initial states of the corresponding pair-system.

This discussion leads to the following definition of the synthesis method, which shows how an $I$-process $P_i^I$ of the $I$-system $(S_I^0, P_{i_1}^I \| \ldots \| P_{i_K}^I)$ is derived from the pair-process $P_i^j$ of the pair-system $(S_{ij}^0, P_i^j \| P_j^i)$ and how the initial state set $S_I^0$ of $(S_I^0, P_{i_1}^I \| \ldots \| P_{i_K}^I)$ is derived from the initial state set $S_{ij}^0$ of $(S_{ij}^0, P_i^j \| P_j^i)$.

*Definition* 5.1.1 (*MP-Synthesis*). An $I$-*process* $P_i^I$ is derived from the pair-process $P_i^j$ as follows:

$P_i^I$ contains an arc from $s_i$ to $t_i$ with label $\bigwedge_{j \in I(i)} B_i^j \to //_{j \in I(i)} A_i^j$
iff
for every $j$ in $I(i)$: $P_i^j$ contains an arc from $s_i$ to $t_i$ with label $B_i^j \to A_i^j$.

The *initial state set* $S_I^0$ of the $I$-system is derived from the initial state set $S_{ij}^0$ of the pair-system as follows:

$$S_I^0 = \{(s_{i_1}, \ldots, s_{i_K}, v^1, \ldots, v^n) \mid \bigwedge (i,j) \in I \,.\, (s_i, s_j, v_{ij}^1, \ldots, v_{ij}^m) \in S_{ij}^0)\}$$

where $v_{ij}^1, \ldots, v_{ij}^m$ are those values from $v^1, \ldots, v^n$ that denote values of variables in $\mathcal{SH}_{ij}$.

Here $\bigwedge$ is logical conjunction, and $//$ is simultaneous parallel assignment, i.e., $//_{j \in I(i)} A_i^j$ is a single parallel assignment. Its execution is achieved by simultaneously executing every $A_i^j$ (for fixed $i$, and as $j$ varies over $I(i)$). From Definition 5.1.1, we see that the shared variables of $(S_I^0, P_{i_1}^I \| \ldots \| P_{i_K}^I)$ are shared *pairwise*, i.e., $\mathcal{SH}$ is partitioned into the sets $\mathcal{SH}_{ij}$ for all $I$-related pairs $i, j$: $\mathcal{SH} = \cup (i,j) \in I \,.\, \mathcal{SH}_{ij}$ and $\bigwedge i, j, k \in \mathit{dom}(I) \,.\, (\mathcal{SH}_{ij} \cap \mathcal{SH}_{ik} = \emptyset)$.

The MP-synthesis definition constructs the $I$-system as a "conjunctive composition" of the pair-systems for every pair in $I$. This composition operates as follows:

—For an $I$-process $P_i^I$ to traverse an arc from $s_i$ to $t_i$, every $ij$-system $(S_{ij}^0, P_i^j \| P_j^i)$ (for every $j$ in $I(i)$) must be in an $ij$-state $s_{ij}$ whose $P_i^j$-component is $s_i$, and each such $s_{ij}$ must have an outgoing transition to some $ij$-state $t_{ij}$ whose $P_i^j$-component is $t_i$

—When the $P_i^I$-transition from $s_i$ to $t_i$ occurs, the current $ij$-states of all the $ij$-systems are *simultaneously* changed from $s_{ij}$ to $t_{ij}$

This characterization of transitions in the $I$-system as compositions of transitions in all the relevant pair-systems is formalized in Lemma 6.4.1. The above definition of MP-synthesis is, in effect, a *syntactic transformation* that can be carried out in linear time and space (in both $(S_{ij}^0, P_i^j \| P_j^i)$ and $I$). In particular, we avoid explicitly constructing the global state transition diagram of $(S_I^0, P_{i_1}^I \| \ldots \| P_{i_K}^I)$, which is of size exponential in $K = |\mathit{dom}(I)|$. Figure 2 gives an example application of MP-synthesis: the synthesis of a solution to the three-process mutual exclusion problem from the solution to the two-process mutual exclusion problem given in Figure 1. One of the transitions of the three-process mutual exclusion system is $[N_1\, T_2\, C_3] \xrightarrow{1} [T_1\, x_{12} = 2\, T_2\, C_3]$ (it arises from the execution of the arc labeled $T_2 \wedge (N_3 \vee C_3) \to x_{12} := 2$ leading from 1-state $N_1$ to 1-state $T_1$). This transition can be regarded as a composition of the transitions $[N_1\, T_2] \xrightarrow{1} [T_1\, x_{12} = 2\, T_2]$ and $[N_1\, C_3] \xrightarrow{1} [T_1\, C_3]$. The first is a transition of the two-process mutual exclusion

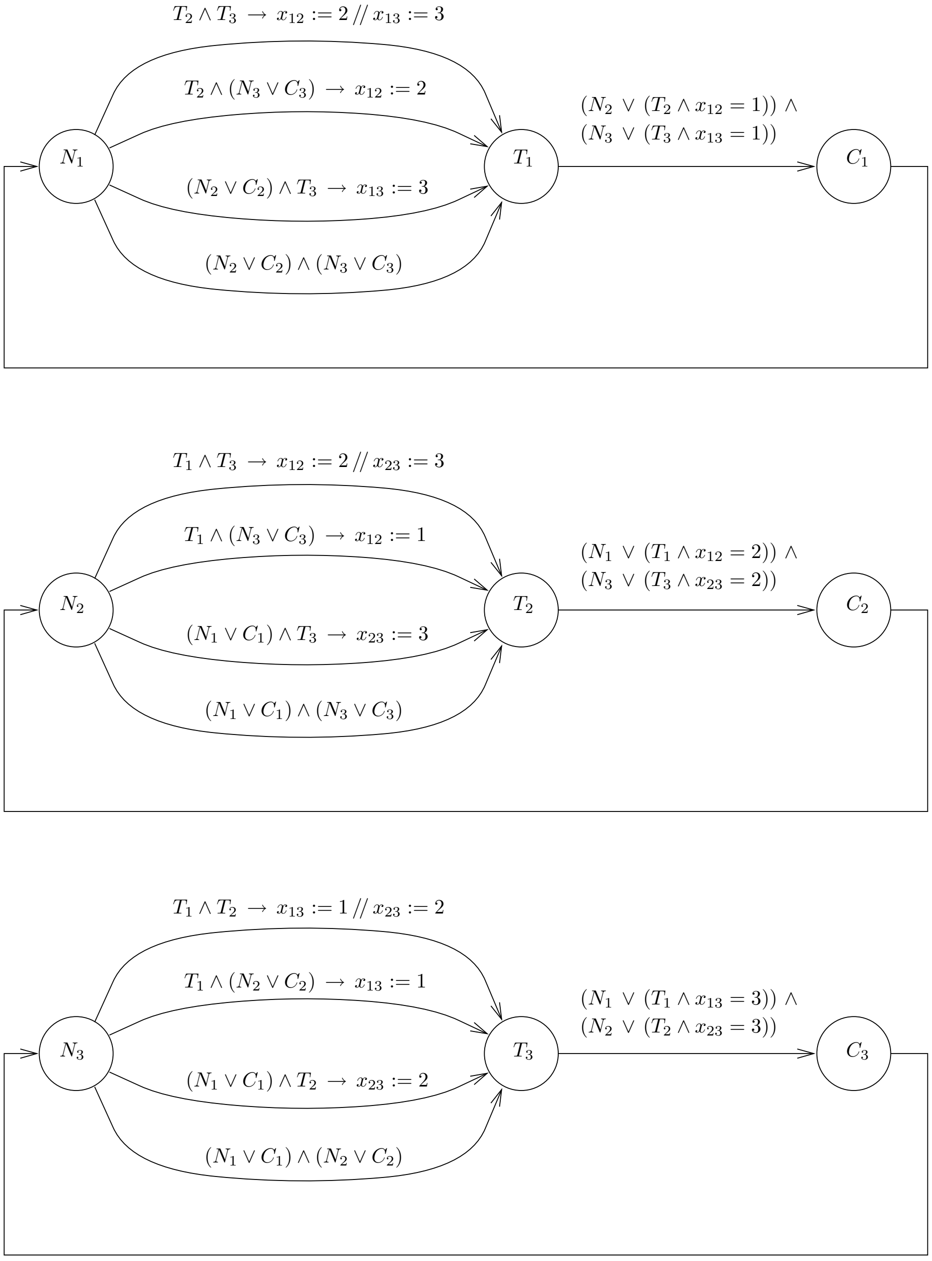



Fig. 2. Three-process mutual exclusion example.

system for processes 1 and 2, and the second is a transition of the two-process mutual exclusion system for processes 1 and 3.

5.1.1 *Subsystems of the $I$-System.* Let $J \subseteq I$, i.e., $J$ is an interconnection relation which is a subrelation of $I$. Thus, $J$ represents a subsystem (called the *$J$-system*) of the $I$-system in that it generally contains a subset of $I$'s processes, and these processes are related by a subset of the pair-constraints in $I$. The $J$-system is, in effect, the composition of the pair-systems for all the pairs in $J$. A global state of the $J$-system, called a *$J$-state*, is a tuple $(s_{j_1}, \ldots, s_{j_L}, v_J^1, \ldots, v_J^m)$, where $\{j_1, \ldots, j_L\}$ is the domain of $J$, and $v_J^1, \ldots, v_J^m$ is a list of values (in some fixed ordering) for the variables in $\bigcup_{(i,j) \in J} \mathcal{SH}_{ij}$.

5.1.2 *State and Path Projection.* Due to the compositional nature of the $I$-system outlined above, it will be very convenient in the sequel to be able to refer to the components of a given $I$-state or $ij$-state. For this purpose, we define the **state projection operator** $\upharpoonright$. This operator has several variants. First of all, we define projection onto a single process from both $I$-states and $ij$-states: if $s = (s_{i_1}, \ldots, s_{i_K}, v^1, \ldots, v^n)$, then $s{\upharpoonright}i = s_i$, and if $s_{ij} = (s_i, s_j, v_{ij}^1, \ldots, v_{ij}^m)$, then $s_{ij}{\upharpoonright}i = s_i$. This gives the $i$-state corresponding to the $I$-state $s$, $ij$-state $s_{ij}$, respectively. Next we define projection of an $I$-state onto a pair-system: if $s = (s_{i_1}, \ldots, s_{i_K}, v^1, \ldots, v^n)$, then $s{\upharpoonright}ij = (s_i, s_j, v_{ij}^1, \ldots, v_{ij}^m)$, where $v_{ij}^1, \ldots, v_{ij}^m$ are those values from $v^1, \ldots, v^n$ that denote values of variables in $\mathcal{SH}_{ij}$. This gives the $ij$-state corresponding to the $I$-state $s$, and is well defined only when $i\, I\, j$. We also define projection onto the shared variables in $\mathcal{SH}_{ij}$ from both $ij$-states and $I$-states: if $s_{ij} = (s_i, s_j, v_{ij}^1, \ldots, v_{ij}^m)$, then $s_{ij}{\upharpoonright}\mathcal{SH}_{ij} = (v_{ij}^1, \ldots, v_{ij}^m)$, and if $s = (s_{i_1}, \ldots, s_{i_K}, v^1, \ldots, v^n)$, then $s{\upharpoonright}\mathcal{SH}_{ij} = (v_{ij}^1, \ldots, v_{ij}^m)$, where $v_{ij}^1, \ldots, v_{ij}^m$ are those values from $v^1, \ldots, v^n$ that denote values of variables in $\mathcal{SH}_{ij}$. Finally, we define projection of an $I$-state onto a $J$-system. If $s = (s_{i_1}, \ldots, s_{i_K}, v^1, \ldots, v^n)$, then $s{\upharpoonright}J = (s_{j_1}, \ldots, s_{j_L}, v_J^1, \ldots, v_J^m)$, where $\{j_1, \ldots, j_L\}$ is the domain of $J$, and $v_J^1, \ldots, v_J^m$ are those values from $v^1, \ldots, v^n$ that denote values of variables in $\bigcup_{(i,j) \in J} \mathcal{SH}_{ij}$. This gives the $J$-state (defined analogously to an $I$-state) corresponding to the $I$-state $s$ and is well defined only when $J \subseteq I$.

To define projection for paths, we first extend the definition of path (and fullpath) to include the index of the process making the transition, e.g., each transition is labeled by an index denoting this process. For example, a path in $M_I$ would be represented as $s^1 \xrightarrow{d_1} s^2 \cdots s^n \xrightarrow{d_n} s^{n+1} \xrightarrow{d_{n+1}} s^{n+2} \cdots$, where $\bigwedge m \geq 1 \,.\, (d_m \in \mathit{dom}(I))$.

Let $\pi$ be an arbitrary path in $M_I$. For any $J$ such that $J \subseteq I$, define a *$J$-block* (cf. Clarke et al. [1986] and Browne et al. [1988]) of $\pi$ to be a maximal subsequence of $\pi$ that starts and ends in a state and does not contain a transition by any $P_i$ such that $i \in \mathit{dom}(J)$. Thus we can consider $\pi$ to be a sequence of $J$-blocks with successive $J$-blocks linked by a single $P_i$-transition such that $i \in \mathit{dom}(J)$ (note that a $J$-block can consist of a single state). It also follows that $s{\upharpoonright}J = t{\upharpoonright}J$ for any pair of states $s, t$ in the same $J$-block. This is because a transition that is not by some $P_i$ such that $i \in \mathit{dom}(J)$ cannot affect any atomic proposition in $\bigcup_{i \in \mathit{dom}(J)} \mathcal{AP}_i$, nor can it change the value of a variable in $\bigcup_{(i,j) \in J} \mathcal{SH}_{ij}$; and a $J$-block contains no such $P_i$ transition. Thus, if $B$ is a $J$-block, we define $B{\upharpoonright}J$ to be $s{\upharpoonright}J$ for some state $s$ in $B$. We now give the formal definition of path projection. We use the same notation ($\upharpoonright$) as for state projection. Let $B^n$ denote the $n$th $J$-block of $\pi$.

*Definition* 5.1.2.1 (*Path Projection*). Let $\pi$ be $B^1 \xrightarrow{d_1} \cdots B^n \xrightarrow{d_n} B^{n+1} \cdots$ where

$B^m$ is a $J$-block for all $m \geq 1$. Then the *Path Projection Operator* $\uparrow J$ is given by

$$\pi\uparrow J = B^1\uparrow J \xrightarrow{d_1} \cdots B^n\uparrow J \xrightarrow{d_n} B^{n+1}\uparrow J \cdots$$

Thus there is a one-to-one correspondence between $J$-blocks of $\pi$ and states of $\pi\uparrow J$, with the $n$th $J$-block of $\pi$ corresponding to the $n$th state of $\pi\uparrow J$ (note that path projection is well defined when $\pi$ is finite). For example, consider the three-process mutual exclusion system given in Figure 2. This is synthesized from the two-process mutual exclusion system given in Figure 1 using $I = \{\{1,2\},\{2,3\},\{3,1\}\}$. Consider the following path $\pi$ of this system (for notational clarity, we omit the values of the shared variables):

$$[N_1\,N_2\,N_3] \xrightarrow{1} [T_1\,N_2\,N_3] \xrightarrow{3} [T_1\,N_2\,T_3] \xrightarrow{1} [C_1\,N_2\,T_3] \xrightarrow{1}$$
$$[N_1\,N_2\,T_3] \xrightarrow{3} [N_1\,N_2\,C_3] \xrightarrow{3} [N_1\,N_2\,N_3]$$

Let $J = \{\{1,2\}\}$. Then the $J$-blocks of $\pi$ are $[N_1\,N_2\,N_3]$, $[T_1\,N_2\,N_3] \xrightarrow{3} [T_1\,N_2\,T_3]$, $[C_1\,N_2\,T_3]$, and $[N_1\,N_2\,T_3] \xrightarrow{3} [N_1\,N_2\,C_3] \xrightarrow{3} [N_1\,N_2\,N_3]$. Note that all the states within a block agree on all propositions in $\mathcal{AP}_1 \cup \mathcal{AP}_2$. The projected path $\pi\uparrow J$ can be constructed in two steps. First, eliminate all state components not in $J$ from each state:

$$[N_1\,N_2] \xrightarrow{1} [T_1\,N_2] \xrightarrow{3} [T_1\,N_2] \xrightarrow{1} [C_1\,N_2] \xrightarrow{1} [N_1\,N_2] \xrightarrow{3} [N_1\,N_2] \xrightarrow{3} [N_1\,N_2]$$

Then collapse all identical states into a single state, eliminating the transitions in between:

$$\pi\uparrow J = [N_1\,N_2] \xrightarrow{1} [T_1\,N_2] \xrightarrow{1} [C_1\,N_2] \xrightarrow{1} [N_1\,N_2]$$

## 5.2 Semantics of the Pair-Systems

The semantics of $(S^0_{ij}, P^j_i \| P^i_j)$ is given by the global state transition diagram $M_{ij}$ generated by its execution. For example, Figure 3 gives the global state transition diagram generated by the execution of the processes in Figure 1 (with the initial state set being $\{[N_1\,x_{12} = 1\,N_2], [N_1\,x_{12} = 2\,N_2]\}$). We call the global state transition diagram of a pair-system a *pair-structure*.

*Definition* 5.2.1 (*Pair-Structure*). Let $i\,I\,j$. The semantics of $(S^0_{ij}, P^j_i \| P^i_j)$ is given by the pair-structure $M_{ij} = (S^0_{ij}, S_{ij}, R_{ij})$ where

(1) $S_{ij}$ is a set of $ij$-states,

(2) $S^0_{ij} \subseteq S_{ij}$ gives the initial states of $(S^0_{ij}, P^j_i \| P^i_j)$, and

(3) $R_{ij} \subseteq S_{ij} \times \{i,j\} \times S_{ij}$ is a transition relation giving the transitions of $(S^0_{ij}, P^j_i \| P^i_j)$. A transition $(s_{ij}, h, t_{ij})$ by $P^{\bar{h}}_h$ is in $R_{ij}$ if and only if all of the following hold:
   - (a) $h \in \{i,j\}$,
   - (b) $s_{ij}$ and $t_{ij}$ are $ij$-states, and
   - (c) there exists an arc $(s_{ij}\uparrow h, B^{\bar{h}}_h \to A^{\bar{h}}_h, t_{ij}\uparrow h)$ in $P^{\bar{h}}_h$ such that
     - (i) $s_{ij}(B^{\bar{h}}_h) = true$,
     - (ii) $< s_{ij}\uparrow\mathcal{SH}_{ij} >\, A^{\bar{h}}_h < t_{ij}\uparrow\mathcal{SH}_{ij} >$, and
     - (iii) $s_{ij}\uparrow\bar{h} = t_{ij}\uparrow\bar{h}$.

Here $\bar{h} = i$ if $h = j$ and $\bar{h} = j$ if $h = i$.

In a transition $(s_{ij}, h, t_{ij})$, we say that $s_{ij}$ is the *start* state and that $t_{ij}$ is the *finish* state. The transition $(s_{ij}, h, t_{ij})$ is called a $P_h^{\bar{h}}$-transition. In the sequel, we use $s_{ij} \xrightarrow{h} t_{ij}$ as an alternative notation for the transition $(s_{ij}, h, t_{ij})$. $< s_{ij}{\upharpoonright}\mathcal{SH}_{ij} > A < t_{ij}{\upharpoonright}\mathcal{SH}_{ij} >$ is Hoare triple notation [Hoare 1969] for total correctness, which in this case means that execution of $A$ always terminates,[4] and, when the shared variables in $\mathcal{SH}_{ij}$ have the values assigned by $s_{ij}$, leaves these variables with the values assigned by $t_{ij}$. $s_{ij}(B_h^{\bar{h}}) = true$ states that the value of guard $B_h^{\bar{h}}$ in state $s_{ij}$ is $true$.[5]

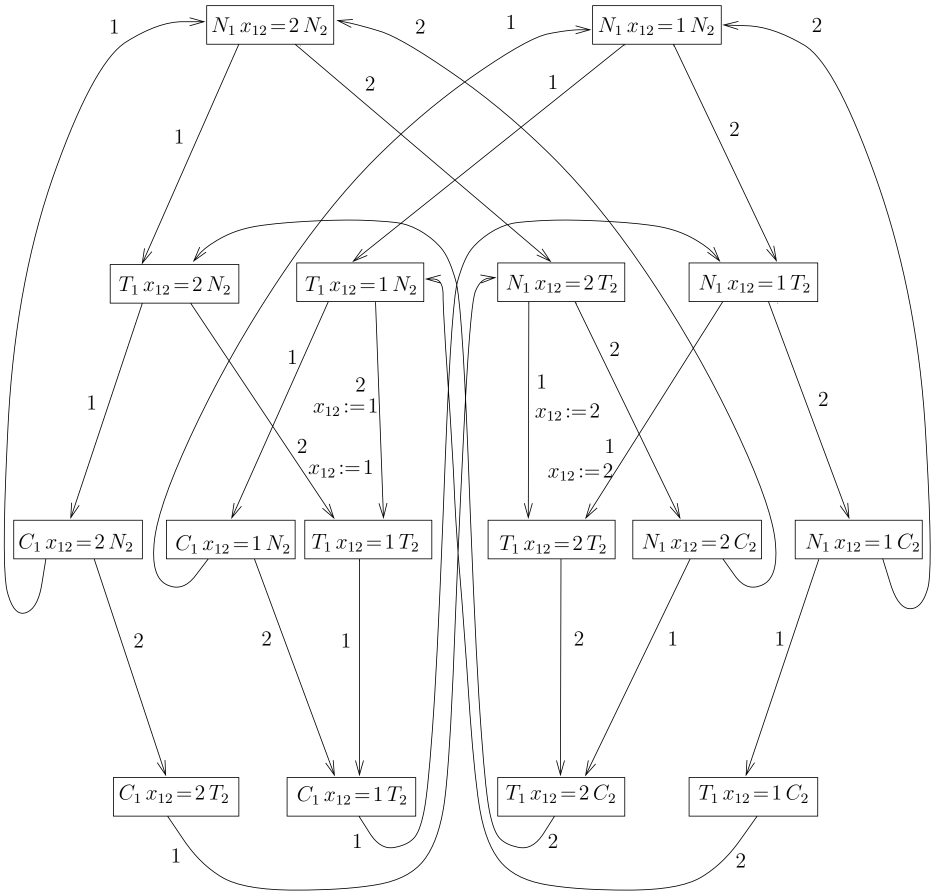
As $M_{ij}$ gives the semantics of $(S_{ij}^0, P_i^j \| P_j^i)$, and we express correctness properties using MPCTL$^*$, we consider that $(S_{ij}^0, P_i^j \| P_j^i)$ possesses a correctness property expressed by an MPCTL$^*$ formula $\bigwedge_{k\ell} f_{k\ell}$ if and only if $M_{ij}, S_{ij}^0 \models \bigwedge_{k\ell} f_{k\ell}$ (note that this is equivalent to $M_{ij}, S_{ij}^0 \models f_{ij}$ by MPCTL$^*$ semantics). We also remark that, as a consequence of the previous definition, $M_{ij}$ does not contain two or more (differently named) states that are identical when viewed as mappings over the domain $\mathcal{AP}_i \cup \mathcal{AP}_j \cup \mathcal{SH}_{ij}$.

In Figure 3, we see that the only states where the value of the shared variable affects the possible outgoing transitions are $[T_1\, x_{12} = 1\, T_2], [T_1\, x_{12} = 2\, T_2]$. Hence, we can minimize the number of global states by collapsing together all other propositionally equivalent states (two states are propositionally equivalent if and only if they assign $true$ to exactly the same atomic propositions). This results in the minimized structure shown in Figure 4. State minimization is performed by taking the quotient with respect to strong bisimulation, denoted by $\sim$ [Browne et al. 1988; Clarke et al. 1986], over structures. Essentially, two states are $\sim$-related if and only if (1) they are propositionally equivalent and (2) all their successor states can be paired into $\sim$-related pairs (duplicates allowed). Given an arbitrary structure $M_{12}(= (S_{12}^0, S_{12}, R_{12}))$, let $M'_{12}(= (S_{12}^{0'}, S'_{12}, R'_{12}))$ result from minimizing the number of states in $M_{12}$. Then, the states of $M'_{12}$ are the equivalence classes of $S_{12}$ with respect to $\sim$: $S'_{12} \stackrel{\text{df}}{=} \{\bar{s} \mid s \in S_{12}\}$, where $\bar{s} \stackrel{\text{df}}{=} \{t \mid s \sim t, s \in S_{12}, t \in S_{12}\}$. The initial states of $M'_{12}$ are those equivalence classes containing some initial state of $M_{12}$: $S_{12}^{0'} \stackrel{\text{df}}{=} \{\bar{s} \mid s \in S_{12}^0\}$. Finally, two states of $M'_{12}$ are connected by a $P_i^j$-transition $(i, j \in \{1, 2\}, i \neq j)$ if and only if they have representatives in $M_{12}$ that are connected by a $P_i^j$-transition: $R_{12} \stackrel{\text{df}}{=} \{\bar{s} \xrightarrow{i} \bar{t} \mid s \xrightarrow{i} t \in R_{12}\}$.

In Figure 4, we display the least information needed to uniquely identify each representative. This necessarily includes all the atomic propositions that are true. It may or may not include shared variable values. For example, $[T_1\, N_2]$ represents the equivalence class $\{[T_1\, x_{12} = 1\, N_2], [T_1\, x_{12} = 2\, N_2]\}$. The value of $x_{12}$ is not needed here: the atomic propositions $T_1$ and $N_2$ being true suffices to identify the states of this equivalence class. By contrast, $[T_1\, x_{12} = 1\, T_2]$ represents the (singleton) equivalence class $\{[T_1\, x_{12} = 1\, T_2]\}$, and here, the value of $x_{12}$ *is* needed

[4]Termination is obvious, since the right-hand side of $A$ is a list of constants.

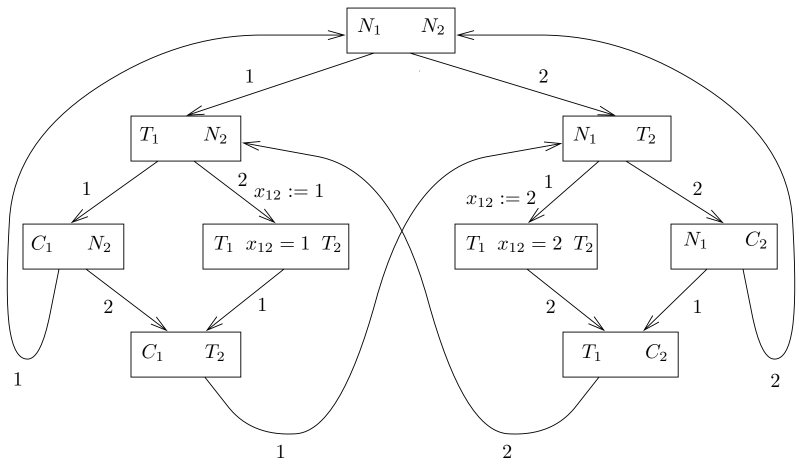
[5]$s_{ij}(B_h^{\bar{h}})$ is defined by the usual inductive scheme: $s_{ij}(\text{“}x_{ij} = h_{ij}\text{”}) = true$ iff $s_{ij}(x_{ij}) = h_{ij}$, $s_{ij}(B1_h^{\bar{h}} \wedge B2_h^{\bar{h}}) = true$ iff $s_{ij}(B1_h^{\bar{h}}) = true$ and $s_{ij}(B2_h^{\bar{h}}) = true$, $s_{ij}(\neg B1_h^{\bar{h}}) = true$ iff $s_{ij}(B1_h^{\bar{h}}) = false$.




The initial state set is $\{[N_1\, x_{12} = 2\, N_2],\ [N_1\, x_{12} = 1\, N_2]\}$.

Fig. 3. Global state transition diagram of the two-process mutual exclusion example.

to distinguish from $[T_1\, x_{12} = 2\, T_2]$, which represents the (singleton) equivalence class $\{[T_1\, x_{12} = 2\, T_2]\}$. Note that, in Figure 3, the states $[T_1\, x_{12} = 1\, T_2]$ and $[T_1\, x_{12} = 2\, T_2]$ are $\sim$-inequivalent, since they have one successor state each, and these successor states (namely, $[C_1\, x_{12} = 1\, T_2]$ and $[T_1\, x_{12} = 2\, C_2]$) are propositionally inequivalent, and so cannot be $\sim$-related. Since the initial states of the structures of Figures 3 and 4 are $\sim$-related, they satisfy the same CTL$^*$ formulae, by Theorem 2 of Clarke et al. [1986]. Hence in particular, the structure of Figure 4 satisfies the CTL$^*$ specification for the two-process mutual exclusion problem. The strong bisimulation relation can be computed in time $O(|S_{12}|^2)$ as a fixpoint, as outlined in Milner [1989, Ch. 4]. Hence, the reduced form can be computed automatically in time $O(|S_{12}|^2)$.




The initial state set is $\{ [N_1\ N_2] \}$.

Fig. 4. Reduced global state transition diagram of the two-process mutual exclusion example.

## 5.3 Semantics of $I$-Systems and $J$-Systems

The semantics of $(S_I^0, P_{i_1}^I \| \dots \| P_{i_K}^I)$ is given by the global state transition diagram $M_I$ generated by its execution. We call the global state transition diagram of an $I$-system an *$I$-structure*.

*Definition* 5.3.1 (*$I$-Structure*). The semantics of $(S_I^0, P_{i_1}^I \| \dots \| P_{i_K}^I)$ is given by the *$I$-structure* $M_I = (S_I^0, S_I, R_I)$ where

(1) $S_I$ is a set of $I$-states,
(2) $S_I^0 \subseteq S_I$ gives the initial states of $(S_I^0, P_{i_1}^I \| \dots \| P_{i_K}^I)$, and
(3) $R_I \subseteq S_I \times \{i_1, \dots, i_K\} \times S_I$ is a transition relation giving the transitions of $(S_I^0, P_{i_1}^I \| \dots \| P_{i_K}^I)$. A transition $(s, i, t)$ by $P_i^I$ is in $R_I$ if and only if
  (a) $i \in \{i_1, \dots, i_K\}$,
  (b) $s$ and $t$ are $I$-states, and
  (c) there exists an arc $(s{\uparrow}i,\ \bigwedge_{j \in I(i)} B_i^j \to //_{j \in I(i)} A_i^j,\ t{\uparrow}i)$ in $P_i^I$ such that all of the following hold:
    (i) for all $j$ in $I(i)$, $s{\uparrow}ij(B_i^j) = \mathit{true}$ and $< s{\uparrow}\mathcal{SH}_{ij} > A_i^j < t{\uparrow}\mathcal{SH}_{ij} >$,
    (ii) for all $j$ in $\{i_1, \dots, i_K\} - \{i\}$: $s{\uparrow}j = t{\uparrow}j$, and
    (iii) for all $j, k$ in $\{i_1, \dots, i_K\} - \{i\}$, $j\, I\, k$: $s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}$.

In a transition $(s, i, t)$, we say that $s$ is the *start* state, and $t$ is the *finish* state. The transition $(s, i, t)$ is called a $P_i^I$-transition. In the sequel, we use $s \xrightarrow{i} t$ as alternative notation for the transition $(s, i, t)$. Also, if $I$ is set to $\{\{i, j\}\}$ in Definition 5.3.1, then the result is, as expected, the pair-structure definition (5.2.1). In other words, the two definitions are consistent. Furthermore, the semantics of a $J$-system, $J \subseteq I$ (see Section 5.1.1) is given by the $J$-structure $M_J = (S_J^0, S_J, R_J)$, which is obtained by using $J$ for $I$ in Definitions 5.1.1 and 5.3.1.

As $M_I$ gives the semantics of $(S_I^0, P_{i_1}^I \| \dots \| P_{i_K}^I)$, and we express correctness properties using MPCTL*, we consider that $(S_I^0, P_{i_1}^I \| \dots \| P_{i_K}^I)$ possesses a correctness property expressed by an MPCTL* formula $\bigwedge_{k\ell} f_{k\ell}$ if and only if $M_I, S_I^0 \models \bigwedge_{k\ell} f_{k\ell}$ (note that this is equivalent to $M_I, S_I^0 \models \bigwedge(i,j) \in I \,.\, (f_{ij})$ by MPCTL* semantics).

## 5.4 A Compact Representation for Synchronization Skeletons

Suppose $P_i^j$ (for every $j \in I(i)$) contains two arcs from $i$-state $s_i$ to $i$-state $s_i'$, e.g., $a_{i,1}^j = (s_i, B_{i,1}^j \to A_{i,1}^j, s_i')$ and $a_{i,2}^j = (s_i, B_{i,2}^j \to A_{i,2}^j, s_i')$. Then, by the MP-synthesis definition (5.1.1), $P_i^I$ contains $2^{|I(i)|}$ arcs from $i$-state $s_i$ to $i$-state $s_i'$, one arc for each element of the cartesian product:

$$\{a_{i,1}^{j_1}, a_{i,2}^{j_1}\} \times \cdots \times \{a_{i,1}^{j_n}, a_{i,2}^{j_n}\}$$

where $\{j_1, \dots, j_n\} = I(i)$. Thus, $P_i^I$ is exponentially large in $K$ $(= |dom(I)|)$ in the worst case (since $|I(i)| = K-1$ when $i \, I \, j$ for every $j$ in $dom(I) - \{i\}$), which defeats the purpose of MP-synthesis. We deal with this by defining a compact representation for processes in which there is at most one arc between any pair of (local) states, thereby avoiding the exponential blowup illustrated above. We provide an overview of this notation here, with full details and definitions in Appendix D.

Consider a pair-process $P_i^j$ which has two arcs from state $s_i$ to state $s_i'$, labeled with the synchronization commands $B_{i,1}^j \to A_{i,1}^j$, $B_{i,2}^j \to A_{i,2}^j$. In compact notation, we replace these two arcs by a single arc whose label is $B_{i,1}^j \to A_{i,1}^j \oplus B_{i,2}^j \to A_{i,2}^j$. $\oplus$ is a binary operator which takes a pair of guarded commands as arguments. Roughly, the operational semantics of $B_{i,1}^j \to A_{i,1}^j \oplus B_{i,2}^j \to A_{i,2}^j$ is that if one of the guards $B_{i,1}^j, B_{i,2}^j$ evaluates to true, then the corresponding body $A_{i,1}^j, A_{i,2}^j$ respectively, can be executed. If neither $B_{i,1}^j$ nor $B_{i,2}^j$ evaluates to true, then the command "blocks," i.e., waits until one of $B_{i,1}^j, B_{i,2}^j$ evaluates to true.[6] We call an arc whose label has the form $\oplus_{\ell \in [1:n]} B_{i,\ell}^j \to A_{i,\ell}^j$ a *(pair) move*. In compact notation, a pair-process has at most one move between any pair of local states.

Now an $I$-process $P_i^I$ is derived by "composing" all the moves of $P_i^j$ (as $j$ varies over $I(i)$) which have the same start and end states. Towards this end we define the binary composition operator $\otimes$ on guarded commands. Roughly, the operational semantics of $B_{i,1}^j \to A_{i,1}^j \otimes B_{i,2}^j \to A_{i,2}^j$ is that if both of the guards $B_{i,1}^j, B_{i,2}^j$ evaluate to true, then the bodies $A_{i,1}^j, A_{i,2}^j$ can be executed in parallel. If at least one of $B_{i,1}^j$, $B_{i,2}^j$ evaluates to false, then the command "blocks," i.e., waits until both of $B_{i,1}^j, B_{i,2}^j$ evaluate to true.

We now recast the MP-synthesis Definition (5.1.1) into compact form as follows.

*Definition* 5.4.1 (*Compact MP-Synthesis*). A *compact I-process* $P_i^I$ is derived from the compact pair-process $P_i^j$ as follows:

$P_i^I$ contains a move from $s_i$ to $t_i$ with label $\otimes_{j \in I(i)} \oplus_{\ell \in [1:n]} B_{i,\ell}^j \to A_{i,\ell}^j$

iff

for every $j$ in $I(i)$: $P_i^j$ contains a move from $s_i$ to $t_i$ with label $\oplus_{\ell \in [1:n]} B_{i,\ell}^j \to A_{i,\ell}^j$.

[6] This interpretation was proposed by Dijkstra [1982].

The *initial state set* $S_I^0$ of the $I$-system is derived from the initial state $S_{ij}^0$ of the pair-system as follows:

$$S_I^0 = \{s \mid \bigwedge(i,j) \in I \,.\, (s{\uparrow}ij \in S_{ij}^0)\}.$$

Thus an $I$-process in compact notation has at most one move of the form $(s_i, \otimes_{j\in I(i)} \oplus_{\ell\in[1:n]} B_{i,\ell}^j \to A_{i,\ell}^j, t_i)$ between any pair of $i$-states $s_i, t_i$. The $I$-move $(s_i, \otimes_{j\in I(i)} \oplus_{\ell\in[1:n]} B_{i,\ell}^j \to A_{i,\ell}^j, t_i)$ can be regarded as a type of "conjunctive normal form" for guarded commands, being a conjunction over all $j$ in $I(i)$ of the pair-moves $\oplus_{\ell\in[1:n]} B_{i,\ell}^j \to A_{i,\ell}^j$, each of which can be regarded as a "disjunction" of guarded commands.

A move in a pair-process is simply a special case of a move in an $I$-process when $I$ is a single pair, i.e., if $I = \{\{i,k\}\}$, then $I(i)$ in $\otimes_{j\in I(i)}$ expands to the singleton $\{k\}$ giving a move in the pair-process $P_i^k$.

We define the following notation. We use $a_i^j, a_i^I$ to denote moves in $P_i^j, P_i^I$, respectively. Consider a move $a_i^I$ of $P_i^I$. By the compact MP-synthesis definition (5.4.1), $a_i^I$ has the form $(s_i, \otimes_{j\in I(i)} \oplus_{\ell\in[1:n]} B_{i,\ell}^j \to A_{i,\ell}^j, t_i)$. Let $a_i^I.start, a_i^I.guard_j$ denote $s_i, \bigvee_{\ell\in[1:n]} B_{i,\ell}^j$, respectively, and let $a_i^I.guard$ denote $\bigwedge_{j\in I(i)} a_i^I.guard_j$. In the special case $I = \{\{i,j\}\}$, let $a_i^j$ denote the move $(s_i, \oplus_{\ell\in[1:n]} B_{i,\ell}^j \to A_{i,\ell}^j, t_i)$, and let $t_k.moves$ denote the set $\{a_k^I \mid a_k^I \in P_k^I \text{ and } a_k^I.start = t_k\}$.

The semantics of compact pair-systems and compact $I$-systems is given by *compact pair-structures* and *compact $I$-structures*, respectively. The following definitions are a direct consequence of the meaning of $\oplus, \otimes$, and the definitions for pair-structures (Definition 5.2.1) and $I$-structures (Definition 5.3.1).

*Definition* 5.4.2 (*Compact Pair-Structure*). Let $i\,I\,j$. The semantics of $(S_{ij}^0, P_i^j \| P_j^i)$ in compact notation is given by the pair-structure $M_{ij} = (S_{ij}^0, S_{ij}, R_{ij})$ where

(1) $S_{ij}$ is a set of $ij$-states,
(2) $S_{ij}^0 \subseteq S_{ij}$ gives the initial states of $P_i^j \| P_j^i$, and
(3) $R_{ij} \subseteq S_{ij} \times \{i,j\} \times S_{ij}$ is a transition relation giving the transitions of $P_i^j \| P_j^i$. A transition $(s_{ij}, h, t_{ij})$ by $P_h^{\bar{h}}$ is in $R_{ij}$ if and only if all of the following hold:
  (a) $h \in \{i,j\}$,
  (b) $s_{ij}$ and $t_{ij}$ are $ij$-states, and
  (c) there exists a move $(s_{ij}{\uparrow}h, \oplus_{\ell\in[1:n]} B_{h,\ell}^{\bar{h}} \to A_{h,\ell}^{\bar{h}}, t_{ij}{\uparrow}h)$ in $P_h^{\bar{h}}$ such that there exists $m \in [1:n]$:
    (i) $s_{ij}(B_{h,m}^{\bar{h}}) = true$,
    (ii) $< s_{ij}{\uparrow}\mathcal{SH}_{ij} > A_{h,m}^{\bar{h}} < t_{ij}{\uparrow}\mathcal{SH}_{ij} >$, and
    (iii) $s_{ij}{\uparrow}\bar{h} = t_{ij}{\uparrow}\bar{h}$.
  Here $\bar{h} = i$ if $h = j$ and $\bar{h} = j$ if $h = i$.

*Definition* 5.4.3 (*Compact I-Structure*). The semantics of $(S_I^0, P_{i_1}^I \| \ldots \| P_{i_K}^I)$ in compact notation is given by the $I$-structure $M_I = (S_I^0, S_I, R_I)$ where

(1) $S_I$ is a set of $I$-states,
(2) $S_I^0 \subseteq S_I$ gives the initial states of $P_{i_1}^I \| \ldots \| P_{i_K}^I$, and

(3) $R_I \subseteq S_I \times dom(I) \times S_I$ is a transition relation giving the transitions of $P^I_{i_1} \| \ldots \| P^I_{i_K}$. A transition $(s, i, t)$ by $P^I_i$ is in $R_I$ if and only if
   (a) $i \in dom(I)$,
   (b) $s$ and $t$ are $I$-states, and
   (c) there exists a move $(s{\uparrow}i, \otimes_{j \in I(i)} \oplus_{\ell \in [1:n]} B^j_{i,\ell} \rightarrow A^j_{i,\ell}, t{\uparrow}i)$ in $P^I_i$ such that all of the following hold:
      (i) for all $j$ in $I(i)$, there exists $m \in [1:n]$:
      $$s{\uparrow}ij(B^j_{i,m}) = true \text{ and } < s{\uparrow}\mathcal{SH}_{ij} > A^j_{i,m} < t{\uparrow}\mathcal{SH}_{ij} >,$$
      (ii) for all $j$ in $dom(I) - \{i\}$: $s{\uparrow}j = t{\uparrow}j$, and
      (iii) for all $j, k$ in $dom(I) - \{i\}$, $j\, I\, k$: $s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}$.

Returning to the example given at the beginning of this section, the pair of arcs $(s_i, B^j_{i,1} \rightarrow A^j_{i,1}, s'_i)$, $(s_i, B^j_{i,2} \rightarrow A^j_{i,2}, s'_i)$ in $P^j_i$ are replaced by the single move $(s_i, (B^j_{i,1} \rightarrow A^j_{i,1} \oplus B^j_{i,2} \rightarrow A^j_{i,2}),\ s'_i)$, and the $2^{|I(i)|}$ arcs in $P^I_i$ are replaced by the single move $(s_i,\ \otimes_{j \in I(i)}(B^j_{i,1} \rightarrow A^j_{i,1} \oplus B^j_{i,2} \rightarrow A^j_{i,2}),\ s'_i)$. A full discussion of compact representation is given in Appendix D.

## 5.5 Examples

Using the compact notation, we synthesize the following programs. We start with the compact solution to the two-process mutual exclusion problem, which is shown in Figure 5 (this is simply the compact representation of the solution shown in Figure 1). From this we synthesize the process $P_1$ in the solution for the original dining philosophers problem (five philosophers in a circle), which is shown in Figure 6. The processes $P_2 - P_5$ can be obtained from $P_1$ by appropriate process index substitutions. Figure 7 gives, in compact notation, the process $P^I_i$ in the solution for the generalized dining philosophers problem. The interconnection relation $I$ gives the neighborhood relationship among philosophers. If $I$ is a complete graph, then we get a solution to the $K$-process mutual exclusion problem.

Finally, Figure 8 gives the process $P^I_i$ in the solution to the drinking philosophers problem [Chandy and Misra 1984; 1988]. $r^j_i$ is a shared variable which is assigned to nondeterministically upon entry to the trying region. If it is set to $true$, then $P^I_i$ will, upon entry into its critical section, require the "bottle" that is shared with $P^I_j$. If, on the other hand, $r^j_i$ is set to $false$, then $P^I_i$ will not require the shared bottle upon critical section entry. Thus, if $r^j_i$ and $r^i_j$ are both set to $true$, then $P^I_i$ and $P^I_j$ will contend for the bottle that they share, and so will exclude each other upon entry to the critical section this time around. If either (or both) of $r^j_i, r^i_j$ are set to $false$, then $P^I_i$ and $P^I_j$ may enter their critical sections simultaneously this time around, since at least one of them will not require access to the shared bottle. We have not shown the two-process drinking philosophers solution here.

In Figures 7 and 8, the interconnection relation $I$ is not fixed, but is a parameter to the synthesized program. A single such parameterized program thus represents an infinite family of "concrete" programs, one for each value of $I$. In particular, $I$ can be "arbitrarily large." For example, we could set $I = \{(i,j) \mid i \neq j, i, j \in \{1, \ldots, 1000\}\}$. Figure 7 would then represent a mutual exclusion solution for 1000 processes. Figure 9 shows process $P_1$ of this program. As this program has on the order of $2^{499,500}$ global states (there are $1000 * 999/2 = 499,500$ binary pairwise shared variables), straightforward state exploration is clearly infeasible for the synthesis of this program. By using other "large" values for $I$, we can obtain solutions containing large numbers of processes for the generalized dining and drinking philosophers problems from Figures 7 and 8, respectively. For example, if $I = \{\{1,2\},\{2,3\},\ldots,\{999,1000\},\{1000,1\}\}$, then we obtain dining, drinking philosophers solutions for 1000 philosophers arranged in a circle. One very important advantage of the compact notation is that it permits this "parameterized" representation, whereas our normal notation does not.

## 6. SOUNDNESS OF THE METHOD

Since we synthesize $I$-systems from pair-systems, we formulate the soundness of the synthesis method as follows: if a pair-system has a particular correctness property, then the $I$-system also has that property. As stated previously, we express correctness properties *pairwise*, i.e., as MPCTL$^*$ formulae. Furthermore, we only consider a subset of MPCTL$^*$—which nonetheless can express many safety and liveness properties that arise in practice, e.g., invariants that can be expressed pairwise, and temporal leads-to properties over individual processes. We discuss correctness properties in more detail in Section 6.1. Also, independently of our soundness results, we characterize sufficient conditions under which the $I$-system is deadlock-free.

Leading up to our main soundness results are many preliminary results that establish the prerequisite relationships between pair-structures and $I$-structures. Our results can be grouped into the following sequence:

—*Similarity Results (Section 6.2)*: Formalize the notion of similarity of processes and establish that all pair-structures are similar.

—*Projection Results (Section 6.3)*: Establish that a state and its projection satisfy the same propositional formulae over their common atomic propositions. Also establish that a path and its projection satisfy the same linear-time temporal logic, i.e., PTL (see Section 3.1), formulae over their common atomic propositions. These results relate states (paths) and their projections independently of the occurrence of these states (paths) in the structures of interest, namely the pair and $I$-structures.

—*Mapping Results (Section 6.4)*: Establish that a $P_i^I$-transition occurs in the $I$-structure if and only if all of its projections onto $\{\{i,j\}\}$, for $j \in I(i)$, occur in the corresponding $ij$-structures. Also establish that if a state (path) occurs in the $I$-structure, then its projection onto $J$ occurs in the $J$-structure (for any $J \subseteq I$). In other words, these results relate the occurrence of states (paths) in the $I$-structure to the occurrence of states (paths) in the $J$-structure. Together with the projection results, the mapping results provide a basis for relating the MPCTL$^*$ formulae satisfied by the $I$-structure to the MPCTL$^*$ formulae satisfied by the pair-structures.

—*Deadlock-Freedom Results (Section 6.5)*: Establish that the $I$-system is free of deadlock. This is done using a technical assumption (the "wait-for-graph assumption") which, roughly, guarantees that circular waiting chains cannot form. Uses the mapping results and the compact representation. These results stand

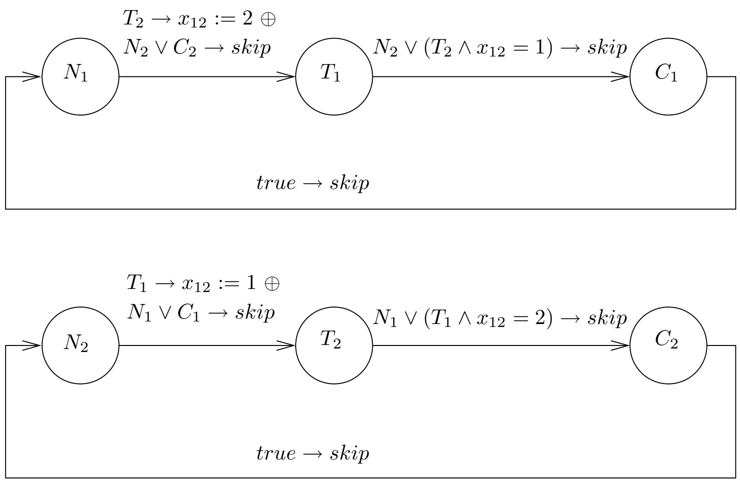



Fig. 5. Compact skeletons $P_1^2$ (top) and $P_2^1$ (bottom) for two-process mutual exclusion.

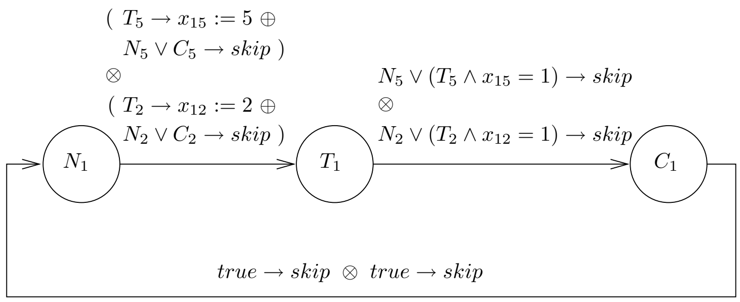



Fig. 6. Compact skeleton $P_1$ in the solution to the original dining philosophers problem.

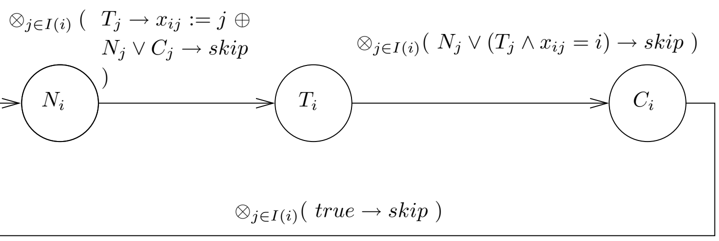



Fig. 7. Compact skeleton $P_i$ in the solution to the generalized dining philosophers problem.

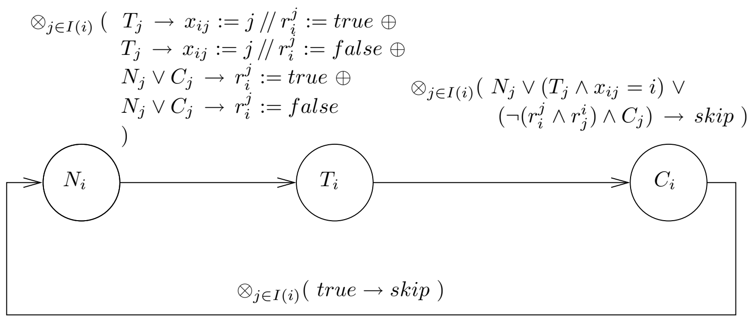



Fig. 8. Compact skeleton $P_i$ in the drinking philosophers solution.

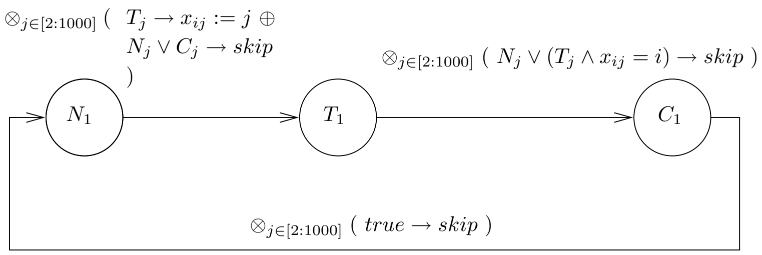



Fig. 9. Compact skeleton $P_1$ in the solution to the 1000-process mutual exclusion problem.

alone as characterizing conditions under which the $I$-system is deadlock-free. In the soundness results, deadlock-freedom is assumed.

—*Liveness Results (Section 6.7)*: Establish that individual starvation of processes cannot occur in the $I$-system. Again this requires a technical assumption (the "liveness assumption"). This assumption prevents a process from executing forever while indirectly (i.e., via a "chain" of other processes) blocking another process that needs to progress. Uses the mapping and deadlock-freedom results and the compact representation.

—*Soundness Results (Sections 6.6 and 6.8)*: Our main soundness results. We establish two results, one for completely interconnected systems (i.e., every pair of processes are $I$-related) and another for systems with arbitrary interconnection. Uses the projection, mapping, and liveness results.

## 6.1 Correctness Properties and Reachability

Correctness properties in our framework are specified as MPCTL* formulae. Since MPCTL* formulae are evaluated on particular states of a structure, a suitable notion of correctness is that the formula specifying the correctness property of interest is true in all initial states of the structure. In other words, $P$ satisfies $f$ if and only if $M, S^0 \models f$, where $M = (S^0, S, R)$ is the global state transition diagram of $P$. Now, from CTL* semantics, we see that the truth value of an MPCTL* formula in an initial state of a structure depends only on the "reachable" portion of the structure. We therefore define an *initialized path* to be a path whose first state is an initial state, and a *reachable state* to be a state lying on some initialized path. Also define a *reachable transition* to be a transition lying along some initialized path.

In the sequel, we assume that every $i$-state $s_i$ of $P_i^j$ is reachable in $M_{ij}$, i.e., for every $s_i$, there exists a reachable $ij$-state $s_{ij}$ such that $s_{ij}{\uparrow}i = s_i$. Any such $s_i$ that is not reachable can be removed from $P_i^j$ (along with all its incident arcs) without affecting the reachable portion of $M_{ij}$ generated by the execution of $(S_{ij}^0, P_i^j \| P_j^i)$, and hence the CTL* formulae satisfied by $M_{ij}$.

We define the following notation: $S_I^r$ is the set of reachable states of $M_I$, and $R_I^r = R_I{\uparrow}S_I^r$ is the restriction of the transition relation $R_I$ to the set of reachable states of $M_I$. $M_I^r$ is the structure $(S_I^0, S_I^r, R_I^r)$. If $I = \{\{i,j\}\}$ (i.e., the two-process case) then we use $S_{ij}^r, R_{ij}^r, M_{ij}^r$ for $S_I^r, R_I^r, M_I^r$, respectively. For an arbitrary $J \subseteq I$, we define $S_J^r, R_J^r$, and $M_J^r = (S_J^0, S_J^r, R_J^r)$ by using $J$ for $I$ in the previous definitions.

In this article, the correctness properties of interest are as follows, where $h_{k\ell}$ is a formula of $\mathcal{L}(\mathcal{AP}_k, \mathcal{AP}_\ell, \neg, \wedge)$, and $p_k, q_k, r_k$ are formulae of $\mathcal{L}(\mathcal{AP}_k, \neg, \wedge)$:

—$\bigwedge_{k\ell} h_{k\ell}$: Specifies the initial states; all initial states must satisfy $h_{k\ell}$.

—$\bigwedge_{k\ell} AGh_{k\ell}$: Expresses a propositional invariant; $h_{k\ell}$ must be true in all reachable states.

—$\bigwedge_k AG(p_k \Rightarrow A[q_k U r_k])$: Expresses a temporal leads-to properties, combined with an "unless" property; from any global state satisfying $p_k$, eventually a state satisfying $r_k$ must be reached (along all outgoing fullpaths), and all intervening states must satisfy $q_k$.

—$\bigwedge_k AG(p_k \Rightarrow AY_k q_k)$: Expresses a "local structure" property; every local state of process $k$ that satisfies $p_k$ has as immediate successors in the skeleton for process $k$ only local states that satisfy $q_k$.[7]

—$\bigwedge_k AG(p_k \Rightarrow EX_k q_k)$: Expresses a "local structure" property; every local state of process $k$ that satisfies $p_k$ has some immediate successor in the skeleton for process $k$ which satisfies $q_k$, and when process $k$ is in the former local state, it always has some enabled transition to the latter local state, no matter what the current global state is.[8]

## 6.2 Process Similarity

Our notion of process similarity is a syntactic one: two processes are *similar* if and only if one can be obtained from the other by swapping their indices. To formalize this, we introduce the *process index substitution operator* $\theta = \{j_1/i_1, \ldots, j_m/i_m\}$ which denotes the simultaneous replacement of process indices $i_1, \ldots, i_m$ by process indices $j_1, \ldots, j_m$, respectively. We require that $i_1, \ldots, i_m$ be pairwise distinct, and $j_1, \ldots, j_m$ be pairwise distinct, and so $\theta$ specifies a "renaming" of process indices and can be applied to all of the pair syntactic constructs defined in the article, as well as any pair model (e.g., $M_{ij}$). We define $\theta$ formally in Appendix C. To preserve the similarity of pair-processes under interchange of process indices, we make the

[7] We ignore the possibility of "dead code," i.e., unreachable portions of the skeleton for process $k$.
[8] In our technical results, some further technical restrictions on the form of $p_k, q_k$ are made.

convention that the shared variables $x_{ij}$ and $x_{ji}$ are actually the same variable, i.e., $x_{ij}\{i/j, j/i\}$, $x_{ji}$, and $x_{ij}$ all denote the same variable. However, if the variable is superscripted, then index interchange results in a new variable. Thus $x^i_{ij}\{i/j, j/i\}$ and $x^j_{ji}$ denote the same variable, which is a different variable than that denoted by $x^i_{ij}$. Our assumption of process similarity is now formally stated as follows:

*Process Similarity Assumption.* For all $i, j, i', j' \in \{i_1, \ldots, i_K\}$ such that $i \neq j$, $i' \neq j'$, $i\,I\,j$, $i'\,I\,j'$,

$$P^j_i \;=\; P^{j'}_{i'}\{i/i', j/j'\}.$$

In other words, any pair-process can be derived from any other pair-process by index substitution. For special cases of the similarity relation among pair-processes, we have $P^j_i = P^i_j\{i/j, j/i\}$, $P^j_i = P^{j'}_i\{j/j'\}$, and $P^j_i = P^j_{i'}\{i/i'\}$.

The MP-synthesis definition (5.1.1) requires us to generate $P^j_i$ for every pair $(i, j)$ in $I$. Using the process similarity assumption, we only require a single pair-system in order to generate all the required pair-systems. For example, (assuming $(1, 2) \in I$), if we are given the pair-system $(S^0_{12}, P^2_1 \| P^1_2)$, then we can generate $P^j_i$ for any pair $(i, j)$ in $I$ using $P^j_i = P^2_1\{i/1, j/2\}$, which is obtained from the process similarity assumption by letting $i' = 1, j' = 2$.

Analogously to our assumption that all pair-processes are similar, we assume that all pair-systems have similar sets of initial states.

*Initial State Assumption.* Let $i, j, i', j'$ be arbitrary elements of $\{i_1, \ldots, i_K\}$ such that $i\,I\,j$ and $i'\,I\,j'$. Then we have

$$S^0_{ij} = S^0_{i'j'}\{i/i', j/j'\}.$$

The initial state assumption guarantees that $S^0_I$ (given by the MP-synthesis definition (5.1.1)) is nonempty, given that $S^0_{ij}$ is nonempty. Since both the initial states and the processes of all pair systems are similar, it follows that the pair-structures generated by the executions of the pair-systems are similar.

PROPOSITION 6.2.1 (PAIR-STRUCTURE SIMILARITY). *Let $i, j, i', j'$ be arbitrary elements of $\{i_1, \ldots, i_K\}$ such that $i\,I\,j$ and $i'\,I\,j'$. Then we have*

$$M_{ij} = M_{i'j'}\{i/i', j/j'\}.$$

By MPCTL* semantics (Section 3), an MPCTL* formula is not sensitive to the particular process indices in a structure. Hence, if two structures are identical up to a renaming of process indices, then they satisfy exactly the same MPCTL* formulae. Together with Proposition 6.2.1, this yields the following observation:

*Observation* 6.2.2. Let $i, j, i', j'$ be arbitrary elements of $\{i_1, \ldots, i_K\}$ such that $i\,I\,j$ and $i'\,I\,j'$. Then we have

$$M_{ij}, S^0_{ij} \models \textstyle\bigwedge_{k\ell} f_{k\ell} \text{ iff } M_{i'j'}, S^0_{i'j'} \models \textstyle\bigwedge_{k\ell} f_{k\ell}.$$

### 6.3 State and Path Projection Results

This section relates states (paths, respectively) and their projection in terms of the CTL* formulae that they agree on. No assumption is made about the occurrence of these states (paths) in the structures of interest, namely the pair and $I$-structures.

Since an $I$-state $s$ and its projection $s{\uparrow}J$ agree, by definition of $\uparrow$, on all propositions in $\bigcup_{i \in dom(J)} \mathcal{AP}_i$, we expect these states to satisfy the same propositional logic formulae which contain atomic propositions only from $\bigcup_{i \in dom(J)} \mathcal{AP}_i$.

PROPOSITION 6.3.1 ($I$-STATE PROJECTION). *Let $J \subseteq I$, and let $s$ be an $I$-state. Then*

$$s \models f \text{ iff } s{\uparrow}J \models f$$

*where $f$ is a formula of $\mathcal{L}(\bigcup_{i \in dom(J)} \mathcal{AP}_i, \neg, \wedge)$.*

Likewise, a $J$-state $s_J$ and its projection $s_J{\uparrow}i$ satisfy exactly the same propositional logic formulae which contain atomic propositions only from $\mathcal{AP}_i$ ($i \in dom(J)$).

PROPOSITION 6.3.2 (LOCAL STATE PROJECTION). *Let $i \in dom(J)$, and let $s_J$ be a $J$-state. Then*

$$s_J \models f_i \text{ iff } s_J{\uparrow}i \models f_i$$

*where $f_i$ is a formula of $\mathcal{L}(\mathcal{AP}_i, \neg, \wedge)$.*

We remind the reader that a $J$-block is a maximal subsequence of a path containing no transition by a process in the domain of $J$ (Section 5.1.2). Since a $J$-block $B$ of a path $\pi$ and its corresponding state $B{\uparrow}J$ in $\pi{\uparrow}J$ agree on all atomic propositions in $\bigcup_{i \in dom(J)} \mathcal{AP}_i$, the sequence of truth values assigned to these propositions along the paths $\pi$ and $\pi{\uparrow}J$ is the same if we ignore repetitions, i.e., allow *stuttering* [Browne et al. 1988; Clarke et al. 1986]. Hence, these paths satisfy exactly the same linear-time temporal logic, i.e., the logic PTL (see Section 3.1), formulae provided that we disallow the next-time operator $X$, which allows repetition to be detected.

LEMMA 6.3.3 (PATH PROJECTION). *Let $\pi$ be a path in $M_I$, and let $J \subseteq I$. Then*

$$\pi \models f \text{ iff } \pi{\uparrow}J \models f$$

*where $f$ is a formula of $\mathcal{L}(\bigcup_{i \in dom(J)} \mathcal{AP}_i, \neg, \wedge, U)$.*

### 6.4 Mapping of $I$-Structures into $J$-Structures

This section relates the occurrence of transitions, states, and paths in the $I$-structure to the occurrence of the projected transitions, states, and paths respectively in $J$-structures ($J \subseteq I$), paying particular attention to the special case where $J$ specifies a single pair (Lemma 6.4.1). Specifically, it shows that projection onto $J$ defines a mapping of the transitions, reachable states, and paths of $M_I$ into the transitions, reachable states, and paths of $M_J$.

By virtue of an arc in an $I$-process $P_i^I$ being a "composition" of the corresponding arcs in the pair-processes $P_i^j$ in all the pair-systems $(S_{ij}^0, P_i^j \| P_j^i)$ (as $j$ ranges over $I(i)$), a transition by $P_i^I$ can occur in the $I$-system if and only if the "projected" transitions by $P_i^j$ in all the pair systems $(S_{ij}^0, P_i^j \| P_j^i)$ (as $j$ ranges over $I(i)$) can all occur. This is formalized as the transition-mapping lemma, which provides the basis for establishing the remaining mapping results.

LEMMA 6.4.1 (TRANSITION MAPPING). *For all $I$-states $s, t \in S_I$ and $i \in dom(I)$,*

$s \xrightarrow{i} t \in R_I$ iff :

$\bigwedge j \in I(i) \,.\, (s{\uparrow}ij \xrightarrow{i} t{\uparrow}ij \in R_{ij})$ *and*
$\bigwedge j \in \{i_1, \ldots, i_K\} - \hat{I}(i) \,.\, (s{\uparrow}j = t{\uparrow}j)$ *and*
$\bigwedge j, k \in \{i_1, \ldots, i_K\} - \{i\}, j\, I\, k \,.\, (s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk})$.

We remind the reader of the example of "composition" of transitions (Section 5.1), which illustrates Lemma 6.4.1. Note that Lemma 6.4.1 relates $M_I$ to a set of $J$-systems (namely, for all $J = (i, j)$ as $j$ ranges over $I(i)$), whereas the remaining results below relate $M_I$ to $M_J$ for a single, arbitrary $J \subseteq I$.

Since, by Lemma 6.4.1, a transition by $P_i^I$ entails "projected" transitions by $P_i^j$ in all the pair systems $(S_{ij}^0, P_i^j \| P_j^i)$ as $j$ ranges over $I(i)$, it follows that a transition by $P_i^I$ entails "projected" transitions by $P_i^j$ in all the pair-systems $(S_{ij}^0, P_i^j \| P_j^i)$ as $j$ ranges over $J(i)$, where $J \subseteq I$. This latter set of transitions entails (again by Lemma 6.4.1) a corresponding transition by $P_i^J$ in the $J$-system. Hence, we conclude that the projection onto $J$ of every transition in the $I$-system is a transition in the $J$-system:

Corollary 6.4.2 (Transition Mapping). *Let $J \subseteq I$ and $i \in dom(J)$. If $s \xrightarrow{i} t \in R_I$, then $s{\uparrow}J \xrightarrow{i} t{\uparrow}J \in R_J$.*

By applying the transition-mapping corollary to every transition along a path $\pi$ in $M_I$, we show that $\pi{\uparrow}J$ is a path in $M_J$:

Lemma 6.4.3 (Path Mapping). *Let $J \subseteq I$. If $\pi$ is a path in $M_I$, then $\pi{\uparrow}J$ is a path in $M_J$.*

As an example of this lemma, consider the path $\pi$ given in Section 5.1.2 and its projection $\pi_{12} = \pi{\uparrow}\{\{1,2\}\}$, where $\pi$ is a path of the three-process mutual exclusion system given in Figure 2. We can verify by inspection that $\pi_{12}$ is a path of the two-process mutual exclusion system given in Figure 1.

From Definition 5.1.1, we see that an initial $I$-state projects onto an initial $J$-state. Hence, a consequence of Lemma 6.4.3 is that initialized paths (see Section 6.1) in $M_I$ project onto initialized paths in $M_J$:

Corollary 6.4.4 (Path Mapping). *Let $J \subseteq I$. If $\pi$ is an initialized path in $M_I$ then $\pi{\uparrow}J$ is an initialized path in $M_J$.*

Since every reachable state lies at the end of some initialized path, we can use the path-mapping corollary to relate reachable states in $M_I$ to their projections in $M_J$:

Corollary 6.4.5 (State Mapping). *Let $J \subseteq I$. If $t$ is a reachable state in $M_I$, then $t{\uparrow}J$ is a reachable state in $M_J$.*

We relativize our notion of reachability as follows. We say that a state $t$ is $s$-*reachable* in $M_I$ if and only if there exists a path in $M_I$ from $s$ to $t$. Note that neither $s$ nor $t$ need be reachable states. The definition for pair-systems, is, as usual, obtained by setting $I$ to $\{\{i, j\}\}$. Since, by Lemma 6.4.3 and the definition of path projection, the projection onto $J$ of a path in $M_I$ from $I$-state $s$ to $I$-state $t$ gives a path in $M_J$ from $s{\uparrow}J$ to $t{\uparrow}J$, we have the following corollary:

Corollary 6.4.6 (Relativized State Mapping). *Let $J \subseteq I$. If $t$ is a $s$-reachable state in $M_I$, then $t{\uparrow}J$ is a $s{\uparrow}J$-reachable state in $M_J$.*

### 6.5 Deadlock Freedom of the $I$-System

Our focus in this article is nonterminating concurrent systems, so we assume that $(S^0_{ij}, P^j_i \| P^i_j)$ is nonterminating, i.e., deadlock-free.[9] Now it is possible that $(S^0_I, P^I_{i_1} \| \dots \| P^I_{i_K})$ is deadlock-prone; we present an example below of a deadlock-prone $I$-system (for $I = \{1,2\},\{2,3\},\{3,1\}\}$) that is synthesized from a deadlock-free pair-system. Thus, even though the pair-system itself is deadlock-free, it permits a certain pattern of blocking behavior that allows deadlock to occur in the synthesized $I$-system. We shall see below that this blocking behavior is that a process executes a transition which results in a global state where the process both blocks some move of another process[10] and has all of its own moves blocked by other processes. Hence, some technical assumption on the pair-system $(S^0_{ij}, P^j_i \| P^i_j)$ that is input to the MP-synthesis definition (5.1.1) is needed. To formulate this assumption, we first define a notion of "wait-for-graph" for our model of parallel computation. Our model of waiting is a restricted form of the AND-OR model [Knapp 1987], since each process has, in general, many possible moves at any time (the OR aspect), and to execute a move requires that all $I$-neighbors of the process allow the move to be executed (the AND aspect).[11] For a given $I$-system in a particular global state, the wait-for-graph characterizes the current blocking relations, i.e., which moves are blocked by which processes. We then characterize the occurrence of deadlocks as the occurrence within the wait-for-graph of a subgraph of a certain form that we call a "supercycle." Essentially, a supercycle is a generalization to the AND-OR waiting model of what a cycle is in the AND model or a knot is in the OR model. Finally, we formulate an assumption on the behavior of pair-systems—the "wait-for-graph assumption"—that prevents the occurrences of supercycles, and hence of deadlocks, and formally establish that this is indeed the case.

6.5.1 *An Example of a Deadlock-Prone $I$-System.* Consider the pair-system with initial state set $\{[V_1\, V_2]\}$ given in Figure 10. The variables $x^2_{12}, y^2_{12}$ are shared variables that are written by $P_1$ only and read by $P_2$ only. Shared variable $x^2_{12}$ controls the move of $P_2$ from $W_2$ to $A_2$, which can be made only when $x^2_{12} = 2$. Likewise, $y^2_{12}$ controls the move of $P_2$ from $W_2$ to $B_2$. By making appropriate assignments to these variables, $P_1$ enables or disables the moves of $P_2$ out of $W_2$

[9] We do not distinguish between deadlock and termination in this article, and thus deadlock freedom and nontermination are expressed by the same CTL formula, namely $AGEXtrue$. It is in principle straightforward to make this distinction: for each $i \in \{i_1, \dots, i_K\}$, introduce a local proposition $term_i$ which is true if and only if process $i$ has terminated. Deadlock-freedom is then written as $AG((term_{i_1} \vee \dots \vee term_{i_K}) \Rightarrow EXtrue)$, i.e., as long as some process has not terminated, some process can execute a transition. Nontermination, on the other hand, is expressed as $AG(\neg(term_{i_1} \vee \dots \vee term_{i_K}))$, i.e., no process terminates.

[10] We say an $I$-process $P^I_i$ blocks a move of $I$-process $P^I_j$ if and only if $i\,I\,j$ and there is an $I$-move $a^I_j$ of $P^I_j$ such that the current local state of $P^I_j$ is the start state of $a^I_j$ (i.e., local control is "at" $a^I_j$), and the conjunct of the guard of $a^I_j$ that tests the shared state with $P^I_i$ (i.e., $a^I_j.guard_i$) is false. In other words, $P^I_j$ is prevented from executing $a^I_j$ by $P^I_i$.

[11] The general AND-OR model allows arbitrary combinations of "and" and "or" resource requests, whereas our model strictly alternates "and" and "or" requests, i.e., an $I$-process requests some move to execute, and that move in turn requests "permission" (true guards) from all the $I$-neighbors of the process.

(namely, $(W_2, x_{12}^2 = 2 \rightarrow skip, A_2)$ and $(W_2, y_{12}^2 = 2 \rightarrow skip, B_2)$) as follows. When $P_1$ moves from $V_1$ to $W_1$, it either disables both these moves (if $P_2$'s current local state is $V_2$), or it non-deterministically chooses one of these moves to disable, and enables the other (if $P_2$'s current local state is $W_2$). $P_2$ exercises a symmetric control of the moves of $P_1$ out of $W_1$ (namely, $(W_1, x_{12}^1 = 1 \rightarrow skip, A_1)$ and $(W_1, y_{12}^1 = 1 \rightarrow skip, B_1)$) by assigning to the variables $x_{12}^1, y_{12}^1$, which play a "bilaterally symmetric" role with respect to $P_1$ and $P_2$ that $x_{12}^2, y_{12}^2$ do. By inspection, the only potential deadlock state is when both processes are at $W$ states and all outgoing moves are blocked, i.e., the state $[W_1\, x_{12}^1 = 2\, y_{12}^1 = 2\, W_2\, x_{12}^2 = 1\, y_{12}^2 = 1]$. However we can easily verify that $(W_1 \wedge W_2) \Rightarrow (x_{12}^1 = 1 \vee y_{12}^1 = 1 \vee x_{12}^2 = 2 \vee y_{12}^2 = 2)$ is an invariant, so this state is not reachable. Hence the pair-system of Figure 10 is deadlock-free. The $I$-system for $I = \{\{1,2\},\{2,3\},\{3,1\}\}$ derived from Figure 10 by the MP-synthesis definition (5.1.1) is shown in Figure 11. This system can reach a deadlocked state via the following path:

$$
\begin{array}{l}
[V_1\, V_2\, V_3] \xrightarrow{1} \\
[x_{13}^3 = 1\, y_{13}^3 = 1\, W_1\, x_{12}^2 = 1\, y_{12}^2 = 1\, V_2\, V_3] \xrightarrow{2} \\
[x_{13}^3 = 1\, y_{13}^3 = 1\, W_1\, x_{12}^2 = 1\, y_{12}^2 = 1\, x_{12}^1 = 2\, y_{12}^1 = 1\, W_2\, x_{23}^3 = 2\, y_{23}^3 = 2\, V_3] \xrightarrow{3} \\
[x_{13}^3 = 1\, y_{13}^3 = 1\, W_1\, x_{12}^2 = 1\, y_{12}^2 = 1\, x_{12}^1 = 2\, y_{12}^1 = 1\, W_2\, x_{23}^3 = 2\, y_{23}^3 = 2 \\
\;x_{23}^2 = 3\, y_{23}^2 = 2\, W_3\, x_{13}^1 = 1\, y_{13}^1 = 3\,]
\end{array}
$$

We can easily see by inspection that the last state (call it $u^{dead}$) of this path is a deadlock state, i.e., has no outgoing transitions: The $P_1$-transition results in both moves out of $W$ of both $P_2, P_3$ being disabled. The $P_2$-transition results in the move $(W_1, x_{12}^1 = 1 \rightarrow skip \otimes x_{13}^1 = 1 \rightarrow skip, A_1)$ of $P_1$ being disabled. Finally, the $P_3$ transition results in the move $(W_1, y_{12}^1 = 1 \rightarrow skip \otimes y_{13}^1 = 1 \rightarrow skip, B_1)$ of $P_1$ being disabled. Hence, in state $u^{dead}$ no move is enabled, and the system is deadlocked.

6.5.2 *The Wait-For-Graph.* We remind the reader of the following notation (see Section 5.4). A move $a_i^I$ of $P_i^I$ has the form $(s_i, \otimes_{j \in I(i)} \oplus_{\ell \in [1:n]} B_{i,\ell}^j \rightarrow A_{i,\ell}^j, t_i)$. $a_i^I.start, a_i^I.guard_j$ denote $s_i, \bigvee_{\ell \in [1:n]} B_{i,\ell}^j$, respectively; and $a_i^I.guard$ denotes $\bigwedge_{j \in I(i)} a_i^I.guard_j$. In the special case of $I = \{\{i,j\}\}$, $a_i^j$ denotes the move $(s_i, \oplus_{\ell \in [1:n]} B_{i,\ell}^j \rightarrow A_{i,\ell}^j, t_i)$. Finally, we use $t_k.moves$ to denote the set of moves $\{a_k^I \mid a_k^I \in P_k^I \text{ and } a_k^I.start = t_k\}$.

The "wait-for-graph" $W_I(s)$ of $(S_I^0, P_{i_1}^I \| \dots \| P_{i_K}^I)$ at state $s$ is a directed bipartite graph. The nodes of this graph are the $I$-processes $P_i^I$ for $i \in \{i_1, \dots, i_K\}$, i.e., all the $I$-processes of $(S_I^0, P_{i_1}^I \| \dots \| P_{i_K}^I)$, and the moves $a_i^I$ of each $P_i^I$ such that $s{\uparrow}i = a_i^I.start$, i.e., all the moves that are candidates for execution. There is an edge in $W_I(s)$ from $P_i^I$ to each node of the form $a_i^I$ in $W_I(s)$. This edge indicates that $a_i^I$ is a possibility for execution by $P_i^I$. Thus $P_i^I$ is an "and" node, since $P_i^I$ is blocked if and only if *all* of its successors (i.e., its possible moves) are blocked. Also, there is an edge in $W_I(s)$ from $a_i^I$ to $P_j^I$ if and only if $j \in I(i)$ and $s{\uparrow}ij(a_i^I.guard_j) = false$, i.e., $a_i^I$ is blocked by $P_j^I$ ($a_i^I$ can be executed in $s$ only if $s{\uparrow}ij(a_i^I.guard_j) = true$ for all $j \in I(i)$. Thus, if there is some $j$ in $I(i)$ such that $s{\uparrow}ij(a_i^I.guard_j) = false$, then $a_i^I$ cannot be executed in state $s$ (of $M_I$)). Thus each move $a_i^I$ is an "or" node,

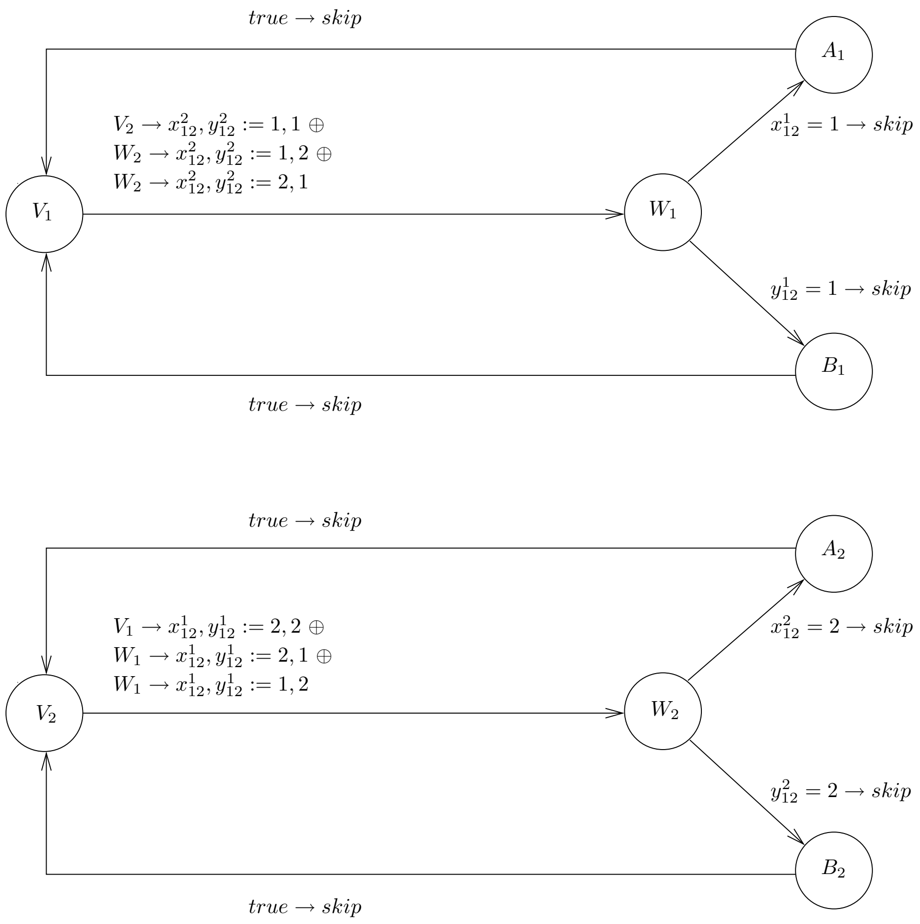



Fig. 10. Deadlock example for two processes.

since $a_i^I$ is blocked if and only if *some* neighbor $P_j^I$ of $P_i^I$ blocks $a_i^I$. Note, however, that we cannot say that $P_i^I$ itself is blocked by $P_j^I$, since there could be another move $b_i^I$ in $P_i^I$ such that $s{\uparrow}ij(b_i^I.guard_j) = true$, i.e., $b_i^I$ is not blocked by $P_j^I$ (in state $s$), so $P_i^I$ can progress in state $s$ by executing $b_i^I$. We now give the formal definition.

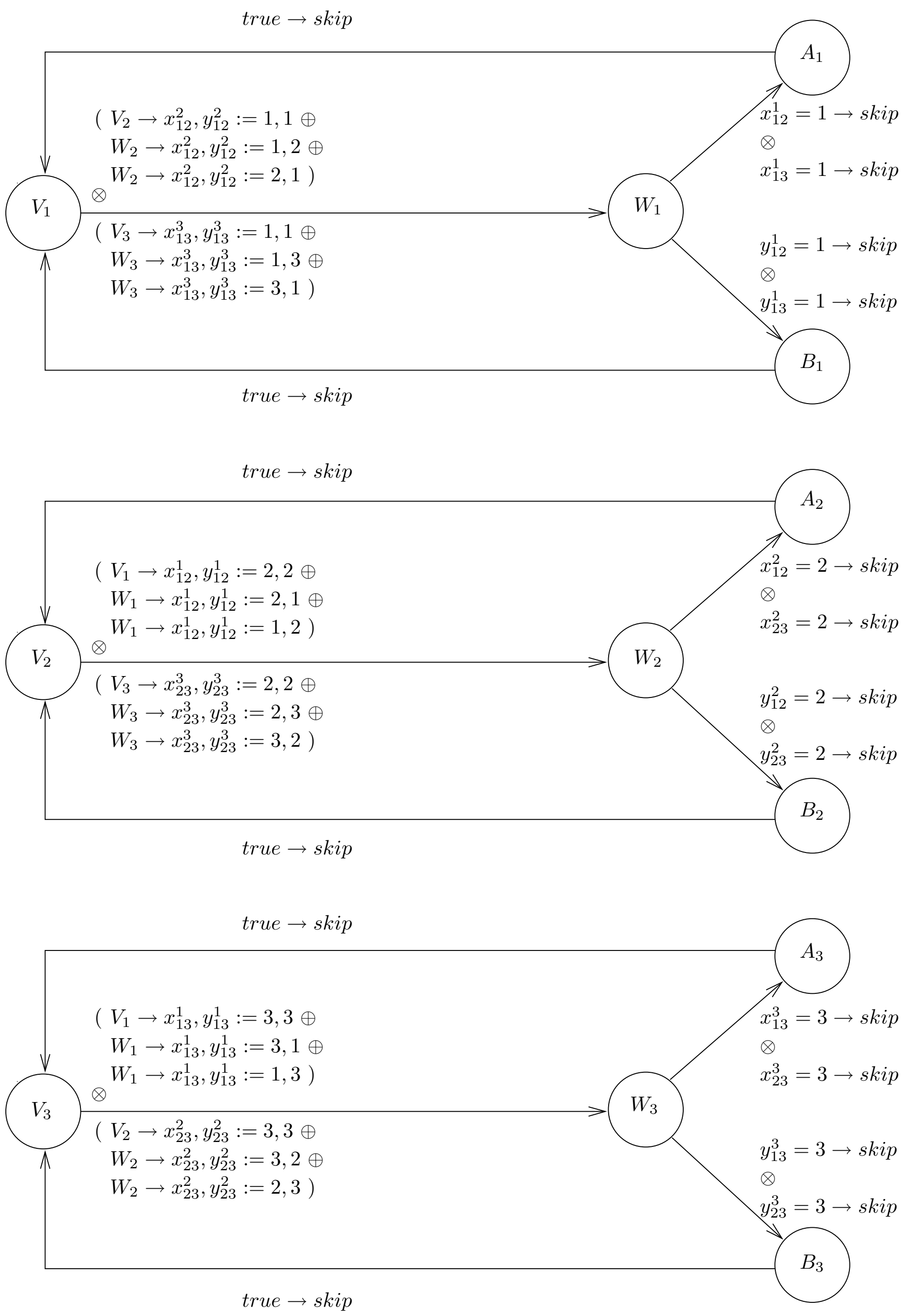



Fig. 11. Deadlock example for three processes.

*Definition* 6.5.2.1 (*Wait-For-Graph* $W_I(s)$). Let $s$ be an arbitrary $I$-state. The *wait-for-graph* $W_I(s)$ of $s$ is a directed bipartite graph, where

(1) the nodes of $W_I(s)$ are
  (a) the $I$-processes $\{P_i^I \mid i \in dom(I)\}$, and
  (b) the moves $\{a_i^I \mid i \in dom(I)$ and $a_i^I \in P_i^I$ and $s{\uparrow}i = a_i^I.start\}$
(2) there is an edge from $P_i^I$ to every node of the form $a_i^I$ in $W_I(s)$, and
(3) there is an edge from $a_i^I$ to $P_j^I$ in $W_I(s)$ if and only if $i\,I\,j$ and $a_i^I \in W_I(s)$ and $s{\uparrow}ij(a_i^I.guard_j) = false$.

In the sequel, we use $P_i^I \longrightarrow a_i^I \in W_I(s)$ to denote the existence of an edge from $P_i^I$ to $a_i^I$ in $W_I(s)$, and $a_i^I \longrightarrow P_j^I \in W_I(s)$ to denote the existence of an edge from $a_i^I$ to $P_j^I$ in $W_I(s)$. We also abbreviate ($P_i^I \longrightarrow a_i^I \in W(s)$ and $a_i^I \longrightarrow P_j^I \in W(s)$) with $P_i^I \longrightarrow a_i^I \longrightarrow P_j^I \in W(s)$, and similarly for longer "wait-chains."

*Observation* 6.5.2.2 (*Move Enablement*). Let $s$ be an arbitrary $I$-state such that $s{\uparrow}i = a_i^I.start$. If $a_i^I$ has no outgoing edges in $W_I(s)$, then by the wait-for-graph definition (6.5.2.1), $s{\uparrow}ij(a_i^I.guard_j) = true$ for all $j \in I(i)$. Hence, by the compact $I$-structure definition (5.4.3), $a_i^I$ can be executed in state $s$.

6.5.3 *Supercycles.* We characterize a deadlock as the occurrence in the wait-for-graph of a graph-theoretic construct that we call a *supercycle*:

*Definition* 6.5.3.1 (*Supercycle*). $SC$ is a supercycle in $W_I(s)$ if and only if all of the following hold:

(1) $SC$ is nonempty,
(2) if $P_i^I \in SC$ then for all $a_i^I$ such that $a_i^I \in W_I(s)$, $P_i^I \longrightarrow a_i^I \in SC$, and
(3) if $a_i^I \in SC$ then there exists $P_j^I$ such that $a_i^I \longrightarrow P_j^I \in W_I(s)$ and $a_i^I \longrightarrow P_j^I \in SC$.

Note that this definition implies that $SC$ is a subgraph of $W_I(s)$.

If an $I$-process $P_i^I$ is in a supercycle $SC$, then every potential move of $P_i^I$ (i.e., every move out of the current local state of $P_i^I$) is also in $SC$ and is blocked by some other $I$-process $P_j$ that is also in $SC$. It follows that no $I$-process in $SC$ is enabled. Furthermore this situation is stable, since every $I$-process $P_i^I$ in $SC$ is blocked by a set of $I$-processes (the $I$-processes that each block at least one potential move of $P_i^I$) that are all in $SC$. Thus, no $I$-process in $SC$ can execute a transition, so the state of all these $I$-processes remains the same. Hence, the wait-for relationships defined by the supercycle persist forever.

Figure 12 shows the wait-for-graph $W_I(u^{dead})$ for the $I$-system of Figure 11 in its deadlocked state $u^{dead}$. And-nodes (processes) are shown as $\bullet$, while or-nodes (moves) are shown as $\circ$. $a_i, i \in \{1,2,3\}$ is the move of $P_i$ from $W_i$ to $A_i$, and $b_i, i \in \{1,2,3\}$ is the move of $P_i$ from $W_i$ to $B_i$. Since every move in $W_I(u^{dead})$ has at least one outgoing edge, i.e., is blocked by at least one process, $W_I(u^{dead})$ is also an example of a supercycle. In fact, several edges can be removed and still leave a supercycle (for example, $a_3 \longrightarrow P_1$, $b_3 \longrightarrow P_2$, $a_2 \longrightarrow P_1$ can all be removed). Thus, $W_I(u^{dead})$ contains several subgraphs that are also supercycles.

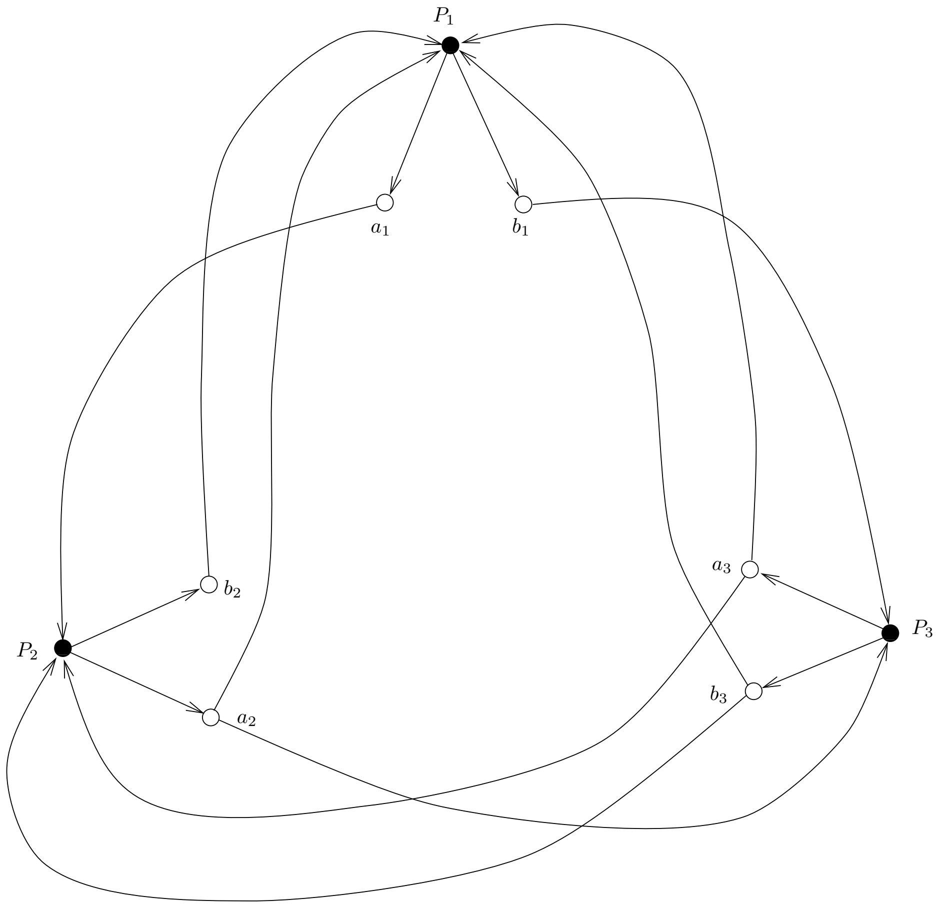



Fig. 12. Example wait-for-graph.

6.5.4 *Establishing Supercycle-Freedom: The Wait-for-Graph Assumption.* We shall establish deadlock freedom by showing that the wait-for-graph is always supercycle-free. We achieve this by making a technical assumption—the wait-for-graph assumption—which effectively precludes the formation of supercycles in the wait-for-graph. The essential idea is roughly as follows. The wait-for-graph assumption will allow us to conclude that in a state $s$ resulting from a transition by $P^I_k$, either $P^I_k$ has an enabled move, or $P^I_k$ does not block any move by any other process (in neither case can $P^I_k$ belong to a supercycle). Formally, this condition is

for every reachable $I$-state $t$ in $M_I$ such that $s \xrightarrow{k} t \in R_I$ for some reachable $I$-state $s$,

$$\neg \bigvee a^I_j \,.\, (a^I_j \longrightarrow P^I_k \in W_I(t)) \text{ or}$$
$$\bigvee a^I_k \in W_I(t) \,.\, (\bigwedge \ell \in \{i_1, \ldots, i_K\} \,.\, (a^I_k \longrightarrow P^I_\ell \notin W_I(t))). \qquad (**)$$

The first disjunct says that $P^I_k$ blocks no move by any other process, and the second disjunct says that $P_k^I$ has some move that is blocked by no process, i.e., some enabled move. Since our goal is to devise a completely mechanical synthesis method, the above condition must be mechanically checked. Doing so requires us, however, to generate $M_I$, which is, in general, exponentially large in the number of processes $K(=|dom(I)|)$. Instead, we take advantage of the state-mapping corollary (6.4.5), which relates the reachable states of $M_I$ to the reachable states of $M_J$ (for $J \subseteq I$), and the transition-mapping corollary (6.4.2), which relates $R_I$ to $R_J$ (for $J \subseteq I$), and the following proposition, which relates $W_I(s)$ to $W_J(s{\uparrow}J)$ (again for $J \subseteq I$), to formulate a condition over $M_J$ (for a suitably small $J$) that implies condition (**). This will be the wait-for-graph assumption. If $a_i^I = (s_i, \otimes_{j\in I(i)} \oplus_{\ell\in[1:n]} B_{i,\ell}^j \rightarrow A_{i,\ell}^j, t_i)$, then we let $a_i^J$ denote $(s_i, \otimes_{j\in J(i)} \oplus_{\ell\in[1:n]} B_{i,\ell}^j \rightarrow A_{i,\ell}^j, t_i)$.

PROPOSITION 6.5.4.1 (WAIT-FOR-GRAPH PROJECTION). *Let $J \subseteq I$ and $i\,J\,j$. Furthermore, let $s_I$ be an arbitrary $I$-state. Then*

*(1) $P_i^I \longrightarrow a_i^I \in W_I(s_I)$ iff $P_i^J \longrightarrow a_i^J \in W_J(s_I{\uparrow}J)$, and*
*(2) $a_i^I \longrightarrow P_j^I \in W_I(s_I)$ iff $a_i^J \longrightarrow P_j^J \in W_J(s_I{\uparrow}J)$.*

Now suppose that condition (**) is violated. Then, there is a transition $s \xrightarrow{k} t$ in $R_I$, where $s$ is a reachable $I$-state and where

$$\bigvee a_j^I \in W_I(t) \,.\, (a_j^I \longrightarrow P_k^I \in W_I(t))$$

and

$$\bigwedge a_k^I \in W_I(t) \,.\, (\bigvee \ell \in \{i_1, \dots, i_K\} \,.\, (a_k^I \longrightarrow P_\ell^I \in W_I(t))).$$

The first conjunct states that $P_k^I$ has some edge incoming from some node $a_j^I$. The second conjunct states that for every move node of the form $a_k^I$ in $W_I(t)$, there is some $I$-process $P_\ell^I$ that blocks $a_k^I$. Thus, if $n = |(t{\uparrow}k).moves|$, (i.e., $n$ is the number of move nodes of the form $a_k^I$ in $W_I(t)$), then by selecting one of the $I$-processes that blocks each move node of the form $a_k^I$, we can form a set $\{P_j^I, P_{\ell_1}^I, \dots, P_{\ell_n}^I\}$ of $I$-processes such that

$$a_j^I \longrightarrow P_k^I \in W_I(t)$$

and

$$\bigwedge a_k^I \in W_I(t) \,.\, (\bigvee \ell \in \{\ell_1, \dots, \ell_n\} \,.\, (a_k^I \longrightarrow P_\ell^I \in W_I(t))).$$

Now let $J = \{\{j,k\}, \{\ell_1,k\}, \dots \{\ell_n,k\}\}$. By the state-mapping corollary (6.4.5) $s{\uparrow}J$ is a reachable $J$-state of $M_J$, and by the transition-mapping corollary (6.4.2), $s{\uparrow}J \xrightarrow{k} t{\uparrow}J \in R_J$. Thus, by the wait-for-graph projection proposition (6.5.4.1), there is a transition $s{\uparrow}J \xrightarrow{k} t{\uparrow}J$ in $R_J$, where $s{\uparrow}J$ is a reachable $J$-state and where

$$a_j^J \longrightarrow P_k^J \in W_J(t{\uparrow}J)$$

and

$$\bigwedge a_k^J \in W_J(t{\uparrow}J) \,.\, (\bigvee \ell \in \{\ell_1, \dots, \ell_n\} \,.\, (a_k^J \longrightarrow P_\ell^J \in W_J(t{\uparrow}J))). \qquad (***)$$

It follows that if condition (***) does not hold for any transition of $R_J$ whose initial state is reachable, then (**) cannot be violated. This leads us to the following formulation of the wait-for-graph assumption.

*Definition* 6.5.4.2 (*Wait-For-Graph Assumption*: *WG*). Let $t_k$ be an arbitrary reachable local state of $P_k^\ell$ in $M_{k\ell}$, and let $n = |t_k.moves|$. Also let $J$ be an arbitrary interconnection relation of the form $\{\{j,k\}, \{k,\ell_1\}, \ldots, \{k,\ell_n\}\}$, where $k \notin \{j,\ell_1,\ldots,\ell_n\}$. Then, for every reachable $J$-state $t_J$ in $M_J$ such that $t_J{\uparrow}k = t_k$ and $s_J \xrightarrow{k} t_J \in R_J$ for some reachable $J$-state $s_J$, we have

$$\bigwedge a_j^J \,.\, (a_j^J \longrightarrow P_k^J \notin W_J(t_J))$$

or

$$\bigvee a_k^J \in W_J(t_J) \,.\, (\bigwedge \ell \in \{\ell_1,\ldots,\ell_n\} \,.\, a_k^J \longrightarrow P_\ell^J \notin W_J(t_J)).$$

Note that, in the above discussion, $\ell_1,\ldots,\ell_n$ are not necessarily pairwise distinct, i.e., an $I$-process $P_\ell^I$ could block more than one move of the form $a_k^I$. Thus there are $n$ cases for the form of $J$, as the number $m$ of distinct members of $\{\ell_1,\ldots,\ell_n\}$ varies from 1 to $n$. All of these cases must be considered. Likewise, the above discussion does not specify whether $j$ is a member of $\{\ell_1,\ldots\ell_n\}$ or not. Thus both cases must be considered, which in effect generates two subcases for each of the $n$ cases outlined above. Appendix E presents an algorithm for mechanically checking the wait-for-graph assumption. We now use the wait-for-graph assumption to establish supercycle-freedom of the wait-for-graph of any reachable state.

THEOREM 6.5.4.3 (SUPERCYCLE-FREE WAIT-FOR-GRAPH). *If the wait-for-graph assumption $WG$ holds, and $W_I(s_I^0)$ is supercycle-free for every initial state $s_I^0 \in S_I^0$, then for every reachable state $t$ of $M_I$, $W_I(t)$ is supercycle-free.*

6.5.5 *Establishing Deadlock-freedom.* We show that the absence of supercycles in the wait-for-graph of a state implies that there is at least one enabled move in that state.

PROPOSITION 6.5.5.1 (SUPERCYCLE). *If $W_I(s)$ is supercycle-free, then some move $a_i^I$ has no outgoing edges in $W_I(s)$.*

PROOF. We establish the contrapositive. Since every local state of a process has at least one outgoing arc (Section 2), there exists at least one move of the form $a_i^I$ for every $i \in dom(I)$ in $W_I(s)$. Suppose that every such move has at least one outgoing edge in $W_I(s)$. Consider the subgraph $SC$ of $W_I(s)$ consisting of these edges together with all edges of the form $P_i^I \longrightarrow a_i^I$ in $W_I(s)$. By the wait-for-graph definition (6.5.2.1), and the supercycle definition (6.5.3.1), it is clear that $SC$ is a supercycle in $W_I(s)$. Thus $W_I(s)$ is not supercycle-free. □

Finally, we apply the supercycle proposition (6.5.5.1) to all of the reachable states of $M_I$, thereby establishing the deadlock freedom of $(S_I^0, P_{i_1}^I \| \ldots \| P_{i_K}^I)$.

THEOREM 6.5.5.2 (DEADLOCK FREEDOM). *If, for every reachable state $s$ of $M_I$, $W_I(s)$ is supercycle-free, then $M_I, S_I^0 \models AGEXtrue$.*

PROOF. Let $s$ be an arbitrary reachable state of $M_I$. By the antecedent, $W_I(s)$ is supercycle-free. Hence, by the supercycle proposition (6.5.5.1), some move $a_i^I$ has no outgoing edges in $W_I(s)$. By Observation 6.5.2.2, $a_i^I$ can be executed in state $s$. Since $s$ is an arbitrary reachable state of $M_I$, we conclude that every reachable state of $M_I$ has at least one enabled move $a_i^I$, (where, in general, $a_i^I$ depends on $s$). Hence $M_I, S_I^0 \models AGEXtrue$. □

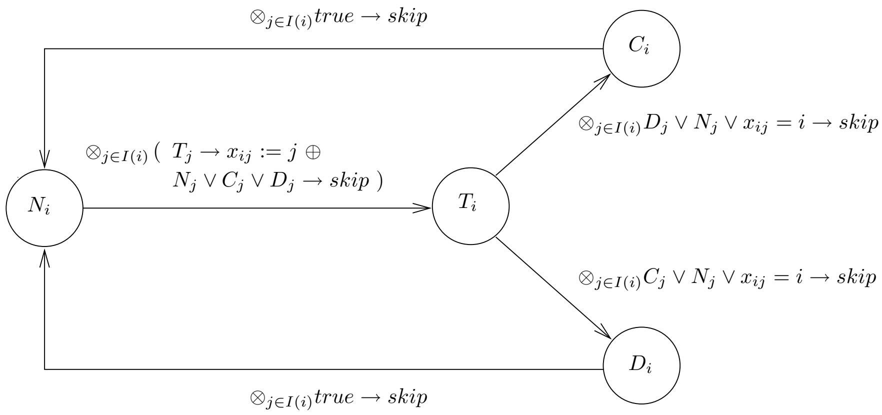



Fig. 13. Mutual exclusion example with two critical sections.

6.5.6 *Examples.* We now turn to some examples. Consider the solution for the generalized dining philosophers problem given in Figure 7. We verify that this solution satisfies the wait-for-graph assumption. The pair-process $P_k^\ell$ from which the solution in Figure 7 is derived has three local states, which, by slight abuse of terminology, we refer to as $[N_k]$, $[T_k]$, and $[C_k]$. We deal with each local state in turn.

For the move from $[N_k]$ to $[T_k]$, we observe, by inspection, that $P_k^I$ blocks no move by any other process in the resulting $I$-state. For the moves from $[T_k]$ to $[C_k]$ and $[C_k]$ to $[N_k]$, we see that, in both cases, $P_k^I$ has an enabled move in the resulting $I$-state.

Having checked $WG$ for every (reachable) local state of $P_k^\ell$, we conclude that the solution of Figure 7 satisfies $WG$ (note that the above analysis takes care of both the subcases $j = \ell_1, j \neq \ell_1$). It remains to show that $W_I(s_I^0)$ is supercycle-free for every initial state $s_I^0$. In an initial state, all processes are in their noncritical ("$[N]$") state, and no process blocks a move of any other process. Hence there can be no supercycles. Therefore, by the above results, we infer that every $I$-system corresponding to the solution in Figure 7 is deadlock-free.

Furthermore, consider the generalized example given in Figure 13 where there are now two critical sections: $[C_i]$ and $[D_i]$. Processes request entry into either of these critical sections, after which they return to their noncritical sections. When $I$ is a "complete graph," this example (which can clearly be generalized to an arbitrary number of critical sections) models the situation where we have some number of identical resources and where processes require exactly one of these resources. Such a resource could be a printer, for example, where a process is satisfied by acquiring any one of some number of printers. Again we can verify by simply checking all the possible moves of $P_k^J$ that this example satisfies the wait-for-graph assumption, and therefore, all the $I$-systems generated from it are deadlock-free.

Looking back at the system of Figure 10, we see that it violates the wait-for-graph assumption in that it allows transitions by a process such that in the resulting state, it blocks a move of some other process, and has all its own moves blocked by other processes. The transition by $P_3$ in the path displayed in Section 6.5.1 is an example of such a transition: in the resulting state $u^{dead}$, $P_3$ blocks the move $(W_1, b^1_{12} = 1 \rightarrow skip \otimes b^1_{13} = 1 \rightarrow skip, B_1)$ of $P_1$, while both its own moves are blocked by $P_2$ (and $P_1$, for that matter).

## 6.6 The Large-Model Theorem

As stated in the introduction, a crucial aspect of MP-synthesis is its soundness: what correctness properties of the pair-system are preserved by the $I$-system? Here, we treat the special case of complete interconnection: the process interconnection relation $I$ is the "complete graph" on $\{i_1, \ldots, i_K\}$. The general case is treated below in Section 6.8. Our soundness result for the case of complete interconnection is expressed as the large-model theorem. We show that propositional invariants expressed pairwise, temporal leads-to properties over individual processes, and local structure properties are all preserved (see Section 6.1). To facilitate the formal statement of our results, we define the sublogic "Flat CTL" (FLCTL) of CTL.

*Definition* 6.6.1 (*Flat CTL*). The logic $\mathrm{FLCTL}_{k\ell}$ is the set of state formulae that are finite conjunctions of the following, where $h_{k\ell}$ is a formula of $\mathcal{L}(\mathcal{AP}_k, \mathcal{AP}_\ell, \neg, \wedge)$, and $p_k, q_k, r_k$ are formulae of $\mathcal{L}(\mathcal{AP}_k, \neg, \wedge)$:

—$h_{k\ell}$
—$AGh_{k\ell}$
—$AG(p_k \Rightarrow A[q_k U r_k])$
—$AG(p_k \Rightarrow AY_k q_k)$
—$AG(a_k \Rightarrow EX_k b_k))$, where $a_k, b_k \in \{\{\!|s_k|\!\} \mid s_k \in S_k\}$

In terms of our notation for sublogics (Section 3.2), $\mathrm{FLCTL}_{k\ell}$ is the sublogic $\mathcal{L}(\wedge, h_{k\ell}, AGh_{k\ell}, AG(p_k \Rightarrow A[q_k U r_k]), AG(p_k \Rightarrow AY_k q_k), AG(a_k \Rightarrow EX_k b_k))$.

THEOREM 6.6.2 (LARGE MODEL). *Let $f_{k\ell}$ be an arbitrary formula of $FLCTL_{k\ell}$. Let $I = \{i_1, \ldots, i_K\} \times \{i_1, \ldots, i_K\} - \{(i,i) \mid i \in \{i_1, \ldots, i_K\}\}$. Furthermore, let $s$ be an arbitrary reachable $I$-state, and, for all $i, j$ such that $i\,I\,j$, let $s_{ij} = s{\uparrow}ij$. If $M_I, S^0_I \models AGEXtrue$, then*

$$\bigwedge(i,j) \in I \,.\, (M_{ij}, s_{ij} \models \textstyle\bigwedge_{k\ell} f_{k\ell}) \textit{ implies } M_I, s \models \textstyle\bigwedge_{k\ell} f_{k\ell}.$$

Note that the formula satisfied by the $I$-system depends (by MPCTL* semantics) on $I$ itself, in that the quantification over pairs given by $\bigwedge_{k\ell}$ has as its range all pairs in $I$. Thus, we cannot directly conclude, from the large-model theorem, any correctness property that relates two processes which are not $I$-neighbors. Such a property could, however, be a consequence of pairwise expressible properties that relate, say, a common neighbor of the non-neighboring pair (thereby imposing some indirect interaction on the pair). The investigation of methods for deducing such indirect correctness properties is outside the scope of this article and is a topic of future research.

Our notion of a correctness property is one that holds in all the initial states of the pair-system or $I$-system (Section 6.1). The following corollary presents the soundness result in terms of such correctness properties.

COROLLARY 6.6.3 (LARGE MODEL). *Let $f_{k\ell}$ be an arbitrary formula of $FLCTL_{k\ell}$. Let $I = \{i_1, \ldots, i_K\} \times \{i_1, \ldots, i_K\} - \{(i,i) \mid i \in \{i_1, \ldots, i_K\}\}$, and let $(i,j)$ be an arbitrary pair in $I$. If $M_I, S_I^0 \models AGEXtrue$, then*

$$M_{ij}, S_{ij}^0 \models \textstyle\bigwedge_{k\ell} f_{k\ell} \textit{ implies } M_I, S_I^0 \models \textstyle\bigwedge_{k\ell} f_{k\ell}.$$

### 6.7 Liveness Properties

We wish to generalize the large-model theorem by dropping the assumption that the process interconnection scheme is a complete graph. This enables us to deal with, for example, the dining philosophers and readers-writers problems. Unfortunately, generalizing the interconnection relation $I$ to arbitrary schemes introduces the possibility that liveness properties may be violated. In this article, we restrict our attention to the correctness properties expressible in Flat CTL (Definition 6.6.1). The only liveness property expressible in Flat CTL is $AG(p_i \Rightarrow A[q_i U r_i])$, where $p_i, q_i, r_i$ are propositional formulae over the atomic propositions of $\mathcal{AP}_i$. Now $AG(p_i \Rightarrow A[q_i U r_i])$ is equivalent to $AG(p_i \Rightarrow A[q_i U_w r_i]) \wedge AG(p_i \Rightarrow AFr_i)$. Since $AG(p_i \Rightarrow A[q_i U_w r_i])$ is not a liveness property, in the sequel we will use $AG(p_i \Rightarrow AFr_i)$ as our prototypical liveness property. A liveness property of this form is sometimes referred to as a "temporal leads-to" property in the literature, since its intuitive meaning is "if $p_i$ is true in some global state, then $r_i$ is guaranteed to eventually be true along every fullpath starting in this state." In other words, $p_i$ "leads to" $r_i$. The formula $AFr_i$ is an *eventuality* formula. Suppose $M_{ij}, s_{ij} \models AFr_i$ for some $ij$-state $s_{ij}$. By CTL* semantics, we have the equivalence $AFr_i \equiv r_i \vee AXAFr_i$. Thus, if $s_{ij} \not\models r_i$, then we have $M_{ij}, s_{ij} \models AXAFr_i$, i.e., for every successor state $t_{ij}$ of $s_{ij}$, we have $M_{ij}, t_{ij} \models AFr_i$. Thus, the eventuality $AFr_i$ has been *propagated* to the successors of $s_{ij}$. This propagation continues as long as $\neg r_i$ holds. In this case we say that $AFr_i$ is *pending* at state $s_{ij}$. In other words, $AFr_i$ is a pending eventuality of $s_{ij}$ if and only if $r_i$ is not true in $s_{ij}$, but is guaranteed to eventually become true along every fullpath starting in $s_{ij}$. When an $ij$-state is reached where $r_i$ is true, $AFr_i$ is said to be *fulfilled*. Note that if $s_{ij} \models r_i$, then $M_{ij}, s_{ij} \models AFr_i$ follows immediately, and $AFr_i$ is fulfilled in $s_{ij}$ itself.

In an $I$-system, pending eventualities may be prevented from being fulfilled in two ways. The first way is due to the particular blocking behavior inherent in the pair-system that is input to MP-synthesis. The solution here is to make a technical assumption on the pair-system—the liveness assumption—that prevents this kind of blocking behavior. The second way is when the $I$-system is "dynamically partitioned" into two subsystems such that, from some point onward, no process in one subsystem ever blocks a move of any process in the other subsystem. In this case, it is possible to schedule processes only from one of the subsystems. We deal with this by requiring the use of fair scheduling, and define the particular fairness that we require—weak blocking fairness—below. We establish a progress lemma that states (roughly) that under weak-blocking fairness, and assuming the liveness-assumption, every $I$-process that must progress in order to fulfill a pending eventuality actually does progress. This lemma is crucial in establishing the generalized large-model theorem (Section 6.8), which in turn allows us to conclude that MP-synthesis preserves temporal leads-to properties (e.g., if a philosopher requests entry to the critical section, it eventually gains entry).

6.7.1 *Example of Nonfulfillment of Pending Eventualities Due to Blocking Behavior.* Consider the $I$-system synthesized using the interconnection relation $I = \{\{1,2\},\ \{2,3\},\ \{3,4\},\ \{4,5\}\}$ and the pair-structure given in Figure 14. Suppose $\bigwedge_k AG(D_k \Rightarrow AFE_k)$ is a conjunct of the problem specification.[12] It is easily seen by inspection that $M_{ij}, S^0_{ij} \models \bigwedge_k AG(D_k \Rightarrow AFE_k)$. We show however, that $M_I, S^0_I \not\models \bigwedge_k AG(D_k \Rightarrow AFE_k)$. In particular, consider the following path $\pi$ in $M_I$, which starts in the initial state $[A_1\, A_2\, A_3\, A_4\, A_5]$:

$$
\begin{array}{l}
[A_1\, A_2\, A_3\, A_4\, A_5] \xrightarrow{1} [B_1\, A_2\, A_3\, A_4\, A_5] \xrightarrow{5} [B_1\, A_2\, A_3\, A_4\, B_5] \xrightarrow{5} \\
[B_1\, A_2\, A_3\, A_4\, X_5] \xrightarrow{4} [B_1\, A_2\, A_3\, B_4\, X_5] \xrightarrow{3} [B_1\, A_2\, B_3\, x_{34}{=}4\, B_4\, X_5] \xrightarrow{2} \\
[B_1\, x_{12}{=}1\, B_2\, x_{23}{=}3\, B_3\, x_{34}{=}4\, B_4\, X_5] \xrightarrow{1} [C_1\, x_{12}{=}1\, B_2\, x_{23}{=}3\, B_3\, x_{34}{=}4\, B_4\, X_5] \xrightarrow{1} \\
(\ [D_1 x_{12}{=}1\, B_2\, x_{23}{=}3\, B_3\, x_{34}{=}4\, B_4\, X_5] \xrightarrow{5} \\
\quad [D_1 x_{12}{=}1\, B_2\, x_{23}{=}3\, B_3\, x_{34}{=}4\, B_4\, Y_5] \xrightarrow{5}\ )^\omega
\end{array}
$$

Let $s = [D_1 x_{12} = 1\, B_2\, x_{23} = 3\, B_3\, x_{34} = 4\, B_4\, X_5]$, and $t = [D_1 x_{12} = 1\, B_2\, x_{23} = 3\, B_3\, x_{34} = 4\, B_4\, Y_5]$. Clearly, $\pi \models \neg FE_1$, so $M_I, s \models \neg AFE_1$. But $s \models D_1$, so $M_I, s \models \neg(D_1 \Rightarrow AFE_1)$. Since $s$ is a reachable state, we have $M_I, S^0_I \not\models AG(D_1 \Rightarrow AFE_1)$. Hence, by MPCTL semantics, we finally conclude $M_I, S^0_I \not\models \bigwedge_k AG(D_k \Rightarrow AFE_k)$.

6.7.2 *The Liveness Assumption.* The standard solution to the problem of ensuring that all pending eventualities are fulfilled is to adopt some form of fair scheduling notion [Emerson and Lei 1985; Francez 1986]. However, in this situation, no asymptotic fairness notion can help. Let $\rho$ be the suffix $(s \xrightarrow{5} t)^\omega$ of $\pi$. By inspection, we see that only $P_5$ is enabled in any state along $\rho$. Furthermore, every state of $\rho$ has only a single successor, namely the next state along $\rho$, i.e., once the first state of $\rho$ has been entered, it is not possible to reach any state of $M_I$ other than a state along $\rho$. Thus, no asymptotic fairness notion can preclude $\rho$ as a possible computation. One would need a fairness notion which incorporated "lookahead" and prevented execution from reaching a state lying on an "unfair" path such as $\rho$. Since such fairness notions have not been investigated thoroughly, and in any case seem to be expensive to implement, we avoid this approach. Our only apparent alternatives are to make suitable assumptions on the structure of $I$ and/or $M_{ij}$.

Since the whole point is to generalize the large-model theorem so that it applies to an arbitrary interconnection relation $I$, we avoid making any restrictions on $I$. Thus, we try to restrict the structure of $M_{ij}$ in some way. Now the general situation which the above example illustrates is where $I = \{\ \{i, j_1\},\ \{j_1, j_2\},\ \ldots,\ \{j_{n-1}, j_n\},\ \{j_n, k\}\ \}$ (for arbitrary $n$), and there is a reachable state $s$ of $M_I$ such that

(1) an eventuality of the form $AFr_i$ is pending in $s$ and
(2) there is a wait-chain from $P^I_i$ to $P^I_k$, and only $P^I_k$ can be executed from some point onward.

[12] Since $AFE_k$ is an abbreviation for $A[trueUE_k]$, this is within the fragment of CTL specifications that are preserved by our method.

We require an assumption that prevents the occurrence of such a state, i.e., all reachable states must satisfy the negation of one (or both) of the two clauses given above. We use the following:

$$M_{j_1k}, s{\uparrow}j_1k \not\models EGex_k \text{ or } \bigwedge a_i^I \,.\, (P_i^I \longrightarrow a_i^I \longrightarrow P_{j_1}^I \notin W_I(s)).$$

This is the negation of clause 2 for $I = \{\{i, j_1\}, \{j_1, k\}\}$ (where $i = k$ and $i \neq k$ are both possible, i.e., both cases must be considered). Since clause 2 implicitly universally quantifies over all $I$ of the given form $(\{\{i, j_1\}, \{j_1, j_2\}, \ldots, \{j_{n-1}, j_n\}, \{j_n, k\}\})$, it suffices to assume the negation for a subset of the possible values for $I$. It is imperative to use a small $I$ to make the mechanical evaluation of our assumption tractable, since this evaluation requires $M_I$ to be constructed, and $|M_I|$ grows exponentially with $|I|$. We formally define the required condition as the "liveness assumption." Note that the term $P_i^I \longrightarrow a_i^I$ can be dropped without changing the meaning, since $P_i^I \longrightarrow a_i^I \notin W_I(s)$ implies $a_i^I \longrightarrow P_{j_1}^I \notin W_I(s)$.

*Definition* 6.7.2.1 (*Liveness Assumption*: *LV*). Let $J$ be of the form $\{\{i, j\}, \{j, k\}\}$ where $j \notin \{i, k\}$. Then, for every reachable state $s_J$ in $M_J$,

$$M_{jk}, s_J{\uparrow}jk \models EGex_k \text{ implies } \bigwedge a_i^J \,.\, (a_i^J \longrightarrow P_j^J \notin W_J(s_J)).$$

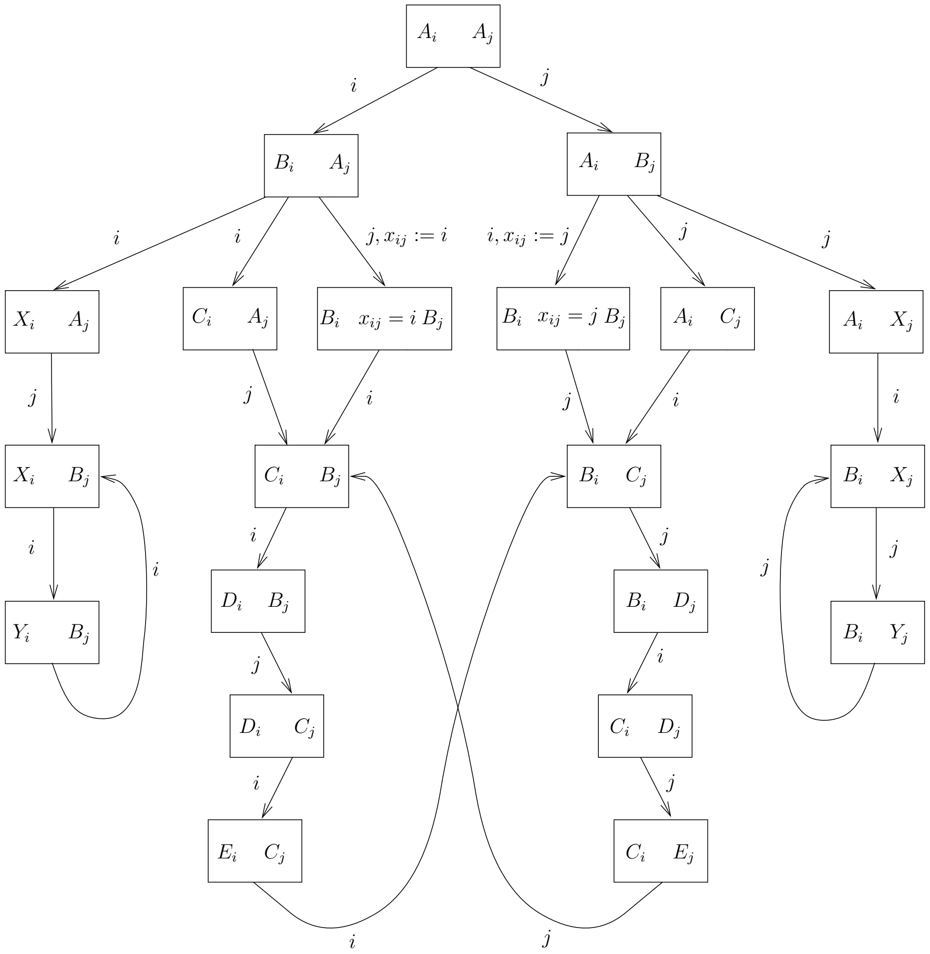
Appendix E shows how the liveness assumption can be mechanically checked using CTL model-checking [Clarke et al. 1986].

6.7.3 *Fairness.* Notwithstanding our comments above about the futility of using asymptotic fairness to deal with the example in Figure 14, we nevertheless demonstrate that some form of fairness is essential if we are to avoid restrictions on the interconnection relation $I$.[13] Suppose $I$ is partitioned into subrelations $I'$ and $I''$ such that no process in (the domain of) $I'$ is $I$-related to a process in the domain of $I''$, and vice-versa. Thus $I'$ and $I''$ represent "uncoupled" systems. Assuming both systems are deadlock-free, it is possible, under nondeterministic scheduling, to always select processes from $I'$ for execution, thereby causing nonfulfillment of pending eventualities of processes in $I''$. Even if $I$ is not partitioned in this way, it is possible for such uncoupled subsets of $I$ to arise dynamically during particular executions. As a concrete example of this, consider the system generated by MP-synthesis for the dining philosophers specification given in Section 3.4, i.e., five philosophers $P_1, P_2, P_3, P_4, P_5$ arranged in a ring. Consider the following path $\pi$, which starts in the initial state $s^0 = [N_1\, N_2\, N_3\, N_4\, N_5]$:

$$[N_1\, N_2\, N_3\, N_4\, N_5] \xrightarrow{1} (\,[T_1\, N_2\, N_3\, N_4\, N_5] \xrightarrow{3} [T_1\, N_2\, T_3\, N_4\, N_5] \xrightarrow{3} [T_1\, N_2\, C_3\, N_4\, N_5] \xrightarrow{3}\,)^\omega$$

Under purely nondeterministic scheduling $\pi$ is an initialized fullpath of (the $I$-structure of) the solution produced by MP-synthesis for the above problem. (For ease of notation, we have omitted the values of the shared variables in $\pi$. The reader can easily ascertain, by inspecting Figure 6, that appropriate values exist.) Now $\pi$ does not satisfy $G(T_1 \Rightarrow FC_1)$, and hence $s^0$ does not satisfy $AG(T_1 \Rightarrow AFC_1)$, which is a conjunct of the dining philosophers specification. The problem here is that $P_1$ is enabled, but fails to make progress (and fulfill the pending eventuality

[13] With the exception of our assumption (see Section 3.3) that every process in $dom(I)$ is $I$-related to at least one other process.




The initial state set is $\{[A_i\, A_j]\}$.

Fig. 14. Pair-structure illustrating nonfulfillment of a pending eventuality.

$AFC_1$) because it is not scheduled for execution. Since $P_1$ has become "decoupled" from $P_3$ (due to the other philosophers remaining permanently in their neutral ($N$) states), it is possible, under nondeterministic scheduling, for $P_3$ to be scheduled for execution continuously in spite of $P_1$ being continuously enabled. Note that this scenario cannot arise when $I$ is a complete graph, because then all pairs of processes would be neighbors, so no pair of processes could ever become decoupled in the manner illustrated above.

In the dining philosophers system, fairness would eventually require that $P_1$ be selected for execution, and thus the path $\pi$ exhibited above would no longer be a possible computation. Let $en_i, ex_i$ mean that $P_i$ is enabled in a state, executed along a transition, respectively (note that $en_i$ is rendered as $EX_i true$ in CTL$^*$).

The three most common fairness notions are

—unconditional fairness: $\bigwedge_{i \in dom(I)} \overset{\infty}{F} ex_i$

—weak fairness: $\bigwedge_{i \in dom(I)} \overset{\infty}{G} en_i \Rightarrow \overset{\infty}{F} ex_i$

—strong fairness: $\bigwedge_{i \in dom(I)} \overset{\infty}{F} en_i \Rightarrow \overset{\infty}{F} ex_i$

Unconditional fairness is undesirable, since it requires that every process be infinitely often enabled, and this may not be the case, in general. We therefore adopt (a form of) weak fairness that is sufficient for our needs, and easier to implement than strong fairness. In the previous example, under weak fairness, $P_1$ is eventually executed, as desired. Unfortunately, $P_2$ is also eventually executed. This is not so desirable, as $P_2$ is in its noncritical state, and therefore weak fairness is forcing $P_2$ out of its noncritical state and into its trying state, i.e., it is forcing $P_2$ to make a request for critical section entry. In this example, we want the fairness to apply only to the trying states, where a request for critical section entry has already been made, and not to noncritical states, where it forces "spurious" requests. In addition, fairness must also apply to the critical states, in order to avoid indefinitely tying up critical resources, (which would prevent other processes from gaining access), and thereby guarantee that all eventualities are met. In the dining philosophers example, if fair scheduling is not applied to $P_i$ in state $s$ such that $s{\uparrow}i = C_i$, $s{\uparrow}(i+1) = T_{i+1}$, then (even though $s$ has no pending eventuality for $P_i$) the pending eventuality of $AFC_{i+1}$ for $P_{i+1}$ might never be fulfilled, as $P_i$ might remain forever in $C_i$, preventing the entry of $P_{i+1}$ into $C_{i+1}$.

Generalizing this discussion to an arbitrary MPCTL* specification and the $I$-system derived from it, suppose that $i\,I\,j$ and $j\,I\,k$, and there is some reachable $I$-state $s$ in $M_I$ such that

—$P^I_j$ has a pending eventuality in $s$,

—in $j$-state $s{\uparrow}j$, $P^I_j$ has two possible moves $a^I_j$ and $b^I_j$ (so $a^I_j.start = s{\uparrow}j$ and $b^I_j.start = s{\uparrow}j$),

—$s{\uparrow}ij \not\models a^I_j.guard_i$, i.e., $a^I_j$ is blocked by $P^I_i$, and

—$s{\uparrow}jk \not\models b^I_j.guard_k$, i.e., $b^I_j$ is blocked by $P^I_k$.

Clearly, if fair scheduling is not applied to $P^I_i$ and $P^I_k$, then $P^I_j$'s pending eventuality may never be fulfilled. Note however, that $s{\uparrow}ij \models b^I_j.guard_i$ and $s{\uparrow}jk \models a^I_j.guard_k$ are both consistent with the previous example, i.e., it is not necessary for any one neighbor of $P^I_j$ to block all the possible moves of $P^I_j$ for $P^I_j$ to be blocked. We must therefore apply fair scheduling to every $I$-process that is blocking at least one move of one of its neighbors. Furthermore we wish our fairness notion to depend only on the local state ($s{\uparrow}i$) of processes, and not on the shared variables ($\mathcal{SH}_{ij}$, for $i\,I\,j$) or the local state $s{\uparrow}j$ of any neighbor $P^I_j$. This makes the implementation of the fairness easier, since, for $P^I_i$, only its local state $s{\uparrow}i$ need be detected, rather than $\bigcup_{j \in I(i)} s{\uparrow}ij$ if shared variables and the local states of neighbors were included in the fairness notion. The latter involves the combined shared state over all of $P^I_i$'s neighbors. Detecting this combined state instantaneously is quite expensive, and we avoid the problem by formulating a fairness notion over local states only.

Hence, we must apply fair scheduling to every $i$-state $s_i$ that is the $P_i$-component of some $I$-state $s$ such that $s{\uparrow}ij \not\models a_j^I.guard_i$ for some move $a_j^I$ of $P_j^I$. Note that since $a_j^I.guard_i$ depends on the shared variables as well as the current $i$-state, there may be other $I$-states $t$ also having $P_i$-component $s_i$ where $t{\uparrow}ij \models a_j^I.guard_i$. Thus, in some cases, the fair scheduling is not really necessary, i.e., our fairness is suboptimal. Making our fairness notion optimal would require instantaneous detection of the shared state with all $I$-neighbors, which, as mentioned above, would be quite expensive. Thus, we sacrifice some optimality in fair scheduling in return for a fairness notion that can be implemented reasonably efficiently.

Consider a move $a_j^I$ of $P_j^I$. By the compact MP-synthesis definition (5.4.1), $a_j^I$ has the form $(s_i, \otimes_{k\in I(j)} \oplus_{\ell\in[1:n]} B_{i,\ell}^j \rightarrow A_{i,\ell}^j, t_i)$. Hence, $a_j^I.guard_i = a_j^i.guard$. Hence, $s{\uparrow}ij \not\models a_j^I.guard_i$ if and only if $s{\uparrow}ij \not\models a_j^i.guard$. By the state-mapping corollary (6.4.5), we know that $s{\uparrow}ij$ is reachable in $M_{ij}$ if $s$ is reachable in $M_I$. Hence, we conclude that $s{\uparrow}ij \not\models a_j^I.guard_i$ can be evaluated by constructing $M_{ij}$ only (and not $M_I$). This discussion leads us to the following definition.

*Definition* 6.7.3.1 (*Sometimes-Blocking State*, $blk_i$). An $i$-state $s_i$ is *sometimes-blocking* if and only if:

$$\bigvee s_{ij}^0 \in S_{ij}^0 \,.\, (M_{ij}, s_{ij}^0 \models EF(\ \{\!\!\{s_i\}\!\!\} \wedge (\bigvee a_j^i \in P_j^i \,.\, (\{\!\!\{a_j^i.start\}\!\!\} \wedge \neg a_j^i.guard))\ )).$$

Note that $\bigvee a_j^i \in P_j^i \,.\, (\{\!\!\{a_j^i.start\}\!\!\} \wedge \neg a_j^i.guard)$ is a purely propositional formula.

$$blk_i \stackrel{\text{df}}{=} (\bigvee\ \{\!\!\{s_i\}\!\!\} : s_i \text{ is sometimes-blocking}).$$

Thus, $s_i$ is sometimes-blocking if and only if some (not necessarily all) state $s_{ij}$ in $M_{ij}$ blocks some move $a_j^i$ of $P_j^i$ and has $s_i$ as its $P_i$-component. Note that it is entirely possible for $M_{ij}$ to contain another state $t_{ij}$ such that $t_{ij}{\uparrow}i = s_i$ and $t_{ij}$ does not block $a_j^I$. Thus, being in a sometimes-blocking state does not imply that a neighbor's move is blocked: it only indicates (as the name suggests) the possibility that a neighbors move is blocked. Also, being in a state that is not sometimes-blocking implies that no move of any neighbor is blocked. $blk_i$ is a propositional formula (over the propositions in $\mathcal{AP}_i$) which is true in $ij$-state $s_{ij}$ (respectively, global state $s$) exactly when $s_{ij}{\uparrow}i$ (respectively, $s{\uparrow}i$) is sometimes-blocking. If an $i$-state $s_i$ is not sometimes-blocking, we say that it is *always-nonblocking*, and we define $nblk_i \stackrel{\text{df}}{=} \neg blk_i$.

For example, in the pair-structure (of the two-process mutual exclusion system) of Figure 4, $[N_1]$ is an always-nonblocking state: it occurs in three global states, namely $[N_1\, N_2]$, $[N_1\, T_2]$, and $[N_1\, C_2]$, and all these states have an outgoing $P_2^1$-transition. On the other hand, $[T_1]$ is a sometimes-blocking state, since the global state $[T_1\, x_{12} = 1\, T_2]$ has no outgoing $P_2^1$-transition (but note that the global state $[T_1\, x_{12} = 2\, T_2]$ does have an outgoing $P_2^1$-transition). $[C_1]$ is also a sometimes-blocking state, since $[C_1\, T_2]$ has no outgoing $P_2^1$-transition.

*Observation* 6.7.3.2. By Definition 6.7.3.1, if $s_i$ is always-nonblocking, then $M_{ij}, S_{ij}^0 \models AG(\{\!\!\{s_i\}\!\!\} \Rightarrow \bigwedge a_j^i \in P_j^i \,.\, (\{\!\!\{a_j^i.start\}\!\!\} \Rightarrow a_j^i.guard))$. This implies that the wait-for-graph $W_I(s)$ contains no edge whose target is $P_i^I$ when $s{\uparrow}i$ is always-nonblocking.

Since we want fairness to apply only to sometimes-blocking states, we relativize weak fairness to the sometimes-blocking states by defining our fairness notion $\Phi$ as a CTL* path formula as follows:

*Definition* 6.7.3.3 (*Weak Blocking Fairness* $\Phi$).

$$\Phi \stackrel{\mathrm{df}}{=} \bigwedge_{i \in dom(I)} \overset{\infty}{G}(blk_i \wedge en_i) \Rightarrow \overset{\infty}{F} ex_i.$$

We saw above that in the pair-structure (of the two-process mutual exclusion system) of Figure 4, $T_1$ and $C_1$ are the sometimes-blocking states. Hence, for the $I$-system (Figure 7) synthesized from the two-process mutual exclusion system (Figure 1), weak blocking fairness is: $\bigwedge_{i \in dom(I)} \overset{\infty}{G}((T_i \vee C_i) \wedge en_i) \Rightarrow \overset{\infty}{F} ex_i$. In other words, processes that remain enabled and in their trying or critical sections continuously must eventually be executed (note that requiring execution "eventually" or "infinitely often" makes no difference).

We denote the use of weak blocking fairness in our technical results by relativizing our notion of satisfaction of CTL* formulae ($\models$) so that only fullpaths that satisfy weak blocking fairness are taken into account. The resulting notion of satisfaction is denoted by $\models_\Phi$, and is defined by Emerson and Lei [1985] (for the more traditional weak- and strong-fairness notions):

$$M, s \models_\Phi Af \text{ iff } M, s \models A(\Phi \Rightarrow f)$$
$$M, s \models_\Phi Ef \text{ iff } M, s \models E(\Phi \wedge f)$$

Effectively path quantification is only over the paths that satisfy $\Phi$.

The following proposition is a consequence of the definition of sometimes-blocking. The essential idea is that if $M_{ij}, s_{ij} \models \neg r_i \wedge AFr_i$, then along every path leaving $s_{ij}$, $P_i^j$ must eventually be executed, since the current $i$-state must eventually change for $r_i$ (which depends only on the current $i$-state) to go from $false$ to $true$. If $s_i$ is not sometimes-blocking (i.e., always-nonblocking), then there will be at least one fullpath leaving $s_{ij}$ that consists entirely of $P_j^i$ transitions. This is because every $P_j^i$ transition from $s_{ij}$ to, say $s'_{ij}$, leaves the current $i$-state unchanged, i.e., still always-nonblocking. This implies that there is still (another) enabled $P_j^i$-transition from $s'_{ij}$. Repeating this argument, we conclude the existence of the aforementioned fullpath.

PROPOSITION 6.7.3.4 (SOMETIMES-BLOCKING). *Let $s_{ij}$ be a reachable state of $M_{ij}$ and $r_i \in \mathcal{L}(\mathcal{AP}_i, \neg, \wedge)$. If $M_{ij}, s_{ij} \models \neg r_i \wedge AFr_i$, then $s_{ij}{\uparrow}i$ is sometimes-blocking.*

The converse of this proposition — namely, "if $s_{ij}{\uparrow}i$ is sometimes-blocking, then an eventuality of the form $AFr_i$ is pending in $s_{ij}$" — does not hold in general. For example, in the pair-structure for the mutual exclusion problem (given in Figure 4), $[C_i]$ is a sometimes-blocking $i$-state, but there is no eventuality of the form $AFr_i$ pending in state $[C_i\, T_j]$ of that structure.

6.7.4 *The Progress Lemma.* We can now demonstrate that, under weak blocking fairness as defined above, and given the liveness assumption, every $I$-process with a pending eventuality (clause (3) in the antecedent of Lemma 6.7.4.1) is eventually executed. Very roughly, the proof idea is as follows. The liveness assumption (clause (1)) guarantees that a process with a pending eventuality is not delayed forever by another process via a wait-chain. Given that a process cannot be delayed forever, weak blocking fairness ($\models_\Phi$) is sufficient to guarantee that the process is eventually executed, provided that a deadlock (clause (2)) does not occur.

LEMMA 6.7.4.1 (PROGRESS). *If*

(*1*) *the liveness assumption LV holds,*

(*2*) *for every reachable I-state s, $W_I(s)$ is supercycle-free, and*

(*3*) *v is a reachable I-state of $M_I$ such that $\bigwedge k \in I(\ell) \,.\, (M_{k\ell}, v{\uparrow}k\ell \models \neg r_\ell \wedge AFr_\ell)$ for some $\ell \in dom(I)$ and $r_\ell \in \mathcal{L}(\mathcal{AP}_\ell, \neg, \wedge)$, then*

$$M_I, v \models_\Phi AFex_\ell.$$

### 6.8 The Generalized Large-Model Theorem

We now establish soundness of MP-synthesis for the general case of arbitrary interconnection relations. It turns out that the argument differs from the large-model theorem only for temporal leads-to properties. For this case, our argument is roughly as follows. Assume that some temporal leads-to property is satisfied by the pair-systems. We use the progress lemma above to show that a process with a pending eventuality is always guaranteed to be eventually executed. Thus, until the eventuality is fulfilled, the process will be executed repeatedly. Nonfulfillment of the pending eventuality coupled with infinitely often execution, would, by path projection, imply the same phenomenon in the pair-systems. Hence, the pending eventuality is not fulfilled in the pair-systems, so the pair-systems do not satisfy the temporal leads-to property. But this contradicts our initial assumption.

THEOREM 6.8.1 (GENERALIZED LARGE MODEL). *Let $f_{k\ell}$ be an arbitrary formula of $FLCTL_{k\ell}$ (Definition 6.6.1). Let s be an arbitrary reachable I-state, and, for all i, j such that $i\,I\,j$, let $s_{ij} = s{\uparrow}ij$. If the liveness assumption LV holds, and $W_I(u)$ is supercycle-free for every reachable I-state u, then*

$$\bigwedge(i,j) \in I \,.\, (M_{ij}, s_{ij} \models \textstyle\bigwedge_{k\ell} f_{k\ell}) \text{ implies } M_I, s \models_\Phi \textstyle\bigwedge_{k\ell} f_{k\ell}.$$

As in Section 6.6, we also present the soundness result in terms of correctness properties, i.e., properties that hold in all the initial states of the pair-system or $I$-system.

COROLLARY 6.8.2 (GENERALIZED LARGE MODEL). *Let $f_{k\ell}$ be an arbitrary formula of $FLCTL_{k\ell}$ (Definition 6.6.1). Let $(i,j)$ be an arbitrary pair in I. If the liveness assumption LV holds, and $W_I(u)$ is supercycle-free for every reachable I-state u, then*

$$M_{ij}, S^0_{ij} \models \textstyle\bigwedge_{k\ell} f_{k\ell} \text{ implies } M_I, S^0_I \models_\Phi \textstyle\bigwedge_{k\ell} f_{k\ell}.$$

PROOF. Identical to that of the large-model corollary (6.6.3), except that the generalized large-model theorem (6.8.1) is invoked instead of the large-model theorem (6.6.2). □

We now look at our solution to the generalized dining philosophers problem given in Figure 7. We verify that this solution satisfies the liveness assumption.

Here $M_{jk}$ is the pair-structure for the two-process mutual exclusion problem. We easily check, by inspection of Figure 4, that if $M_{jk}, s_J \uparrow jk \models EGex_k$, then $s_J \uparrow j = [N_j]$. From Figure 5 and the compact MP-synthesis definition (5.4.1), we infer, by straightforward checking, that, for $J = \{\{i,j\},\{j,k\}\}$, $\bigwedge a_i^J . (a_i^J \longrightarrow P_j^J \notin W_J(s_J))$ is always true when $s_J \uparrow j = [N_j]$, i.e., a process in its noncritical state does not block any move of any neighbor. Thus, we conclude that the dining philosophers solution satisfies the liveness assumption.

In Section 6.5, we showed that the solution for the generalized dining philosophers problem given in Figure 7 satisfies the wait-for-graph assumption. Now the wait-for-graph assumption depends only on the pair-system from which the $K$-process system is derived. It is independent of the interconnection relation $I$. Since the dining philosophers solution is derived from the same pair-system as the $K$-process mutual exclusion solution (namely, the two-process mutual exclusion system), the only difference being the particular interconnection relation used, we conclude that the dining philosophers solution satisfies the wait-for-graph assumption. We can likewise verify that $W_I(s_I^0)$ is supercycle-free for every initial state $s_I^0$ of the dining philosophers solution, by an argument analogous to that for the $K$-process mutual exclusion solution given in Section 6.5. Thus, by the supercycle-free wait-for-graph theorem (6.5.4.3), we conclude that $W_I(t)$ is supercycle-free for every reachable state $t$ of the dining philosophers solution.

We have thus shown that the dining philosophers solution meets both assumptions of the generalized large-model corollary (6.8.2), so we conclude that the dining philosophers solution satisfies the MPCTL* specification given in Section 3.4. Since the interconnection relation $I$ can specify any interconnection pattern, our solution solves the *generalized* dining philosophers problem put forth by Chandy and Misra [1984]. In particular, when interpreted for $I = \{\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{5,1\}\}$, our solution solves the original dining philosophers problem put forth by Dijkstra.

## 7. IMPLEMENTATION ON REALISTIC ARCHITECTURES

The programs that we synthesize have a large grain of atomicity in that a process must in one atomic step (1) inspect the local states and (pairwise) shared variables of all of its neighbors and (2) modify all of the shared variables. We now sketch a method of implementing our programs using low atomicity interprocess communication and synchronization primitives that are easily provided by most distributed systems, namely broadcast (as in most local area networks, which are ethernet based), and asynchronous point-to-point FIFO message passing (as in most wide-area networks).

First, we translate a synthesized program into a form which uses the *multiparty interaction* [Back and Kurki-Suonio 1984; 1988; 1989; Charlesworth 1987; Forman 1987; Francez and Forman 1996; Francez et al. 1986; Ramesh and Mehndiratta 1987] as the sole interprocess communication and synchronization primitive. A multiparty interaction is an atomic synchronous event involving a fixed set of processes (the *participants*). The participant processes must all reach a point where executing the interaction is one of the possible continuations. The interaction is then *enabled*. Only enabled interactions can be executed. During an interaction, data can be exchanged between any pair of processes [Forman 1987; Francez and Forman 1996; Ramesh and Mehndiratta 1987]. However, we do not make use of this feature; we use interactions for synchronization only.

The translation of an $I$ system $(S_I^0, P_{i_1}^I \| \dots \| P_{i_K}^I)$ into a multiparty-interaction-based program proceeds as follows:

—For each $P_i^I$, $i \in \{i_1, \dots, i_K\}$, every move is replaced by a single arc labeled with a unique interaction name.

—Every pair-structure $M_{ij}$, $(i\, I\, j)$ is instantiated as a process, which we call a *pair-controller*. Each $P_i^j$-transition in $M_{ij}$ is relabeled with the same interaction name as the corresponding move in $P_i^I$. The variables in $\mathcal{SH}_{ij}$ now become local variables of the process $M_{ij}$.

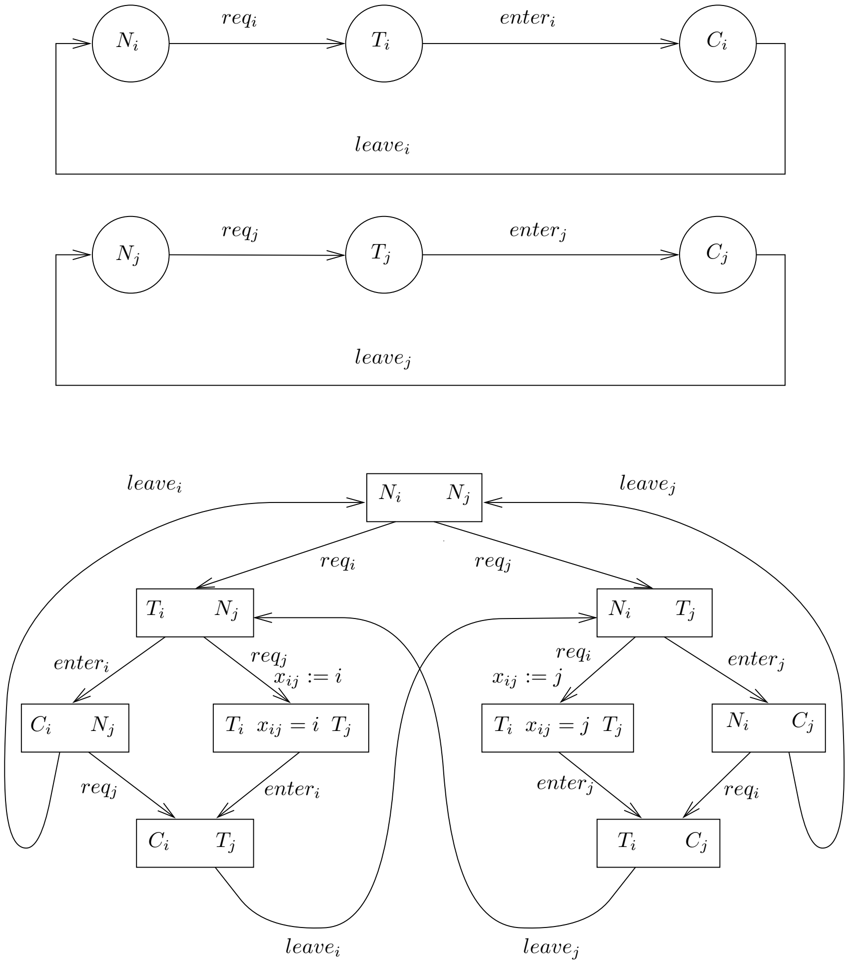
—For every initial state $s_I^0 \in S_I^0$ we have an initial state $t_I^0$ of the translated program. In $t_I^0$, the state of every $P_i^I$, $i \in \{i_1, \dots, i_K\}$, is $s_I^0{\upharpoonright}i$, and the state of every $M_{ij}$, $i\, I\, j$, is $s_I^0{\upharpoonright}ij$.

For example, the generalized dining philosophers solution is given in Figure 15, where $i$ and $j$ are replicated for all $(i, j) \in I$ (it is not necessary to display $P_j^I$ as well as $P_i^I$; this is done to help clarify the example). In $P_i^I$, the move from $T_i$ to $C_i$ has been replaced by a single multiparty interaction, $enter_i$, whose participants are $P_i^I$, and $M_{ij}$ for all $j \in I(i)$. Since every $M_{ij}$ such that $j \in I(i)$ must participate in the execution of $enter_i$, we see that $P_i^I$ cannot enter the critical section without the "permission" of all the pair-controllers just as in the original solution. In fact, it is relatively straightforward to see that the original and translated programs have the same semantics, i.e., they generate isomorphic $I$-structures. The $K$-process mutual exclusion solution is a special case, which contains pair-controllers for every pair $P_i^I, P_j^I$ of processes.

The second step of our implementation method uses one of the many distributed algorithms available [Back and Kurki-Suonio 1988; Back et al. 1985; Bagrodia 1989; Chandy and Misra 1987; 1988] for implementing multiparty interactions in terms of asynchronous message passing or ethernet broadcast. This then gives us our final implementation in terms of widely available, low atomicity primitives.

The number of pair-controllers (and therefore of processes introduced) is equal to the number of pairs in $I$, which, in the worst case, is $O(K^2)$. Thus, this scheme may, in some cases, produce somewhat inefficient programs, with large numbers of processes whose only purpose is to implement the pairwise constraints. This problem may be alleviated by merging some of the pair-controllers. For example, pair-controllers $M_{ij}$ and $M_{jk}$ may be merged into a process $M_{ijk}$ as follows: $M_{ijk}$ stores $M_{ij}$ and $M_{jk}$ as data structures, together with the "current states" of $M_{ij}$ and $M_{jk}$. Using this information, $M_{ijk}$ can determine which interactions it should participate in, and what the next states of $M_{ij}$ and $M_{jk}$ should be after the execution of an interaction. We emphasize that we do not generate $M_{ijk}$ by taking the automata-theoretic product of $M_{ij}$ and $M_{jk}$. Rather, $M_{ijk}$ acts as an "interpreter" that simulates the execution of $M_{ij}$ and $M_{jk}$ (thus, we avoid generating in the worst case an exponentially large structure). Note that pair-controllers which are merged do not have to have common process indices, but the most efficient programs are generated when pair-controllers with common indices are merged.

Since we can merge any number of pair-controllers in this way, we can generate multiparty-interaction-based programs that range across the distributed/centralized




The initial state set is $\{ \; [N_i \, N_j \, [N_i \, N_j]] \; \}$

Fig. 15. Multiparty-interaction solution to the generalized dining philosophers problem.

spectrum. The more centralized programs will use fewer messages overall, but will have worse bottleneck performance, since a process that represents a large number of pair-controllers will have to handle a large number of messages, as it will interact with all of the $P_i^I$ for every $i$ that is an index of one of the pair-controllers represented. Likewise, more distributed programs will have better bottleneck performance, but will generate more messages overall. Thus, we have the usual tradeoffs and can make the appropriate choice based on the desired performance characteristics and on the target architecture. A detailed study of these issues is beyond the scope of this article.

In certain special cases, we can reduce the atomicity of the operations in our synthesized programs without appealing to a multiparty-interaction scheduling algorithm. We outline this in the following two sections.

### 7.1 Reducing Read Atomicity: Guard Stability

The solutions produced by MP-synthesis contain read and write operations on vectors of shared variables. There are certain conditions under which these vector operations can be broken down into a sequence of simple operations. For example, in the MP-synthesis solution to the mutual exclusion problem, the guard on the $T_i$ to $C_i$ arc is $\bigwedge_{j \in I(i)}(N_j \vee (T_j \wedge x_{ij} = i))$. Each individual conjunct of this guard, i.e., $(N_j \ \vee \ (T_j \wedge x_{ij} = i))$ for some particular value of $j$, is *temporarily stable* [Katz 1986] in the sense that if $P_i^I$ is "at" $T_i$, then once $(N_j \ \vee \ (T_j \wedge x_{ij} = i))$ holds, it will continue to hold until $P_i^I$ enters the critical section $C_i$. Hence the conjuncts of the guard do not need to be checked simultaneously in one "large" vector operation. Any sequential order of checking the guard conjuncts will do.

We generalize this discussion as follows. Let $(s_i, \bigwedge_{j \in I(i)} B_i^j \rightarrow //_{j \in I(i)} A_i^j, t_i)$ be an arc of $P_i^I$. If $M_{ij}, S_{ij}^0 \models AG((\{\!\!\{s_i\}\!\!\} \wedge B_i^j) \Rightarrow A[(\{\!\!\{s_i\}\!\!\} \wedge B_i^j) \ U \ t_i])$, then the conjuncts $B_i^j$ in $\bigwedge_{j \in I(i)} B_i^j$ can be checked sequentially instead of simultaneously, since once true, each $B_i^j$ will remain true until the arc is executed. The condition $M_{ij}, S_{ij}^0 \models AG((\{\!\!\{s_i\}\!\!\} \wedge B_i^j) \Rightarrow A[(\{\!\!\{s_i\}\!\!\} \wedge B_i^j) \ U \ t_i])$ can be checked in polynomial time by the model-checking algorithm of Clarke et al. [1986].

### 7.2 Reducing the Atomicity of Assignment Statements

In the previous section we gave a way of reducing read atomicity. We now show how to reduce write atomicity in some cases. The solutions produced by MP-synthesis contain write operations on vectors of shared variables, e.g.,, $//_{j \in I(i)} A_i^j$ in $P_i^I$, where the $A_i^j$ have to be $x_{ij} := i$ or $x_{ij} := j$. We now assume that $P_i^j$ contains only assignments of the form $x_{ij} := j$; $P_j^i$ contains only assignments of the form $x_{ij} := i$; and every move assigns to at most one shared variable. We now implement these assignments in terms of the solution to the generalized dining philosophers problem given in Chandy and Misra [1984] and in Chandy and Misra [1988], which uses asynchronous FIFO message passing. Let each $P_i^I$ be a philosopher, and let $P_i^I$ and $P_j^I$ share a fork if and only if $i \, I \, j$. For each shared variable $x_{ij}$ in the pair-system $(S_{ij}^0, P_i^j \| P_j^i)$, we introduce an instance of the dining philosophers algorithm. By our assumption, the only assignments to $x_{ij}$ in $P_i^I$ are of the form $//_{j \in I(i)} \, x_{ij} := j$. We implement $x_{ij}$ in terms of the *priority relation* of the dining philosophers algorithm. This relation determines which philosopher has priority in obtaining the shared fork. We encode the value of $x_{ij}$ as follows:

$$
\begin{array}{c}
x_{ij} = i \text{ iff } P_i^I \text{ has priority} \\
x_{ij} = j \text{ iff } P_j^I \text{ has priority}
\end{array}
$$

Now when $P_i^I$ needs to execute $//_{j \in I(i)} \, x_{ij} := j$, it transitions to "hungry" and waits to "eat." After $P_i^I$ has eaten, all forks will be at $P_i^I$ and will be dirty, i.e., $P_j^I$ (for all $j \in I(i)$) will have priority, as required. Thus the large atomicity multiple assignment $//_{j \in I(i)} \, x_{ij} := j$ has been implemented in terms of the small atomicity operations of the dining philosophers algorithm.

## 8. DISCUSSION

### 8.1 Scope of the Synthesis Method

Our synthesis method requires the pair-systems to satisfy two technical assumptions: the wait-for-graph assumption and the liveness-assumption. A consequence of this is that the method is not *complete*: it is possible that a given pair-system may not satisfy these assumptions, and yet the $I$-system synthesized from it may still satisfy the MPCTL$^*$ specification from which the pair-system was produced. It is natural to ask to what extent the range of applicability of our method is limited by imposing these assumptions.

Roughly, the liveness assumption prevents a process from executing continuously while delaying another process that has a pending eventuality. This behavior should not occur in a correct program anyway, so the liveness assumption is not restrictive at all; it can be thought of as a *design rule*: any system permitting the behavior prevented by the liveness assumption should be considered to be badly designed, since it allows individual starvation. The wait-for-graph assumption is somewhat restrictive. Roughly, it requires that a process join a wait-chain either at its beginning or at its end. However, there are situations where processes enter wait-chains "in the middle," e.g. priority queues, and where the $I$-system would still be deadlock-free. Essentially, the wait-for-graph assumption represents a particular strategy for *deadlock prevention*. We expect that a more general deadlock prevention strategy (e.g., acquire resources in a fixed linear order) should also be applicable to our synthesis method. Investigating this and generalizing the wait-for-graph assumption are two of our research goals.

Even with our current wait-for-graph assumption, we point out that many systems can be synthesized. We have successfully applied the method to the well-known resource contention problems from the distributed computing literature: mutual exclusion, readers-writers, dining philosophers, drinking philosophers, and the $m$-out-of-$n$ mutual exclusion problem (in which $n$ processes contend for $m$ identical and unsharable resources). Thus, we see that our method already deals with a large and important class of concurrent and distributed systems, namely resource contention. Generalizing the wait-for-graph assumption will enlarge its scope even further.

### 8.2 Complexity of the Synthesis Method

There are three algorithmic steps in applying the method: (1) applying Definition 5.1.1 to synthesize the $I$-system from the given pair-system, (2) checking the wait-for-graph assumption, and (3) checking the liveness-assumption. These have worst-case time complexities of $O(|I| \cdot |P_k^\ell|)$, $O(b \cdot |P_k^\ell|^{b+4} \cdot 2^{(b+1)|\mathcal{SH}_{k\ell}|})$, and $O(|P_k^\ell|^3 \cdot 2^{2|\mathcal{SH}_{k\ell}|})$, respectively (see Appendix E), where $b$ is the maximum branching factor (i.e., maximum number of outgoing moves from a single node) of $P_k^\ell$. Since $b$ is at least 1, the second term dominates the third, and hence the overall complexity of the synthesis method is the sum of the first two terms, i.e., $O(|I| \cdot |P_k^\ell| + b \cdot |P_k^\ell|^{b+4} \cdot 2^{(b+1)|\mathcal{SH}_{k\ell}|})$.

The first term clearly cannot be improved, since the given pair-system and interconnection relation that are the inputs to the method have to be examined at least once. The second term is due to the check of the wait-for-graph assumption. The check given in this article is basically a brute-force approach: the structure $M_J$ for the relevant $J$-system(s) is first constructed, and then essentially model checked against the wait-for-graph assumption. It may be possible to perform this check without exhaustively generating $M_J$, e.g., by using the BDD-based model-checking techniques of Browne et al. [1986; 1990] and Clarke et al. [1992]. In particular, Browne et al. [1990] state that symmetric functions have compact BDD representations. Since the wait-for-graph assumption is symmetric in most of its arguments, namely the processes $\ell_1, \ldots, \ell_n$, this approach appears promising.

Devising a more efficient check may allow us to eliminate the exponential factor in $b$. We note that the exponential factor in $|\mathcal{SH}_{k\ell}|$ cannot be eliminated in general, since the number of reachable states of $M_{k\ell}$ is exponential in $|\mathcal{SH}_{k\ell}|$ in the worst case.

### 8.3 Efficiency of the Synthesized System

The systems currently synthesized are not very efficient, since they have poor *locality*: a process can wait for another process that is quite distant from it (a process that communicates with it indirectly via a long chain of other processes). Furthermore, a process can wait for another process repeatedly, i.e., the second process can acquire and release resources many times before the first process progresses. By regarding the pair-controllers as resources (they can be regarded as such, since their permission is required in order to make a transition), we can apply efficient distributed resource allocation strategies to enhance the efficiency of the distributed low-atomicity systems produced by the implementation technique outlined above. For example, in Lynch [1981] and Styer and Peterson [1988], distributed resource allocation algorithms are presented where a process' waiting time is a "small function of the number and speed of nearby processes" and where a process cannot repeatedly acquire and release resources ahead of its neighbors ($I$-neighbors in our context).

## 9. RELATED WORK

The only other temporal logic synthesis method which exploits similarity of processes that we are aware of is that of Emerson and Srinivasan [1990] and Srinivasan [1991]. Specifications in this method are expressed in a logic similar to MPCTL. The method constructs a model of a given (satisfiable) specification that can be regarded as a global state transition diagram in which the (global) states record the number of processes in a particular local state. From this model, a concurrent program with an arbitrary number of processes is extracted. By recording only the number of processes in each local state, instead of the explicit local states, the size of the model is kept independent of the number of processes, and so state explosion is avoided.

One potential disadvantage of this method is that all the processes share a single variable *dist*, which holds the identity of a current "distinguished" process. In our framework, this means that the interconnection relation $I$ is always the complete graph. Thus, communication between certain pairs of processes (e.g., nonneighboring philosophers) will be required even when it is not really necessary for a solution. Variable *dist* is used to schedule in a round-robin fashion all processes that have pending eventualities. Since this may include pairs of processes that do not "conflict" with each other (again the example of non-neighboring philosophers is illustrative) the resulting ordering would, in many cases, be overly restrictive and would result in poor performance. Additionally, *dist* constitutes a bottleneck, with all of the well-known performance and reliability problems that result.

Other temporal logic synthesis methods to date [Anuchitanukul and Manna 1994; Emerson and Clarke 1982; Manna and Wolper 1984; Pnueli and Rosner 1989a; 1989b] all rely on some form of explicit global state enumeration (either generating an explicit global state transition diagram or a tree automaton) and so have at least exponential time complexity.

Related to synthesis is the technique of *model checking*: given a program and a specification, determine mechanically whether the program satisfies the specification. In Clarke et al. [1986] and Sistla and German [1992], methods of model checking finite-state concurrent programs (with respect to a specification expressed in some temporal logic) that exploit process similarity are presented. Clarke et al. [1986] require the manual construction of a bisimulation between the two-process and $K$-process programs, and their method is therefore not completely mechanical. Clarke and Grumberg [1987] attempt to automate the method of Clarke et al. [1986], but require an intermediate construction that is exponentially large in the size of a single process. The method is also incomplete in the same sense that ours is: there is no guarantee that the required technical assumptions will be satisfied in every case. Sistla and German [1992] present a model-checking method for programs consisting of strictly identical processes (no differentiating process indices) that communicate via CCS-style [Milner 1989] synchronous actions. Their model-checking algorithm runs in time polynomial in the size of a single process and exponential in the size of the specification. Unfortunately, the program notation is so weak that a program satisfying mutual exclusion cannot be written. To accommodate mutual exclusion, a distinguished controller process must be introduced. However, the running time now becomes doubly exponential in both process and specification size.

Finally, we mention Clarke et al. [1993] and Emerson and Sistla [1993], where the symmetry of the global state transition diagram that results from process similarity is exploited. The basic idea is to model check a reduced form of the global state transition diagram that results by taking the quotient with respect to a symmetry group. Computing the symmetry group is expensive however, so either manual techniques [Emerson and Sistla 1993] or approximation methods [Clarke et al. 1993] must be used.

## 10. FURTHER WORK

We believe our work is a significant step in the direction of producing a practical temporal logic synthesis technique although there are still many open problems for future research. Some of these are as follows:

(1) The accommodation of multiple classes of processes, with each class containing an arbitrarily large number of similar processes. Let the classes be $\mathcal{C}_1, \ldots, \mathcal{C}_N$. Now we require the following pair-systems as input:
   —One pair-system per class $\mathcal{C}_n$, which is a model for a pair of generic processes $P_{n,i}, P_{n,j}$ which are representative of that class. This model captures the interactions of the processes within $\mathcal{C}_n$.

—One pair-system for every pair of classes $\mathcal{C}_n, \mathcal{C}_m$, which is a model for a pair of generic representative processes $P_{n,i}, P_{m,j}$, where $P_{n,i}$ represents $\mathcal{C}_n$ and $P_{m,j}$ represents $\mathcal{C}_m$. This model captures the interactions of processes in $\mathcal{C}_n$ with processes in $\mathcal{C}_m$.

The interconnection relation would now relate processes in different classes as well as processes in the same class. The method of computing $I$-processes from pair-processes by Definition 5.1.1 remains as before. However, the spatial modalities of MPCTL are modified slightly so that the indices that they quantify over range over a particular class. So, for example, we would have $\bigwedge_{j \in \mathcal{C}_n}$ quantify over the set of processes in class $\mathcal{C}_n$, and we would have $\bigwedge_{i \in \mathcal{C}_m, j \in \mathcal{C}_m}$ to quantify over the pairs of processes where one element of the pair is in class $\mathcal{C}_n$ and the other process is in class $\mathcal{C}_m$. The generalized large-model theorem should then allow us to conclude that

$$M_{ij}, S^0_{ij} \models \bigwedge_{k \in \mathcal{C}_n, \ell \in \mathcal{C}_m} f_{k\ell} \textit{ implies } M_I, S^0_I \models \bigwedge_{k \in \mathcal{C}_n, \ell \in \mathcal{C}_m} f_{k\ell},$$

given that some antecedents (the analogue of the wait-for-graph assumption, for example) hold. Here $i$ and $j$ are arbitrary elements of $\mathcal{C}_n$ and $\mathcal{C}_m$, respectively.

(2) We intend to formalize the implementation method outlined in Section 7 and prove its correctness. We will also improve the efficiency of the final implemented system using the resource allocation algorithms of Lynch [1981] and Styer and Peterson [1988] (see Section 8.3). Further improvements may also be obtained by devising special-purpose multiparty-interaction implementation algorithms that exploit the symmetric structure of the programs produced by MP-synthesis and/or the specific program semantics. For example, in some cases, the tight synchronization of multiparty interactions may not be needed, and it may be possible to directly replace a multiparty interaction by a much more efficient asynchronous broadcast [Birman and Clark 1994].

(3) Another issue that bears investigation is the use of a "disjunctive" composition technique to synthesize many-process systems. The technique we presented is "conjunctive" and is suited for problems such as the mutual exclusion problem, where access to the critical section requires the "permission" of all the processes in the system. Some problems have a "disjunctive" nature, for example the diffusing computation problem [Dijkstra et al. 1983], and it would be worthwhile to attempt to produce a synthesis technique which addresses such problems.

(4) There has been considerable success in the application of model checking to the verification of VLSI circuits [Browne et al. 1986; Clarke et al. 1989; 1992; Mishra and Clarke 1985]. It would be worthwhile to investigate the problem of synthesizing circuits from temporal logic specifications (this would require the extension to multiple-process classes as a prerequisite).

(5) We intend to investigate the synthesis of real-time concurrent systems. Our approach is to use a real-time variant of CTL, such as RTCTL [Emerson et al. 1993], to express real-time specifications. Already, we have established a real-time version of the large-model theorem [Attie 1995]. It remains to do the same for the generalized large-model theorem.

(6) Finally we shall investigate the synthesis of fault-tolerant concurrent systems.

## 11. CONCLUSIONS

In this article, we have shown how to extend the synthesis methods of Emerson and Clarke [1982] and Manna and Wolper [1984] to deal with an arbitrary number of similar processes without incurring the exponential overhead due to the state explosion problem. Since we have reduced the problem of reasoning about $K$ processes to the problem of reasoning about a small number of processes, the results of Emerson et al. [1992] imply that the synthesis problem can be solved in polynomial time. Our method works for arbitrary process interconnection schemes. We also outlined a way of implementing the synthesized programs in terms of interprocess communication and synchronization primitives that are available in practice. Thus, we see this contribution as a significant step toward producing a mechanical synthesis technique capable of generating solutions for realistic problems.

## APPENDIX

## A. GLOSSARY OF SYMBOLS

| | |
|---|---|
| $\models$ | Satisfies relation of CTL, CTL* |
| $\models_\Phi$ | Satisfies relation of CTL, CTL* relativized to fairness notion $\Phi$ |
| $\Phi$ | CTL* path formula that specifies weak blocking fairness |
| $\otimes$ | "Conjunctive" guarded-command composition operator |
| $\oplus$ | "Disjunctive" guarded-command composition operator |
| $\{\!\mid\!\}$ | State to formula operator |
| $\bigwedge_i$ | Spatial modality (quantifies over individual processes) |
| $\bigwedge_{ij}$ | Spatial modality (quantifies over pairs of processes related by $I$) |
| $\uparrow i$ | State projection onto a single process $i$ |
| $\uparrow ij$ | State or path projection onto an $ij$-system |
| $\uparrow J$ | State or path projection onto a $J$-system |
| $j_1/i_1, ..., j_m/i_m$ | Process index substitution operator |
| $\theta$ | Synonym for $j_1/i_1, ..., j_m/i_m$ |
| $a_i^j$ | Pair-move (move in a compact pair-process) |
| $a_i^I$ | $I$-move (move in a compact $I$-process) |
| $a_i^I.start$ | Start state of the $I$-move $a_i^I$ |
| $a_i^I.guard$ | Guard of the $I$-move $a_i^I$ |
| $a_i^I.guard_j$ | Conjunct of $a_i^I.guard$ that tests shared state with process $j$ |
| $blk_i$ | Propositional formula true in exactly the sometimes-blocking local states |
| $dom(I)$ | Synonym for $i_1, ..., i_K$ |
| $i_1, ..., i_K$ | Domain of $I$ (i.e., the set of process indices in the synthesized system) |
| $t_k.moves$ | The set of moves with start state $t_k$ |
| $FLCTL_{kl}$ | The sublogic Flat CTL |
| $I$ | The interconnection relation |
| $M_{ij}$ | The $ij$-structure |
| $M_I$ | The $I$-structure |
| $M_I^r$ | The reachable portion of $M_I$ |
| $P_i^j$ | Process $i$ in the pair-system consisting of processes $i$ and $j$ |
| $P_i^I$ | Process $i$ in the system synthesized using interconnection |

| | |
|---|---|
| | relation $I$ |
| $R_{ij}$ | The transition relation of $M_{ij}$ |
| $R_I$ | The transition relation of $M_I$ |
| $R_I^r$ | The reachable portion of $R_I$ |
| $S_{ij}$ | The set of states of $M_{ij}$ |
| $S_{ij}^0$ | The set of initial states of $M_{ij}$ |
| $S_I$ | The set of states of $M_I$ |
| $S_I^0$ | The set of initial states of $M_I$ |
| $S_I^r$ | The set of reachable states of $M_I$ |
| $W_I(s)$ | The wait-for-graph for the $I$-system in global state $s$ |
| $(S_{ij}^0, P_i^j \\| P_j^i)$ | The pair-system consisting of processes $i$ and $j$ |
| $(S_I^0, P_{i_1}^I \\| \ldots \\| P_{i_K}^I)$ | The $I$-system |
| $\mathcal{AP}$ | The family of sets of atomic propositions of the synthesized system |
| $\mathcal{AP}_i$ | The set of atomic propositions of process $i$ |
| $\mathcal{L}$ | The sublogics constructor |
| $\mathcal{SH}_{ij}$ | The set of variables shared by processes $i$ and $j$ |

## B. PROOF OF THE LARGE-MODEL THEOREM

THEOREM 6.6.2 (LARGE MODEL). *Let $f_{k\ell}$ be an arbitrary formula of $FLCTL_{k\ell}$. Let $I = \{i_1, \ldots, i_K\} \times \{i_1, \ldots, i_K\} - \{(i,i) \mid i \in \{i_1, \ldots, i_K\}\}$. Furthermore, let $s$ be an arbitrary reachable $I$-state, and, for all $i, j$ such that $i\,I\,j$, let $s_{ij} = s{\upharpoonright}ij$. If $M_I, S_I^0 \models AGEXtrue$, then*

$$\bigwedge(i,j) \in I\,.\,(M_{ij}, s_{ij} \models \textstyle\bigwedge_{k\ell} f_{k\ell}) \textit{ implies } M_I, s \models \textstyle\bigwedge_{k\ell} f_{k\ell}.$$

PROOF. By MPCTL* semantics (see Section 3), $M_{ij}, s_{ij} \models \bigwedge_{k\ell} f_{k\ell}$ is equivalent to $M_{ij}, s_{ij} \models f_{ij}$. Also by MPCTL* semantics, $M_I, s \models \bigwedge_{k\ell} f_{k\ell}$ is equivalent to $M_I, s \models \bigwedge(i,j) \in I\,.\,(f_{ij})$. This is equivalent, by CTL* semantics, to $\bigwedge(i,j) \in I\,.\,(M_I, s \models f_{ij})$. Hence the large-model theorem is established if we can prove that

$$\bigwedge(i,j) \in I\,.\,(M_{ij}, s_{ij} \models f_{ij}) \text{ implies } \bigwedge(i,j) \in I\,.\,(M_I, s \models f_{ij}) \qquad \text{(P)}$$

given the assumptions of the large-model theorem. The proof is by induction on the structure of $f_{ij}$. In each case we assume the antecedent $\bigwedge(i,j) \in I\,.\,(M_{ij}, s_{ij} \models f_{ij})$, and we establish the consequent $\bigwedge(i,j) \in I\,.\,(M_I, s \models f_{ij})$. In the remainder of the proof, we make the convention that all free occurrences of $i, j$ represent arbitrary elements of $dom(I)$ such that $i\ I\ j$. Thus it suffices to establish $M_I, s \models f_{ij}$. Note that in some cases (which do not invoke the inductive hypothesis), we will, for convenience, use $M_{ij}, s_{ij} \models f_{ij}$ as the antecedent, with $i, j$ free. Hence we will, for these cases, actually be establishing that $\bigwedge(i,j) \in I\,.\,(M_{ij}, s_{ij} \models f_{ij}$ implies $M_I, s \models f_{ij})$, which implies (P). This is permissible, since we only do this for those cases which do not invoke the inductive hypothesis.

$f_{ij} = h_{ij}$. The antecedent is $M_{ij}, s_{ij} \models h_{ij}$. Now $s_{ij} = s{\upharpoonright}ij$ by assumption. Thus by the $I$-state projection proposition (6.3.1) with $J = \{\{i,j\}\}$, we have $M_I, s \models h_{ij}$.

$f_{ij} = f'_{ij} \wedge f''_{ij}$. The antecedent is $\bigwedge(i,j) \in I \,.\, (M_{ij}, s_{ij} \models (f'_{ij} \wedge f''_{ij}))$. So, by CTL* semantics, we get $\bigwedge(i,j) \in I \,.\, (M_{ij}, s_{ij} \models f'_{ij})$ and $\bigwedge(i,j) \in I \,.\, (M_{ij}, s_{ij} \models f''_{ij})$. From these and the induction hypothesis, we get $\bigwedge(i,j) \in I \,.\, (M_I, s \models f'_{ij})$ and $\bigwedge(i,j) \in I \,.\, (M_I, s \models f''_{ij})$. So by CTL* semantics we get $\bigwedge(i,j) \in I \,.\, (M_I, s \models (f'_{ij} \wedge f''_{ij}))$.

$f_{ij} = AGh_{ij}$. Let $t$ be an arbitrary $s$-reachable state in $M_I$. By assumption $s{\uparrow}ij = s_{ij}$. Thus, by the relativized state-mapping corollary (6.4.6) with $J = \{\{i,j\}\}$, $t{\uparrow}ij$ is a $s_{ij}$-reachable state in $M_{ij}$. The antecedent is $M_{ij}, s_{ij} \models AGh_{ij}$. Hence $M_{ij}, t{\uparrow}ij \models h_{ij}$ by CTL* semantics. From this and the $I$-state projection proposition (6.3.1) with $J = \{\{i,j\}\}$, we get $M_I, t \models h_{ij}$. Since $t$ is an arbitrary $s$-reachable state in $M_I$, we conclude $M_I, s \models AGh_{ij}$ by CTL* semantics.

$f_{ij} = AG(p_i \Rightarrow AY_i q_i)$. We will establish that

$$M_I, t \models (p_i \Rightarrow AY_i q_i) \tag{*}$$

where $t$ is an arbitrary $s$-reachable state in $M_I$ (remember that $t$ is $s$-reachable if and only if there is a path from $s$ to $t$). If $M_I, t \models \neg p_i$ then (*) holds immediately, and we are done. Otherwise $M_I, t \models p_i$, and we proceed as follows. The antecedent is

$$M_{ij}, s_{ij} \models AG(p_i \Rightarrow AY_i q_i). \tag{a}$$

By assumption, $s{\uparrow}ij = s_{ij}$. Thus by the relativized state-mapping corollary (6.4.6) with $J = \{\{i,j\}\}$, $t{\uparrow}ij$ is a $s_{ij}$-reachable state in $M_{ij}$. So, by (a) we have

$$M_{ij}, t{\uparrow}ij \models (p_i \Rightarrow AY_i q_i). \tag{b}$$

Since $M_I, t \models p_i$, we have $M_{ij}, t{\uparrow}ij \models p_i$ by the $I$-state projection proposition (6.3.1) with $J = \{\{i,j\}\}$. Together with (b), this yields

$$M_{ij}, t{\uparrow}ij \models AY_i q_i. \tag{c}$$

Now let $t \xrightarrow{i} u$ be an arbitrary $P_i^I$-transition (in $M_I$) out of state $t$ (if no such transition exists, then (*) holds by CTL* semantics, and we are done). By the transition-mapping lemma (6.4.1), we have

$$t{\uparrow}ij \xrightarrow{i} u{\uparrow}ij \in R_{ij}. \tag{d}$$

By (c), (d), and CTL* semantics we have

$$u{\uparrow}ij \models q_i. \tag{e}$$

By (e) and the $I$-state projection proposition (6.3.1) with $J = \{\{i,j\}\}$ we have

$$u \models q_i.$$

Since $t \xrightarrow{i} u$ is an arbitrary $P_i^I$-transition out of state $t$, we have $M_I, t \models AY_i q_i$, and therefore $M_I, t \models (p_i \Rightarrow AY_i q_i)$.

We have therefore established (*). Since $t$ is an arbitrary $s$-reachable state in $M_I$, we conclude that $M_I, s \models AG(p_i \Rightarrow AY_i q_i)$.

$f_{ij} = AG(p_i \Rightarrow A[q_i U r_i])$. We will establish $M_I, t \models (p_i \Rightarrow A[q_i U r_i])$ where $t$ is an arbitrary $s$-reachable state in $M_I$. If $M_I, t \models \neg p_i$ then $M_I, t \models (p_i \Rightarrow A[q_i U r_i])$, and we are done. Otherwise we have $M_I, t \models p_i$, and we proceed as follows.

Let $\pi$ be an arbitrary fullpath starting in $t$. By assumption, $M_I, S_I^0 \models AGEXtrue$, and so $\pi$ is infinite. Since the number $K$ $(= |dom(I)|)$ of $I$-processes is finite, $\pi$ contains an infinite number of transitions of at least one $I$-process. Call this $I$-process $P_k^I$. We now select an $I$-process $P_\ell^I$ as follows. If $k = i$, then $\ell$ is an arbitrary member of $I(i)$. If $k \neq i$, then $\ell$ is $k$ (note that $k, \ell$ depend on the choice of $\pi$, in general). Hence, we have

$\pi$ contains an infinite number of $P_i^I$ transitions or
an infinite number of $P_\ell^I$ transitions. (**)

The antecedent is

$$\bigwedge(i,j) \in I \,.\, (M_{ij}, s_{ij} \models AG(p_i \Rightarrow A[q_i U r_i])).$$

By assumption, $I = \{i_1, \ldots, i_K\} \times \{i_1, \ldots, i_K\} - \{(i,i) \mid i \in \{i_1, \ldots, i_K\}\}$, so we have $(i, \ell) \in I$. Thus

$$M_{i\ell}, s_{i\ell} \models AG(p_i \Rightarrow A[q_i U r_i]). \tag{a}$$

By assumption, $s{\uparrow}i\ell = s_{i\ell}$. Thus by the relativized state-mapping corollary (6.4.6) with $J = \{\{i, \ell\}\}$, $t{\uparrow}i\ell$ is a $s_{i\ell}$-reachable state in $M_{i\ell}$. So by (a) we have

$$M_{i\ell}, t{\uparrow}i\ell \models (p_i \Rightarrow A[q_i U r_i]). \tag{b}$$

Since $M_I, t \models p_i$, we have $M_{i\ell}, t{\uparrow}i\ell \models p_i$ by the $I$-state projection proposition (6.3.1) with $J = \{\{i, \ell\}\}$. Together with (b), this yields

$$M_{i\ell}, t{\uparrow}i\ell \models A[q_i U r_i]. \tag{c}$$

By the path-mapping lemma (6.4.3), $\pi{\uparrow}i\ell$ is a path in $M_{i\ell}$. By (**) and the path projection definition (5.1.2.1), $\pi{\uparrow}i\ell$ contains an infinite number of $P_\ell^i$-transitions or an infinite number of $P_i^\ell$-transitions, and so $\pi{\uparrow}i\ell$ is a fullpath. Also, by the path projection definition (5.1.2.1), $\pi{\uparrow}i\ell$ starts in $t{\uparrow}i\ell$, and therefore, by (c), we have $M_{i\ell}, \pi{\uparrow}i\ell \models [q_i U r_i]$. By the path projection lemma (6.3.3) with $J = \{\{i, \ell\}\}$, we have $M_I, \pi \models [q_i U r_i]$.

Since $\pi$ is an arbitrary fullpath starting in $t$, we conclude, by CTL* semantics, that $M_I, t \models A[q_i U r_i]$, and hence $M_I, t \models (p_i \Rightarrow A[q_i U r_i])$. Since $t$ is an arbitrary $s$-reachable state in $M_I$, we conclude that $M_I, s \models AG(p_i \Rightarrow A[q_i U r_i])$.

$f_{ij} = AG(a_i \Rightarrow EX_i b_i)$. We will establish $M_I, t \models (a_i \Rightarrow EX_i b_i)$ where $t$ is an arbitrary $s$-reachable state in $M_I$. If $M_I, t \models \neg a_i$ then $M_I, t \models (a_i \Rightarrow EX_i b_i)$, and we are done. Otherwise $M_I, t \models a_i$, and we proceed as follows.

Since $a_i, b_i \in \{\{\!|s_i|\!\} \mid s_i \in S_i\}$, let $a_i = \{\!|s_i^m|\!\}, b_i = \{\!|s_i^n|\!\}$, respectively, for some $s_i^m, s_i^n$ in $S_i$. By the antecedent, we have

$$\bigwedge(i,j) \in I \,.\, (M_{ij}, s_{ij} \models AG(a_i \Rightarrow EX_i b_i)).$$

Hence, letting $i$ be free (and remembering our convention for $i, j$) we have

$$\bigwedge j \in I(i) \,.\, (M_{ij}, s_{ij} \models AG(a_i \Rightarrow EX_i b_i)).$$

By assumption, $s{\uparrow}ij = s_{ij}$. Thus, by the relativized state-mapping corollary (6.4.6) with $J = \{\{i,j\}\}$, $t{\uparrow}ij$ is a $s_{ij}$-reachable state in $M_{ij}$ for all $j$ in $I(i)$. So,

$$\bigwedge j \in I(i) \,.\, (M_{ij}, t{\uparrow}ij \models (a_i \Rightarrow EX_i b_i)).$$

Since $M_I, t \models a_i$, we conclude $M_{ij}, t{\uparrow}ij \models a_i$ by the $I$-state projection proposition (6.3.1). Together with the above this yields

$$\bigwedge j \in I(i) \,.\, (M_{ij}, t{\uparrow}ij \models EX_i b_i). \tag{a}$$

From this and CTL* semantics, we get

$$\bigwedge j \in I(i) \,.\, (t{\uparrow}ij \xrightarrow{i} u_{ij} \in R_{ij} \text{ for some } ij\text{-state } u_{ij} \text{ such that } u_{ij} \models b_i). \tag{b}$$

From $b_i = \{\!\!\{s_i^n\}\!\!\}$ and $u_{ij} \models b_i$, we have

$$\bigwedge j \in I(i) \,.\, (u_{ij} \models \{\!\!\{s_i^n\}\!\!\}). \tag{d}$$

By (d) and the local state projection proposition (6.3.2) we have

$$\bigwedge j \in I(i) \,.\, (u_{ij}{\uparrow}i \models \{\!\!\{s_i^n\}\!\!\}).$$

By this and the definition of $\{\!\!\{s_i^n\}\!\!\}$ (Section 3) we get

$$\bigwedge j \in I(i) \,.\, (u_{ij}{\uparrow}i = s_i^n). \tag{e}$$

Now let $u$ be a global state such that

$$\begin{aligned}
&u{\uparrow}i = s_i^n \\
&\bigwedge j \in I(i) \,.\, (u{\uparrow}j = u_{ij}{\uparrow}j \text{ and } u{\uparrow}\mathcal{SH}_{ij} = u_{ij}{\uparrow}\mathcal{SH}_{ij}) \\
&\bigwedge j \in dom(I) - \hat{I}(i) \,.\, (u{\uparrow}j = t{\uparrow}j) \\
&\bigwedge j,k \in dom(I) - \{i\}, j\, I\, k \,.\, (u{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}).
\end{aligned} \tag{f}$$

By (e) and (f), we have

$$\bigwedge j \in I(i) \,.\, (u{\uparrow}i = u_{ij}{\uparrow}i \text{ and } u{\uparrow}j = u_{ij}{\uparrow}j \text{ and } u{\uparrow}\mathcal{SH}_{ij} = u_{ij}{\uparrow}\mathcal{SH}_{ij}).$$

From this we have

$$\bigwedge j \in I(i) \,.\, ((u{\uparrow}i \cup u{\uparrow}j \cup u{\uparrow}\mathcal{SH}_{ij}) = (u_{ij}{\uparrow}i \cup u_{ij}{\uparrow}j \cup u_{ij}{\uparrow}\mathcal{SH}_{ij})).$$

Since $(u{\uparrow}i \cup u{\uparrow}j \cup u{\uparrow}\mathcal{SH}_{ij}) = u{\uparrow}ij$, and $(u_{ij}{\uparrow}i \cup u_{ij}{\uparrow}j \cup u_{ij}{\uparrow}\mathcal{SH}_{ij}) = u_{ij}$, we have

$$\bigwedge j \in I(i) \,.\, (u{\uparrow}ij = u_{ij}). \tag{g}$$

By (b) and (g) we have

$$\bigwedge j \in I(i) \,.\, (t{\uparrow}ij \xrightarrow{i} u{\uparrow}ij \in R_{ij}). \tag{h}$$

By (h) and (f) we have

$$\begin{aligned}
&\bigwedge j \in I(i) \,.\, (t{\uparrow}ij \xrightarrow{i} u{\uparrow}ij \in R_{ij}) \\
&\bigwedge j \in dom(I) - \hat{I}(i) \,.\, (u{\uparrow}j = t{\uparrow}j) \\
&\bigwedge j,k \in dom(I) - \{i\}, j\, I\, k \,.\, (u{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}).
\end{aligned} \tag{i}$$

By (i) and the transition-mapping lemma (6.4.1), we have

$$t \xrightarrow{i} u \in R_I.$$

Now $u{\uparrow}ij = u_{ij}$, and $u_{ij} \models b_i$ (by (b)). So by the $I$-state projection proposition (6.3.1), $u \models b_i$. Thus by CTL* semantics we get

$$M_I, t \models EX_i b_i,$$

so $M_I, t \models (a_i \Rightarrow EX_i b_i)$. Since $t$ is an arbitrary $s$-reachable state in $M_I$, we conclude $M_I, s \models AG(a_i \Rightarrow EX_i b_i)$.

This last case concludes the proof of (P), which in turn completes the proof of the large-model theorem. □

COROLLARY 6.6.3 (LARGE MODEL). *Let $f_{k\ell}$ be an arbitrary formula of $FLCTL_{k\ell}$. Let $I = \{i_1, \ldots, i_K\} \times \{i_1, \ldots, i_K\} - \{(i,i) \mid i \in \{i_1, \ldots, i_K\}\}$, and let $(i,j)$ be an arbitrary pair in $I$. If $M_I, S_I^0 \models AGEXtrue$, then*

$$M_{ij}, S_{ij}^0 \models \textstyle\bigwedge_{k\ell} f_{k\ell} \text{ implies } M_I, S_I^0 \models \textstyle\bigwedge_{k\ell} f_{k\ell}.$$

PROOF. We assume the antecedent $M_{ij}, S_{ij}^0 \models \bigwedge_{k\ell} f_{k\ell}$, and we establish the consequent $M_I, S_I^0 \models \bigwedge_{k\ell} f_{k\ell}$. From $M_{ij}, S_{ij}^0 \models \bigwedge_{k\ell} f_{k\ell}$ and Observation 6.2.2, we have

$$\bigwedge(i,j) \in I \,.\, (M_{ij}, S_{ij}^0 \models \textstyle\bigwedge_{k\ell} f_{k\ell}). \tag{a}$$

Let $s_I^0$ be an arbitrary initial state (i.e., member of $S_I^0$). By the MP-synthesis definition (5.1.1), $\bigwedge(i,j) \in I \,.\, (s_I^0{\uparrow}ij \in S_{ij}^0)$. Thus by (a) we have

$$\bigwedge(i,j) \in I \,.\, (M_{ij}, s_I^0{\uparrow}ij \models \textstyle\bigwedge_{k\ell} f_{k\ell}).$$

Hence, by the large-model theorem (6.6.2), with $s = s_I^0$, we obtain

$$M_I, s_I^0 \models \textstyle\bigwedge_{k\ell} f_{k\ell}.$$

Since $s_I^0$ is an arbitrarily chosen member of $S_I^0$, we have

$$M_I, S_I^0 \models \textstyle\bigwedge_{k\ell} f_{k\ell}.$$ □

## ONLINE-ONLY APPENDICES:

## C. PROOFS

## D. A COMPACT REPRESENTATION FOR SYNCHRONIZATION SKELETONS

## E. CHECKING THE WAIT-FOR-GRAPH ASSUMPTION AND THE LIVENESS ASSUMPTION

Appendices C-E are available only online in a seperate document. You should be able to get the online-only appendix from the citation page for this article:

http://www.acm.org/pubs/citations/journals/toplas/1998-20-1/pxxx-attie/

Alternative instructions on how to obtain online-only appendices are given on the back inside cover of current issues of ACM TOPLAS or on the ACM TOPLAS web page:

http://www.acm.org/toplas

ACKNOWLEDGMENTS

The authors would like to thank Prasad Sistla for very helpful discussions on viewing process indices in terms of spatial modalities. We also thank Mani Chandy, Ed Clarke, and Mike Evangelist for helpful comments. We are also indebted to the referees, whose comments helped to significantly improve the presentation.

REFERENCES

Anuchitanukul, A. and Manna, Z. 1994. Realizability and synthesis of reactive modules. In *Proceedings of the 6th International Conference on Computer Aided Verification*. Lecture Notes in Computer Science, vol. 818. Springer-Verlag, Berlin, 156–169.

Attie, P. C. 1995. Formal methods for the synthesis of concurrent programs from temporal logic specifications. Ph.D. thesis, Dept. of Computer Sciences, The Univ. of Texas at Austin, Austin, Tex.

Attie, P. C., Singh, M., Emerson, E. A., Sheth, A., and Rusinkiewicz, M. 1996. Scheduling workflows by enforcing intertask dependencies. *Distrib. Syst. Eng. J. 3*, 222–238. Special Issue on workflows. Extended abstract appeared in the Proceedings of the 19th VLDB Conference, Dublin, Ireland, August, 1993.

Back, R. J. R. and Kurki-Suonio, R. 1984. Cooperation in distributed systems using symmetric multiprocess handshaking. Tech. Rep. A34, Åbo Akademi, Åbo, Finland.

Back, R. J. R. and Kurki-Suonio, R. 1988. Distributed cooperation with action systems. *ACM Trans. Program. Lang. Syst. 10,* 4 (Oct.), 513–554.

Back, R. J. R. and Kurki-Suonio, R. 1989. Decentralization of process nets with centralized control. *Distrib. Comput. 3*, 73–87. Conference version in Proceedings of the 2nd ACM Symposium on Principles of Programming Languages, Montreal, Canada, August, 1983.

Back, R. J. R., Hartikainen, E., and Kurki-Suonio, R. 1985. Multi-process handshaking on broadcasting networks. Tech. Rep. Ser. A, No. 42, Dept. of Information Processing and Mathematics, Swedish Univ. of Åbo, Åbo, Finland.

Bagrodia, R. 1989. Process synchronization: Design and performance evaluation of distributed algorithms. *IEEE Trans. Softw. Eng. 15,* 9 (Sept.), 1053–1065.

Birman, K. and Clark, T. 1994. Performance of the Isis distributed computing toolkit. Tech. Rep. TR-94-1432, Dept. of Computer Science, Cornell Univ., Ithaca, N.Y.

Browne, M. C., Clarke, E. M., Dill, D. L., and Mishra, B. 1986. Automatic verification of sequential circuits using temporal logic. *IEEE Trans. Comput. C-35,* 12 (Dec.), 1035–1044.

Browne, M. C., Clarke, E. M., and Grumberg, O. 1988. Characterizing finite kripke structures in propositional temporal logic. *Theor. Comput. Sci. 59*, 115–131.

Browne, M. C., Clarke, E. M., Mishra, B., and Dill, D. L. 1990. Sequential circuit verification using symbolic model checking. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*. ACM, New York.

Chandy, K. M. and Misra, J. 1984. The drinking philosophers problem. *ACM Trans. Program. Lang. Syst. 6,* 4 (Oct.), 632–646.

Chandy, K. M. and Misra, J. 1987. Synchronizing asynchronous processes—the committee-coordination problem. Tech. Rep., Dept. of Computer Sciences, The Univ. of Texas at Austin, Austin, Tex.

Chandy, K. M. and Misra, J. 1988. *Parallel Program Design.* Addison-Wesley, Reading, Mass.

Charlesworth, A. 1987. The multiway rendezvous. *ACM Trans. Program. Lang. Syst. 9,* 2 (July), 350–366.

Clarke, E. M. and Grumberg, O. 1987. Avoiding the state explosion problem in temporal logic model checking algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing.* ACM, New York, 294–303.

Clarke, E. M., Burch, J. R., Grumberg, O., Long, D. E., and McMillan, K. L. 1992. Automatic verification of sequential circuit designs. *Phil. Trans. Roy. Soc.* series A, *339*, 105–120.

Clarke, E. M., Emerson, E. A., and Sistla, P. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst. 8,* 2

(Apr.), 244–263. Extended abstract in Proceedings of the 10th Annual ACM Symposium on Principles of Programming Languages.

Clarke, E. M., Filkorn, T., and Jha, S. 1993. Exploiting symmetry in temporal logic model checking. In *Proceedings of the 5th International Conference on Computer Aided Verification*. Lecture Notes in Computer Science, vol. 697. Springer-Verlag, Berlin, 450–462.

Clarke, E. M., Grumberg, O., and Browne, M. C. 1986. Reasoning about networks with many identical finite-state processes. In *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing*. ACM, New York, 240 – 248.

Clarke, E. M., Long, D., and McMillan, K. L. 1989. Compositional model checking. In *Proceedings of the 4th IEEE Symposium on Logic in Computer Science*. IEEE, New York.

Dijkstra, E. W. 1976. *A Discipline of Programming.* Prentice-Hall Inc., Englewood Cliffs, N.J.

Dijkstra, E. W. 1982. *Selected Writings on Computing: A Personal Perspective*. Springer-Verlag, New York, 188–199.

Dijkstra, E. W., Feijen, W. H. J., and van Gasteren, A. J. M. 1983. Derivation of a termination detection algorithm for distributed computations. *Inf. Process. Lett. 16*, 217–219.

Elmagarmid, A., Ed. 1992. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann, San Mateo, Calif.

Emerson, E. A. 1990. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, J. V. Leeuwen, Ed. Vol. B, *Formal Models and Semantics.* The MIT Press/Elsevier, Cambridge, Mass.

Emerson, E. A. and Clarke, E. M. 1982. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program. 2*, 241 – 266.

Emerson, E. A. and Lei, C. 1985. Modalities for model checking: Branching time logic strikes back. Tech. Rep. TR-85-21, Dept. of Computer Sciences, The Univ. of Texas at Austin, Austin, Tex. Oct.

Emerson, E. A. and Sistla, A. P. 1993. Symmetry and model checking. In *Proceedings of the 5th International Conference on Computer Aided Verification*. Lecture Notes in Computer Science, vol. 697. Springer-Verlag, Berlin, 463–477.

Emerson, E. A. and Srinivasan, J. 1990. A decidable temporal logic to reason about many processes. In *Proceedings of the 9th Annual ACM Symposium on the Principles of Distributed Computing.* ACM, New York, 233–246.

Emerson, E. A., Mok, A. K., Sistla, A. P., and Srinivasan, J. 1993. Quantitative temporal reasoning. *Real Time Syst. J. 2*, 331–352.

Emerson, E. A., Sadler, T. H., and Srinivasan, J. 1992. Efficient temporal reasoning. *J. Logic Comput. 2,* 2, 173–210. Extended abstract appears in Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages, 1989, pp. 166–178.

Forman, I. R. 1987. On the design of large distributed systems. Tech. Rep. STP-098-86, Microelectronics and Computer Tecnology Corporation, Austin, Tex. Jan. A preliminary version appears in the 1st International Conference on Computer Languages, Miami, Fla., Oct., 1986.

Francez, N. 1986. *Fairness.* Springer-Verlag, New York.

Francez, N. and Forman, I. 1996. *Interacting Processes, A Multiparty Approach to Coordinated Distributed Programming.* ACM Press, New York.

Francez, N., Hailpern, B. T., and Taubenfeld, G. 1986. Script—A communication abstraction mechanism and its verification. *Sci. Comput. Program. 6,* 1 (Jan.), 35–88.

Hoare, C. A. R. 1969. An axiomatic basis for computer programming. *Commun. ACM 12,* 10, 576–580, 583.

Inaba, Y. 1984. An implementation of synthesizing synchronization skeletons using temporal logic specifications. M.S. thesis, Dept. of Computer Sciences, The Univ. of Texas at Austin, Austin, Tex. Completed under the supervision of E. A. Emerson.

Katz, S. 1986. Temporary stability in parallel programs. Tech. Rep., Computer Science Dept., Technion, Haifa, Israel.

Klein, J. 1991. Advanced rule driven transaction management. In *Proceedings of the IEEE COMPCON.* IEEE, New York.

Knapp, E. 1987. Deadlock detection in distributed databases. *Comput. Surv. 19,* 4 (Dec.), 303–328.

Lynch, N. 1981. Upper bounds for static resource allocation in a distributed system. *J. Comput. Syst. Sci. 23*, 254–278.

Manna, Z. and Wolper, P. 1984. Synthesis of communicating processes from temporal logic specifications. *ACM Trans. Program. Lang. Syst. 6,* 1 (Jan.), 68–93. Also appears in Proceedings of the Workshop on Logics of Programs, Yorktown-Heights, N.Y., Springer-Verlag Lecture Notes in Computer Science (1981).

Milner, R. 1989. *Communication and Concurrency.* Prentice-Hall, Hemel Hempstead, U.K.

Mishra, B. and Clarke, E. M. 1985. Hierarchical verification of asynchronous circuits using temporal logic. *Theor. Comput. Sci. 38*, 269–291.

Nelson, G. 1989. A generalization of Dijkstra's calculus. *ACM Trans. Program. Lang. Syst. 11,* 4 (Oct.), 517–561.

Pnueli, A. and Rosner, R. 1989a. On the synthesis of a reactive module. In *Proceedings of the 16th ACM Symposium on Principles of Programming Languages.* ACM, New York, 179–190.

Pnueli, A. and Rosner, R. 1989b. On the synthesis of asynchronous reactive modules. In *Proceedings of the 16th ICALP.* Lecture Notes in Computer Science, vol. 372. Springer-Verlag, Berlin, 652–671.

Ramesh, S. and Mehndiratta, H. 1987. A methodology for developing distributed programs. *IEEE Trans. Softw. Eng. SE-13,* 8 (Aug.), 967–976.

Sistla, A. P. and German, S. M. 1992. Reasoning about systems with many processes. *J. ACM 39,* 3, 675–735. Conference version appears in IEEE Logic in Computer Science 1987.

Srinivasan, J. 1991. Reasoning about concurrent programs with simplified temporal logics. Ph.D. thesis, Dept. of Computer Sciences, The Univ. of Texas at Austin, Austin, Tex.

Styer, E. and Peterson, G. 1988. Improved algorithms for distributed resource allocation. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing.* ACM, New York.

Wolper, P. 1982. Synthesis of communicating processes from temporal logic specifications. Ph.D. thesis, Stanford Univ., Stanford, Calif.

Received March 1996; revised August 1996; accepted July 1997

THIS DOCUMENT IS THE ON-LINE ONLY APPENDIX TO:

# Synthesis of Concurrent Systems with Many Similar Processes

PAUL C. ATTIE
Florida International University
and
E. ALLEN EMERSON
The University of Texas at Austin

ACM Transactions on Programming Languages and Systems, Vol. 20, No. 1, January 1998, Pages 1–65.

## C. PROOFS

### C.1 Process Similarity

The *process index substitution operator* $\theta = \{j_1/i_1, \ldots, j_m/i_m\}$ denotes the simultaneous replacement of process indices $i_1, \ldots, i_m$ by process indices $j_1, \ldots, j_m$ respectively. We require that $i_1, \ldots, i_m$ be pairwise distinct, and $j_1, \ldots, j_m$ be pairwise distinct. $\theta$ can be applied to all of the pair syntactic constructs defined in the article, as well as any pair model (e.g., $M_{ij}$). We define $\theta$ in a bottom-up manner as follows.

*Definition* C.1.1 (*Process Index Substitution Operator*). The process index substitution operator $\theta = \{j_1/i_1 \ldots j_m/i_m\}$ is defined as follows:

(1) $\theta$ distributes through propositional logic connectives, $=$, $:=$, $\rightarrow$, $//$, and $\cup$.

(2) For any process index $i$:

$$\begin{aligned} i\theta &= j_k \text{ if } i = i_k \text{ for some } k \in [1:m] \\ i\theta &= i \text{ otherwise.} \end{aligned}$$

(3) For any $Q_i \in \mathcal{AP}_i$:

$$Q_i\theta = Q_{i\theta}$$

(4) For any $x_{ij} \in \mathcal{SH}_{ij}$:

$$\begin{aligned} x_{ij}\theta &= x_{i\theta,j\theta} \\ x^i_{ij}\theta &= x^{i\theta}_{i\theta,j\theta} \\ x^j_{ij}\theta &= x^{j\theta}_{i\theta,j\theta} \end{aligned}$$

(5) For any $i$-state $s_i$:

$$s_i\theta = \{< Q_{i\theta}, s_i(Q_i) > \ | \ Q_i \in \mathcal{AP}_i\}$$

i.e., when $i = i_k$, $s_i\theta$ is a $j_k$-state. It satisfies the atomic propositions in $\mathcal{AP}_{j_k}$, which correspond to the atomic propositions in $\mathcal{AP}_i$ that $s_i$ satisfies (remember that the sets of atomic proposition form a uniform family, see Section 3 on MPCTL*). Note that $s_i\theta = s_i$ if $i \neq i_k$ for all $k \in [1:m]$.

(6) For any pair-process $P_i^j$:

$$P_i^j\theta \;=\; \{(s_i\theta, B_i^j\theta \rightarrow A_i^j\theta, t_i\theta) \mid (s_i, B_i^j \rightarrow A_i^j, t_i) \in P_i^j\}$$

(7) For any $ij$-state $s_{ij}$:

$$\begin{aligned} s_{ij}\theta \;=\; & \{< Q_{i\theta}, s_{ij}(Q_i) > \mid Q_i \in \mathcal{AP}_i\} \cup \\ & \{< Q_{j\theta}, s_{ij}(Q_j) > \mid Q_j \in \mathcal{AP}_j\} \cup \\ & \{< x_{ij}\theta, h_{ij}\theta > \mid s_{ij}(x_{ij}) = h_{ij}, x_{ij} \in \mathcal{SH}_{ij}\} \end{aligned}$$

(8) For transition relation $R_{ij}$:

$$R_{ij}\theta = \{(s_{ij}\theta, h_{ij}\theta, t_{ij}\theta) \mid (s_{ij}, h_{ij}, t_{ij}) \in R_{ij}\}$$

(9) For the sets of initial $ij$-states $S_{ij}^0$ and all $ij$-states $S_{ij}$:

$$\begin{aligned} S_{ij}^0\theta &= \{s_{ij}\theta \mid s_{ij} \in S_{ij}^0\} \\ S_{ij}\theta &= \{s_{ij}\theta \mid s_{ij} \in S_{ij}\} \end{aligned}$$

(10) For the pair-structure $M_{ij} = (S_{ij}^0, S_{ij}, R_{ij})$:

$$M_{ij}\theta = (S_{ij}^0\theta, S_{ij}\theta, R_{ij}\theta)$$

PROPOSITION 6.2.1 (PAIR-STRUCTURE SIMILARITY). *Let $i, j, i', j'$ be arbitrary elements of $\{i_1, \ldots, i_K\}$ such that $i\,I\,j$ and $i'\,I\,j'$. Then we have*

$$M_{ij} = M_{i'j'}\{i/i', j/j'\}.$$

PROOF. From the pair-structure definition (5.2.1), we have $M_{i'j'}\{i/i', j/j'\} = (S_{i'j'}^0, S_{i'j'}, R_{i'j'})\{i/i', j/j'\}$. By the process index substitution operator definition (C.1.1), this is equal to $(S_{i'j'}^0\{i/i', j/j'\}$, $S_{i'j'}\{i/i', j/j'\}$, $R_{i'j'}\{i/i', j/j'\})$. Now $S_{i'j'}^0\{i/i', j/j'\} = S_{ij}^0$ by the initial-state assumption. $S_{i'j'}$ is the set of all possible $i'j'$-states. Now $\mathcal{AP}_{i'}, \mathcal{AP}_{j'}$ are similar to $\mathcal{AP}_i, \mathcal{AP}_j$ by assumption (see Section 3.4), and since $P_i^j = P_{i'}^{j'}\{i/i', j/j'\}$, we must have $\mathcal{SH}_{ij} = \mathcal{SH}_{i'j'}\{i/i', j/j'\}$, since $\mathcal{SH}_{ij}$ is merely the set of shared variables the occur in $P_i^j, P_j^i$. Thus, $S_{ij} = S_{i'j'}\{i/i', j/j'\}$, since their domains are related by $\{i/i', j/j'\}$. Finally, by the process similarity assumption and the pair structure definition (5.2.1), we can infer $R_{ij} = R_{i'j'}\{i/i', j/j'\}$, since similar arcs in $P_i^j, P_{i'}^{j'}$ give rise to similar transitions in $R_{ij}, R_{i'j'}$, respectively. $\square$

## C.2 State and Path Projection Results

PROPOSITION 6.3.1 ($I$-STATE PROJECTION). *Let $J \subseteq I$, and let $s$ be an $I$-state. Then*

$$s \models f \text{ iff } s{\uparrow}J \models f$$

*where $f$ is a formula of $\mathcal{L}(\bigcup_{i \in dom(J)} \mathcal{AP}_i,\ \neg,\ \wedge)$.*

PROOF. The proof is by induction on the structure of $f$.

$f \in \bigcup_{i \in dom(J)} \mathcal{AP}_i$. $s \models f$ iff $s(f) = true$ iff (since $s{\uparrow}J = (\bigcup_{i \in dom(J)} s{\uparrow}i) \cup (\bigcup_{(i,j) \in J} s{\uparrow}\mathcal{SH}_{ij})$) $s{\uparrow}J(f) = true$ iff $s{\uparrow}J \models f$.

$f = g \wedge h$. $s \models (g \wedge h)$ iff ($s \models g$ and $s \models h$) iff (by the inductive hypothesis) ($s{\uparrow}J \models g$ and $s{\uparrow}J \models h$) iff $s{\uparrow}J \models (g \wedge h)$.

$f = \neg g$. $s \models \neg g$ iff not ($s \models g$) iff (by the inductive hypothesis) not ($s{\uparrow}J \models g$) iff $s{\uparrow}J \models \neg g$. □

PROPOSITION 6.3.2 (LOCAL STATE PROJECTION). *Let $i \in dom(J)$, and let $s_J$ be a $J$-state. Then*

$$s_J \models f_i \text{ iff } s_J{\uparrow}i \models f_i$$

*where $f_i$ is a formula of $\mathcal{L}(\mathcal{AP}_i, \neg, \wedge)$.*

PROOF. Analogous to that of Proposition 6.3.1. □

LEMMA 6.3.3 (PATH PROJECTION). *Let $\pi$ be a path in $M_I$, and let $J \subseteq I$. Then*

$$\pi \models f \text{ iff } \pi{\uparrow}J \models f$$

*where $f$ is a formula of $\mathcal{L}(\bigcup_{i \in dom(J)} \mathcal{AP}_i, \neg, \wedge, U)$.*

PROOF. The proof is by induction on the structure of $f$.

$f \in \bigcup_{i \in dom(J)} \mathcal{AP}_i$. Let $s, s_J$ be the initial states of $\pi, \pi{\uparrow}J$ respectively. By the definition of path projection, $s_J = s{\uparrow}J$, and so $s_J$ and $s$ agree on all atomic propositions in $\bigcup_{i \in dom(J)} \mathcal{AP}_i$. Hence $s \models f$ iff $s_J \models f$. Also $\pi \models f$ iff $s \models f$ and $\pi{\uparrow}J \models f$ iff $s_J \models f$ by CTL$^*$ semantics (Section 3.1). These three equivalences yield $\pi \models f$ iff $\pi{\uparrow}J \models f$.

$f = g \wedge h$. $\pi \models (g \wedge h)$ iff ($\pi \models g$ and $\pi \models h$) iff, by the inductive hypothesis, ($\pi{\uparrow}J \models g$ and $\pi{\uparrow}J \models h$) iff $\pi{\uparrow}J \models (g \wedge h)$.

$f = \neg g$. $\pi \models \neg g$ iff not($\pi \models g$) iff, by the inductive hypothesis, not($\pi{\uparrow}J \models g$) iff $\pi{\uparrow}J \models \neg g$.

$f = gUh$. Proof by double implication.

*Left to right*: $\pi \models gUh$ implies $\pi{\uparrow}J \models gUh$.

Assume $\pi \models gUh$. Therefore, for some $n_0 \geq 1$, we have by CTL$^*$ semantics (Section 3.1):

$$\pi^{n_0} \models h \text{ and } \bigwedge n\,.\,(1 \leq n < n_0 \Rightarrow \pi^n \models g). \tag{a}$$

By (a) and the induction hypothesis, we get

$$\pi^{n_0}{\uparrow}J \models h \text{ and } \bigwedge n\,.\,(1 \leq n < n_0 \Rightarrow \pi^n{\uparrow}J \models g). \tag{b}$$

Let $B^{m_0}$ denote the $J$-block of $\pi$ that contains $s^{n_0}$ (the first state of $\pi^{n_0}$). By the path projection definition (5.1.2.1), $\pi^{n_0}{\uparrow}J = (\pi{\uparrow}J)^{m_0}$, since the $m_0$'th $J$-block of $\pi$ corresponds to the $m_0$'th state of $\pi{\uparrow}J$. So, by (b) we have

$$(\pi{\uparrow}J)^{m_0} \models h. \tag{c}$$

Let $m$ be an arbitrary integer in $[1 : (m_0 - 1)]$. The first state $s_J^m$ of $(\pi{\uparrow}J)^m$ corresponds to the $m$th $J$-block of $\pi$ (by the path projection definition (5.1.2.1)). Let $s^l$ (the first state of $\pi^l$) be an arbitrary state of the $m$th $J$-block of $\pi$. Hence

$(\pi{\uparrow}J)^m = \pi^l{\uparrow}J$. Also, since $m < m_0$, $s^l$ occurs in an earlier $J$-block of $\pi$ than $s^{n_0}$, and therefore $l < n_0$. So by (b) we have

$$\pi^l{\uparrow}J \models g. \qquad \text{(d)}$$

By (d) and $(\pi{\uparrow}J)^m = \pi^l{\uparrow}J$, we get

$$(\pi{\uparrow}J)^m \models g. \qquad \text{(e)}$$

But $m$ is an arbitrary integer in $[1 : (m_0 - 1)]$. So, by (c) and (e) we get

$$(\pi{\uparrow}J)^{m_0} \models h \text{ and } \bigwedge m \,.\, (1 \leq m < m_0 \Rightarrow (\pi{\uparrow}J)^m \models g).$$

By CTL* semantics (Section 3.1), this is equivalent to $\pi{\uparrow}J \models gUh$.

*Right to left*: $\pi{\uparrow}J \models gUh$ implies $\pi \models gUh$.

Assume $\pi{\uparrow}J \models gUh$. Therefore, for some $m_0 \geq 1$, we have by CTL* semantics (Section 3.1)

$$(\pi{\uparrow}J)^{m_0} \models h \text{ and } \bigwedge m \,.\, (1 \leq m < m_0 \Rightarrow (\pi{\uparrow}J)^m \models g). \qquad \text{(a)}$$

Let $s^{n_0}$ (the first state of $\pi^{n_0}$) be the first state of the $m_0$th $J$-block of $\pi$. By the path projection definition (5.1.2.1), $\pi^{n_0}{\uparrow}J = (\pi{\uparrow}J)^{m_0}$. Hence, by (a) we have

$$\pi^{n_0}{\uparrow}J \models h. \qquad \text{(b)}$$

Now let $m$ be an arbitrary integer in $[1 : (m_0 - 1)]$. By (a), we get

$$(\pi{\uparrow}J)^m \models g. \qquad \text{(c)}$$

Furthermore, let $s^n$ (the first state of $\pi^n$) be an arbitrary state of the $m$th $J$-block of $\pi$. By the path projection definition (5.1.2.1), $\pi^n{\uparrow}J = (\pi{\uparrow}J)^m$. Also, since $m < m_0$, $s^n$ occurs in an earlier $J$-block of $\pi$ than $s^{n_0}$, and therefore $n < n_0$. Furthermore, since $s^{n_0}$ is the *first* state of the $m_0$th $J$-block of $\pi$, we see that $n$ ranges over $[1 : (n_0 - 1)]$, since $s^{n_0 - 1}$ occurs in an earlier $J$-block of $\pi$ than $s^{n_0}$. Hence, by (c) and $\pi^n{\uparrow}J = (\pi{\uparrow}J)^m$ we have

$$\bigwedge n \,.\, (1 \leq n < n_0 \Rightarrow \pi^n{\uparrow}J \models g). \qquad \text{(d)}$$

From (b) and (d) we get

$$\pi^{n_0}{\uparrow}J \models h \text{ and } \bigwedge n \,.\, (1 \leq n < n_0 \Rightarrow \pi^n{\uparrow}J \models g). \qquad \text{(e)}$$

From (e) and the induction hypothesis applied to $g, h$, we get

$$\pi^{n_0} \models h \text{ and } \bigwedge n \,.\, (1 \leq n < n_0 \Rightarrow \pi^n \models g).$$

By CTL* semantics (Section 3.1), this is equivalent to $\pi \models gUh$. □

## C.3 Mapping of $I$-Structures into $J$-Structures

LEMMA 6.4.1 (TRANSITION MAPPING). *For all $I$-states $s, t \in S_I$ and $i \in dom(I)$,*

$$\begin{aligned} & s \xrightarrow{i} t \in R_I \text{ iff :} \\ & \quad \bigwedge j \in I(i) \,.\, (s{\uparrow}ij \xrightarrow{i} t{\uparrow}ij \in R_{ij}) \textit{ and} \\ & \quad \bigwedge j \in dom(I) - \hat{I}(i) \,.\, (s{\uparrow}j = t{\uparrow}j) \textit{ and} \\ & \quad \bigwedge j, k \in dom(I) - \{i\}, j\, I\, k \,.\, (s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}). \end{aligned}$$

PROOF. Let $s, t$ be arbitrary $I$-states, and let $i$ be an arbitrary element of $dom(I)$. By the $I$-structure definition (5.3.1), $s \xrightarrow{i} t \in R_I$ is equivalent to

$$\begin{aligned}
&(s{\uparrow}i, \textstyle\bigwedge_{j\in I(i)} B_i^j \rightarrow //_{j\in I(i)} A_i^j, t{\uparrow}i) \text{ is an arc in } P_i^I \text{ and}\\
&\textstyle\bigwedge j \in I(i) \,.\, (s{\uparrow}ij(B_i^j) = \mathit{true} \text{ and } < s{\uparrow}\mathcal{SH}_{ij} > A_i^j < t{\uparrow}\mathcal{SH}_{ij} >) \text{ and}\\
&\textstyle\bigwedge j \in dom(I) - \{i\} \,.\, (s{\uparrow}j = t{\uparrow}j) \text{ and}\\
&\textstyle\bigwedge j,k \in dom(I) - \{i\}, j\, I\, k \,.\, (s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}). \qquad \text{(a)}
\end{aligned}$$

By the MP-synthesis definition (5.1.1), (a) is equivalent to

$$\begin{aligned}
&\textstyle\bigwedge j \in I(i) \,.\, ((s{\uparrow}i, B_i^j \rightarrow A_i^j, t{\uparrow}i) \text{ is an arc in } P_i^j) \text{ and}\\
&\textstyle\bigwedge j \in I(i) \,.\, (s{\uparrow}ij(B_i^j) = \mathit{true} \text{ and } < s{\uparrow}\mathcal{SH}_{ij} > A_i^j < t{\uparrow}\mathcal{SH}_{ij} >) \text{ and}\\
&\textstyle\bigwedge j \in dom(I) - \{i\} \,.\, (s{\uparrow}j = t{\uparrow}j) \text{ and}\\
&\textstyle\bigwedge j,k \in dom(I) - \{i\}, j\, I\, k \,.\, (s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}). \qquad \text{(b)}
\end{aligned}$$

Now (b), by rewriting the "$\bigwedge j \in dom(I) - \{i\}$..." universal quantification as the conjunction of "for all $j$ in $I(i)$..." and "$\bigwedge j \in dom(I) - \hat{I}(i)$...", is equivalent to

$$\begin{aligned}
&\textstyle\bigwedge j \in I(i) \,.\, ((s{\uparrow}i, B_i^j \rightarrow A_i^j, t{\uparrow}i) \text{ is an arc in } P_i^j) \text{ and}\\
&\textstyle\bigwedge j \in I(i) \,.\, (s{\uparrow}ij(B_i^j) = \mathit{true} \text{ and } < s{\uparrow}\mathcal{SH}_{ij} > A_i^j < t{\uparrow}\mathcal{SH}_{ij} >) \text{ and}\\
&\textstyle\bigwedge j \in I(i) \,.\, (s{\uparrow}j = t{\uparrow}j) \text{ and}\\
&\textstyle\bigwedge j \in dom(I) - \hat{I}(i) \,.\, (s{\uparrow}j = t{\uparrow}j) \text{ and}\\
&\textstyle\bigwedge j,k \in dom(I) - \{i\}, j\, I\, k \,.\, (s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}). \qquad \text{(c)}
\end{aligned}$$

By merging all three "$\bigwedge j \in I(i)$..." universal quantifications, (c) is equivalent to

$$\begin{aligned}
&\textstyle\bigwedge j \in I(i) \,.\, (\\
&\qquad (s{\uparrow}i, B_i^j \rightarrow A_i^j, t{\uparrow}i) \text{ is an arc in } P_i^j \text{ and}\\
&\qquad s{\uparrow}ij(B_i^j) = \mathit{true} \text{ and}\\
&\qquad < s{\uparrow}\mathcal{SH}_{ij} > A_i^j < t{\uparrow}\mathcal{SH}_{ij} > \text{ and}\\
&\qquad s{\uparrow}j = t{\uparrow}j \;) \text{ and}\\
&\textstyle\bigwedge j \in dom(I) - \hat{I}(i) \,.\, (s{\uparrow}j = t{\uparrow}j) \text{ and}\\
&\textstyle\bigwedge j,k \in dom(I) - \{i\}, j\, I\, k \,.\, (s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}). \qquad \text{(d)}
\end{aligned}$$

By the obvious identities $s{\uparrow}i = (s{\uparrow}ij){\uparrow}i$, $s{\uparrow}\mathcal{SH}_{ij} = (s{\uparrow}ij){\uparrow}\mathcal{SH}_{ij}$, $s{\uparrow}j = (s{\uparrow}ij){\uparrow}j$, $t{\uparrow}i = (t{\uparrow}ij){\uparrow}i$, $t{\uparrow}\mathcal{SH}_{ij} = (t{\uparrow}ij){\uparrow}\mathcal{SH}_{ij}$, $t{\uparrow}j = (t{\uparrow}ij){\uparrow}j$, (d) is equivalent to

$$\begin{aligned}
&\textstyle\bigwedge j \in I(i) \,.\, (\\
&\qquad ((s{\uparrow}ij){\uparrow}i, B_i^j \rightarrow A_i^j, (t{\uparrow}ij){\uparrow}i) \text{ is an arc in } P_i^j \text{ and}\\
&\qquad s{\uparrow}ij(B_i^j) = \mathit{true} \text{ and}\\
&\qquad < (s{\uparrow}ij){\uparrow}\mathcal{SH}_{ij} > A_i^j < (t{\uparrow}ij){\uparrow}\mathcal{SH}_{ij} > \text{ and}\\
&\qquad (s{\uparrow}ij){\uparrow}j = (t{\uparrow}ij){\uparrow}j \;) \text{ and}\\
&\textstyle\bigwedge j \in dom(I) - \hat{I}(i) \,.\, (s{\uparrow}j = t{\uparrow}j) \text{ and}\\
&\textstyle\bigwedge j,k \in dom(I) - \{i\}, j\, I\, k \,.\, (s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}). \qquad \text{(e)}
\end{aligned}$$

By the pair-structure definition (5.2.1), (e) is equivalent to

$$\begin{aligned}
&\textstyle\bigwedge j \in I(i) \,.\, (s{\uparrow}ij \xrightarrow{i} t{\uparrow}ij \in R_{ij}) \text{ and}\\
&\textstyle\bigwedge j \in dom(I) - \hat{I}(i) \,.\, (s{\uparrow}j = t{\uparrow}j) \text{ and}\\
&\textstyle\bigwedge j,k \in dom(I) - \{i\}, j\, I\, k \,.\, (s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}).
\end{aligned}$$

The lemma then follows by transitivity of equivalence. □

COROLLARY 6.4.2 (TRANSITION MAPPING). *Let* $J \subseteq I$ *and* $i \in dom(J)$. *If* $s \xrightarrow{i} t \in R_I$, *then* $s{\uparrow}J \xrightarrow{i} t{\uparrow}J \in R_J$.

PROOF. Assume $s \xrightarrow{i} t \in R_I$. Then, by the transition-mapping lemma (6.4.1), we have

$$\begin{aligned}
&\textstyle\bigwedge j \in I(i) \,.\, (s{\uparrow}ij \xrightarrow{i} t{\uparrow}ij \in R_{ij}) \text{ and}\\
&\textstyle\bigwedge j \in dom(I) - \hat{I}(i) \,.\, (s{\uparrow}j = t{\uparrow}j) \text{ and}\\
&\textstyle\bigwedge j,k \in dom(I) - \{i\}, j\, I\, k \,.\, (s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}).
\end{aligned} \tag{a}$$

From $\bigwedge j \in I(i) \,.\, (s{\uparrow}ij \xrightarrow{i} t{\uparrow}ij \in R_{ij})$ and the pair-structure definition (5.2.1), we get $\bigwedge j \in I(i) \,.\, (s{\uparrow}j = t{\uparrow}j)$. Together with (a), this yields

$$\begin{aligned}
&\textstyle\bigwedge j \in I(i) \,.\, (s{\uparrow}ij \xrightarrow{i} t{\uparrow}ij \in R_{ij}) \text{ and}\\
&\textstyle\bigwedge j \in dom(I) - \{i\} \,.\, (s{\uparrow}j = t{\uparrow}j) \text{ and}\\
&\textstyle\bigwedge j,k \in dom(I) - \{i\}, j\, I\, k \,.\, (s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}).
\end{aligned} \tag{b}$$

Since $J \subseteq I$, we have $dom(J) \subseteq dom(I)$. Now $i \in dom(J)$ by assumption, and so $i \in dom(I)$. Thus, by $J \subseteq I$, we have $J(i) \subseteq I(i)$ and $\bigwedge j,k \,.\, (j\, J\, k$ implies $j\, I\, k)$. Thus, from (b) we get

$$\begin{aligned}
&\textstyle\bigwedge j \in J(i) \,.\, (s{\uparrow}ij \xrightarrow{i} t{\uparrow}ij \in R_{ij}) \text{ and}\\
&\textstyle\bigwedge j \in dom(I) - \{i\} \,.\, (s{\uparrow}j = t{\uparrow}j) \text{ and}\\
&\textstyle\bigwedge j,k \in dom(J) - \{i\}, j\, J\, k \,.\, (s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}).
\end{aligned} \tag{c}$$

Since $J \subseteq I$, we have $dom(J) \subseteq dom(I)$. Also, $i \in \hat{J}(i)$, and hence $dom(J) - \hat{J}(i) \subseteq dom(I) - \{i\}$. Thus, by (c) we have

$$\begin{aligned}
&\textstyle\bigwedge j \in J(i) \,.\, (s{\uparrow}ij \xrightarrow{i} t{\uparrow}ij \in R_{ij}) \text{ and}\\
&\textstyle\bigwedge j \in dom(J) - \hat{J}(i) \,.\, (s{\uparrow}j = t{\uparrow}j) \text{ and}\\
&\textstyle\bigwedge j,k \in dom(J) - \{i\}, j\, J\, k \,.\, (s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}).
\end{aligned} \tag{d}$$

Now $s{\uparrow}ij = (s{\uparrow}J){\uparrow}ij$ when $j \in I(i)$, and $s{\uparrow}j = (s{\uparrow}J){\uparrow}j$ when $j \in dom(J)$, and $s{\uparrow}\mathcal{SH}_{jk} = (s{\uparrow}J){\uparrow}\mathcal{SH}_{jk}$ when $j\, J\, k$. Thus, (d) can be rewritten as

$$\begin{aligned}
&\textstyle\bigwedge j \in J(i) \,.\, ((s{\uparrow}J){\uparrow}ij \xrightarrow{i} (t{\uparrow}J){\uparrow}ij \in R_{ij}) \text{ and}\\
&\textstyle\bigwedge j \in dom(J) - \hat{J}(i) \,.\, ((s{\uparrow}J){\uparrow}j = (t{\uparrow}J){\uparrow}j) \text{ and}\\
&\textstyle\bigwedge j,k \in dom(J) - \{i\}, j\, J\, k \,.\, ((s{\uparrow}J){\uparrow}\mathcal{SH}_{jk} = (t{\uparrow}J){\uparrow}\mathcal{SH}_{jk}).
\end{aligned} \tag{e}$$

By the transition-mapping lemma (6.4.1) with $I := J$, and (e), we get $s{\uparrow}J \xrightarrow{i} t{\uparrow}J \in R_J$ as required. □

LEMMA 6.4.3 (PATH MAPPING). *Let* $J \subseteq I$. *If* $\pi$ *is a path in* $M_I$, *then* $\pi{\uparrow}J$ *is a path in* $M_J$.

PROOF. Let $n$ be an arbitrary integer greater than zero, and let $u_J^n \xrightarrow{d_n} u_J^{n+1}$ be the $n$th transition along $\pi{\uparrow}J$. Thus $d_n \in dom(J)$, and $u_J^n, u_J^{n+1}$ are the $n$th, $(n+1)$st states respectively along $\pi{\uparrow}J$. By the path projection definition (5.1.2.1), $u_J^n = B^n{\uparrow}J$, $u_J^{n+1} = B^{n+1}{\uparrow}J$, where $B^n$, $B^{n+1}$ are the $n$th, $(n+1)$st, $J$-blocks respectively along $\pi$. Let $s$, $t$, be the last, first states of $B^n$, $B^{n+1}$, respectively, so $s{\uparrow}J = B^n{\uparrow}J = u_J^n$, $t{\uparrow}J = B^{n+1}{\uparrow}J = u_J^{n+1}$. Also, by the path projection definition (5.1.2.1), $s \xrightarrow{d_n} t$

is a transition along $\pi$, and therefore $s \xrightarrow{d_n} t \in R_I$, since $\pi$ is a path in $M_I$. Hence, by the transition-mapping corollary (6.4.2), $s{\uparrow}J \xrightarrow{d_n} t{\uparrow}J \in R_J$, so $u_J^n \xrightarrow{d_n} u_J^{n+1} \in R_J$.

Since every transition of $\pi{\uparrow}J$ is a transition in $R_J$, it follows that $\pi{\uparrow}J$ is a path in $M_J$. $\square$

Corollary 6.4.4 (Path Mapping). *Let $J \subseteq I$. If $\pi$ is an initialized path in $M_I$ then $\pi{\uparrow}J$ is an initialized path in $M_J$.*

Proof. Let $B^0$ be the first $J$-block of $\pi$. By the path projection definition (5.1.2.1), the first state of $\pi{\uparrow}J$ is $B^0{\uparrow}J$. Since $\pi$ is initialized, $B^0$ contains some initial state $s_I^0 \in S_I^0$. Hence $B^0{\uparrow}J = s_I^0{\uparrow}J$.

Now by the MP-synthesis definition (5.1.1), we have $\bigwedge(i,j) \in I\,.\,(s_I^0{\uparrow}ij \in S_{ij}^0)$. Since $J \subseteq I$, we have $\bigwedge(i,j) \in J\,.\,(s_I^0{\uparrow}ij \in S_{ij}^0)$. Also, by definition of ${\uparrow}J$ (Section 6), $\bigwedge(i,j) \in J\,.\,((s_I^0{\uparrow}J){\uparrow}ij = s_I^0{\uparrow}ij)$. Hence, we have $\bigwedge(i,j) \in J\,.\,((s_I^0{\uparrow}J){\uparrow}ij \in S_{ij}^0)$. Now, by the MP-synthesis definition (5.1.1) with $J$ replacing $I$ we have $S_J^0 = \{s_J \mid \bigwedge(i,j) \in J\,.\,(s_J{\uparrow}ij \in S_{ij}^0)\}$. Hence we conclude $s_I^0{\uparrow}J \in S_J^0$. Thus, the first state of $\pi{\uparrow}J$ is an initial state of $M_J$. By the path-mapping lemma (6.4.3), $\pi{\uparrow}J$ is a path in $M_J$. It follows that $\pi{\uparrow}J$ is an initialized path in $M_J$. $\square$

Corollary 6.4.5 (State Mapping). *Let $J \subseteq I$. If $t$ is a reachable state in $M_I$, then $t{\uparrow}J$ is a reachable state in $M_J$.*

Proof. Since $t$ is reachable in $M_I$, there must exist at least one initialized path $\pi$ in $M_I$ which ends in state $t$. By the path-mapping corollary (6.4.4), $\pi{\uparrow}J$ is an initialized path in $M_J$. By the path projection definition (5.1.2.1), $\pi{\uparrow}J$ ends in state $t{\uparrow}J$. Hence $t{\uparrow}J$ is reachable in $M_J$. $\square$

Corollary 6.4.6 (Relativized State Mapping). *Let $J \subseteq I$. If $t$ is a $s$-reachable state in $M_I$, then $t{\uparrow}J$ is a $s{\uparrow}J$-reachable state in $M_J$.*

Proof. Since $t$ is $s$-reachable in $M_I$, there must exist at least one path $\pi$ in $M_I$ which starts in state $s$ and ends in state $t$. By the path-mapping lemma (6.4.3), $\pi{\uparrow}J$ is a path in $M_J$. By the path projection definition (5.1.2.1), $\pi{\uparrow}J$ starts in state $s{\uparrow}J$ and ends in state $t{\uparrow}J$. Hence $t{\uparrow}J$ is $s{\uparrow}J$-reachable in $M_J$. $\square$

## C.4 Deadlock Freedom of the Many-Process System

Proposition 6.5.4.1 (Wait-For-Graph Projection). *Let $J \subseteq I$ and $i\,J\,j$. Furthermore, let $s_I$ be an arbitrary $I$-state. Then*

*(1)* $P_i^I \longrightarrow a_i^I \in W_I(s_I)$ *iff* $P_i^J \longrightarrow a_i^J \in W_J(s_I{\uparrow}J)$, *and*
*(2)* $a_i^I \longrightarrow P_j^I \in W_I(s_I)$ *iff* $a_i^J \longrightarrow P_j^J \in W_J(s_I{\uparrow}J)$.

Proof. By assumption, $i\,J\,j$ and $J \subseteq I$. Hence $i\,I\,j$.

Proof of clause (1). By the wait-for-graph definition (6.5.2.1), $P_i^I \longrightarrow a_i^I \in W_I(s_I)$ iff $s_I{\uparrow}i = a_i^I.start$. Since $i \in dom(J)$, we have $(s_I{\uparrow}J){\uparrow}i = s_I{\uparrow}i$ by definition of ${\uparrow}J$. Thus $s_I{\uparrow}i = a_i^I.start$ iff $(s_I{\uparrow}J){\uparrow}i = a_i^J.start$ (since $a_i^I.start = a_i^J.start = s_i$). Finally, by the wait-for-graph definition (6.5.2.1) and $i\,J\,j$, $(s_I{\uparrow}J){\uparrow}i = a_i^J.start$ iff

$P_i^J \longrightarrow a_i^J \in W_J(s_I{\uparrow}J)$. These three equivalences together yield clause (1) (using transitivity of equivalence).

Proof of clause (2). By the wait-for-graph definition (6.5.2.1), $a_i^I \longrightarrow P_j^I \in W_I(s_I)$ iff $s{\uparrow}ij \not\models a_i^I.guard_j$. Since $i\,J\,j$, we have $(s_I{\uparrow}J){\uparrow}ij = s_I{\uparrow}ij$ by definition of ${\uparrow}J$. Also, $a_i^I.guard_j = a_i^J.guard_j = \bigvee_{\ell\in[1:n]} B^j_{i,\ell}$. Thus $s_I{\uparrow}ij \not\models a_i^I.guard_j$ iff $(s_I{\uparrow}J){\uparrow}ij \not\models a_i^J.guard_j$ Finally, by the wait-for-graph definition (6.5.2.1) and $i\,J\,j$, $(s_I{\uparrow}J){\uparrow}ij \not\models a_i^J.guard_j$ iff $a_i^J \longrightarrow P_j^J \in W_J(s_I{\uparrow}J)$. These three equivalences together yield clause (2), (using transitivity of equivalence, and noting that $s \not\models B$ and $s(B) = false$ have identical meaning). $\square$

Theorem 6.5.4.3 (Supercycle-Free Wait-For-Graph). *If the wait-for-graph assumption $WG$ holds, and $W_I(s_I^0)$ is supercycle-free for every initial state $s_I^0 \in S_I^0$, then for every reachable state $t$ of $M_I$, $W_I(t)$ is supercycle-free.*

Proof. Let $t$ be an arbitrary reachable state of $M_I$, and let $s$ be an arbitrary reachable state of $M_I$ such that $s \xrightarrow{k} t$ for some $k \in dom(I)$. We shall establish that

$$\text{if } W_I(t) \text{ is supercyclic, then } W_I(s) \text{ is supercyclic.} \tag{P1}$$

The contrapositive of P1 together with the assumption that $W_I(s_I^0)$ is supercycle-free for all $s_I^0 \in S_I^0$ is sufficient to establish the conclusion of the theorem (by induction on the length of a path from some $s_I^0 \in S_I^0$ to $t$).

We say that an edge is *$k$-incident* iff at least one of its vertices is $P_k^I$ or $a_k^I$. The following (P2) will be useful in proving P1

$$\text{if edge } e \text{ is not } k\text{-incident, then } e \in W_I(t) \text{ iff } e \in W_I(s). \tag{P2}$$

Proof of P2. If $e$ is not $k$-incident, then, by the wait-for-graph definition (6.5.2.1), either $e = P_h^I \longrightarrow a_h^I$, or $e = a_h^I \longrightarrow P_\ell^I$, for some $h, \ell$ such that $h \neq k, \ell \neq k$. From $h \neq k, \ell \neq k$ and $s \xrightarrow{k} t \in R_I$, we have $s{\uparrow}h = t{\uparrow}h$ and $s{\uparrow}h\ell = t{\uparrow}h\ell$ by the wait-for-graph definition (6.5.2.1). Since $e \in W_I(t), e \in W_I(s)$ are determined solely by $t{\uparrow}h\ell, s{\uparrow}h\ell$ respectively, (see the wait-for-graph definition (6.5.2.1)), P2 follows. (End proof of P2.)

Let $v$ be a vertex in a supercycle $SC$. We define $depth_{SC}(v)$ to be the length of the longest backward path in $SC$ which starts in $v$. If there exists an infinite backward path (i.e., one that traverses a cycle) in $SC$ starting in $v$, then $depth_{SC}(v) = \omega$ ($\omega$ for "infinity"). We now establish that

$$\text{every supercycle } SC \text{ contains at least one cycle.} \tag{P3}$$

Proof of P3. Suppose P3 does not hold, and $SC$ is a supercycle containing no cycles. Therefore, all backward paths in $SC$ are finite, and so by definition of $depth_{SC}$ all vertices of $SC$ have finite depth. Thus, there is at least one vertex $v$ in $SC$ with maximal depth. But, by definition of $depth_{SC}$, $v$ has no successors in $SC$, which, by the supercycle definition (6.5.3.1), contradicts the assumption that $SC$ is a supercycle. (End proof of P3.)

Our final prerequisite for the proof of P1 is if $SC$ is a supercycle in $W_I(s)$, then the graph $SC'$ obtained from $SC$ by removing all vertices of finite depth from $SC$ (along with incident edges) is also a supercycle in $W_I(s)$. (P4)

Proof of P4. By P3, $SC' \neq \emptyset$. Thus $SC'$ satisfies clause (1) of the supercycle definition (6.5.3.1). Let $v$ be an arbitrary vertex of $SC'$. Thus $v \in SC$ and $depth_{SC}(v) = \omega$ by definition of $SC'$. Let $w$ be an arbitrary successor of $v$ in $SC$. $depth_{SC}(w) = \omega$ by definition of *depth*. Hence $w \in SC'$. Furthermore, $w$ is a successor of $v$ in $SC'$, by definition of $SC'$. Thus every vertex $v$ of $SC'$ is also a vertex of $SC$, and the successors of $v$ in $SC'$ are the same as the successors of $v$ in $SC$ Now since $SC$ is a supercycle, every vertex $v$ in $SC$ has enough successors in $SC$ to satisfy clauses (2) and (3) of the supercycle definition (6.5.3.1). It follows that every vertex $v$ in $SC'$ has enough successors in $SC'$ to satisfy clauses (2) and (3) of the supercycle definition (6.5.3.1). (End proof of P4.)

We now present the proof of (P1). We assume the antecedent of P1 and establish the consequent. Let $SC$ be some supercycle in $W_I(t)$. Let $SC'$ be the graph obtained from $SC$ by removing all vertices of finite depth from $SC$ (along with incident edges). We now show that $P_k^I \notin SC'$ and that $SC'$ contains no move vertex of the form $a_k^I$. There are two cases.

*Case 1:* $P_k^I \notin SC$. Then obviously $P_k^I \notin SC'$. Now suppose some node of the form $a_k^I$ is in $SC'$. By definition of $SC'$, we have $a_k^I \in SC$ and $depth_{SC}(a_k^I) = \omega$. Hence, by definition of *depth*, there exists an infinite backward path in $SC$ starting in $a_k^I$. Thus $a_k^I$ must have a predecessor in $SC$. By the supercycle definition (6.5.3.1), $P_k^I$ is the only possible predecessor of $a_k^I$ in $SC$, and hence $P_k^I \in SC$, contrary to the case assumption. We therefore conclude that $SC'$ contains no vertices of the form $a_k^I$. (End of case 1.)

*Case 2:* $P_k^I \in SC$. By the supercycle definition (6.5.3.1),

$$\bigwedge a_k^I \in W_I(t) \,.\, (\bigvee \ell \,.\, (a_k^I \longrightarrow P_\ell^I \in W_I(t))). \tag{a}$$

Since there are exactly $n$ moves $a_k^I$ of process $P_k^I$ in $W_I(t)$ ($n = |t_k.moves|$), we can select $\ell_1, \ldots, \ell_n$ (where $\ell_1, \ldots, \ell_n$ are not necessarily pairwise distinct) such that

$$\bigwedge a_k^I \in W_I(t) \,.\, (\bigvee \ell \in \{\ell_1, \ldots, \ell_n\} \,.\, (a_k^I \longrightarrow P_\ell^I \in W_I(t))). \tag{b}$$

Now let $J = \{\{j,k\}, \{k,\ell_1\}, \ldots, \{k,\ell_n\}\}$ where $j$ is an arbitrary element of $I(k)$. Applying the wait-for-graph projection proposition (6.5.4.1) to (b) gives us

$$\bigwedge a_k^J \in W_J(t{\uparrow}J) \,.\, (\bigvee \ell \in \{\ell_1, \ldots, \ell_n\} \,.\, (a_k^J \longrightarrow P_\ell^J \in W_J(t{\uparrow}J))). \tag{c}$$

Now $s \xrightarrow{k} t \in R_I$ by assumption. Hence $s{\uparrow}J \xrightarrow{k} t{\uparrow}J \in R_J$ by the transition-mapping corollary (6.4.2). Also, by the state-mapping corollary (6.4.5) $s{\uparrow}J$ is reachable in $M_J$, since $s$ is reachable in $M_I$. Thus we can apply the wait-for-graph assumption to $t{\uparrow}J$ to get

$$\bigwedge a_j^J \,.\, (a_j^J \longrightarrow P_k^J \notin W_J(t{\uparrow}J))$$

or

$$\bigvee a_k^J \in W_J(t{\uparrow}J) \,.\, (\bigwedge \ell \in \{\ell_1, \ldots, \ell_n\} \,.\, (a_k^J \longrightarrow P_\ell^J \notin W_J(t{\uparrow}J))). \tag{d}$$

Now (c) contradicts the second disjunct of (d). Hence

$$\bigwedge a_j^J \,.\, (a_j^J \longrightarrow P_k^J \notin W_J(t{\uparrow}J)),$$

and applying the wait-for-graph projection proposition (6.5.4.1) to this gives us

$$\bigwedge a_j^J \,.\, (a_j^I \longrightarrow P_k^I \notin W_I(t)).$$

Since $j$ is an arbitrary element of $I(k)$, we conclude that $P_k^I$ has no incoming edges in $W_I(t)$. Thus, by definition of *depth*, $depth_{SC}(P_k^I) = 0$, and so $P_k^I \notin SC'$.

Now suppose some node of the form $a_k^I$ is in $SC'$. By definition of $SC'$, we have $a_k^I \in SC$ and $depth_{SC}(a_k^I) = \omega$. Hence, by definition of *depth*, there exists an infinite backward path in $SC$ starting in $a_k^I$. Thus $a_k^I$ must have a predecessor in $SC$. By the supercycle definition (6.5.3.1), $P_k^I$ is the only possible predecessor of $a_k^I$ in $SC$, and hence there exists an infinite backward path in $SC$ starting in $P_k^I$. Thus $depth_{SC}(P_k^I) = \omega$ by definition of *depth*. But we have established $depth_{SC}(P_k^I) = 0$, so we conclude that $SC'$ contains no vertices of the form $a_k^I$. (End of case 2.)

In both cases, $P_k^I \notin SC'$, and $SC'$ contains no move vertex of the form $a_k^I$. Thus every edge of $SC'$ is not $k$-incident. Hence, by P2, every edge of $SC'$ is an edge of $W_I(s)$ (since $SC' \subseteq W_I(t)$). By P4, $SC'$ is a supercycle, so $W_I(s)$ is supercyclic. Thus P1 is established, which establishes the theorem. □

## C.5 Liveness Properties

PROPOSITION 6.7.3.4 (SOMETIMES-BLOCKING). *Let $s_{ij}$ be a reachable state of $M_{ij}$ and $r_i \in \mathcal{L}(\mathcal{AP}_i, \neg, \wedge)$. If $M_{ij}, s_{ij} \models \neg r_i \wedge AFr_i$, then $s_{ij}{\uparrow}i$ is sometimes-blocking.*

PROOF. We assume the antecedent and establish the consequent. Define a $P_j^i$-path to be a path in $M_{ij}$ composed entirely of $P_j^i$-transitions. For a pair move $a_i^j = (s_i, \oplus_{\ell\in[1:n]} B_{i,\ell}^j \rightarrow A_{i,\ell}^j, t_i)$, let $a_i^j.start, a_i^j.guard$ denote $s_i, \bigvee_{\ell\in[1:n]} B_{i,\ell}^j$ respectively.

By assumption, $s_{ij} \not\models r_i$. Also, by $M_{ij}$, $s_{ij} \models AFr_i$ and CTL* semantics, every path starting in $s_{ij}$ must lead to a state $t_{ij}$ which fulfills $AFr_i$, i.e., $t_{ij} \models r_i$. Since $s_{ij}{\uparrow}i \neq t_{ij}{\uparrow}i$ (otherwise we could not have $s_{ij} \not\models r_i$ and $t_{ij} \models r_i$), we conclude, by the pair-structure definition (5.2.1), that every maximal path starting in $s_{ij}$ must eventually contain a $P_i^j$-transition, because a $P_j^i$-transition cannot change any atomic proposition in $\mathcal{AP}_i$. Thus, every maximal $P_j^i$-path starting in $s_{ij}$ (if any) is finite and ends in a state which has no outgoing $P_j^i$-transitions.

We now demonstrate the existence of a reachable state $u_{ij}$ in $M_{ij}$ such that $u_{ij}{\uparrow}i = s_{ij}{\uparrow}i$ and such that $u_{ij}$ has no outgoing $P_j^i$-transitions. There are two cases.

*Case 1: There are no $P_j^i$-paths starting in $s_{ij}$.* Therefore $s_{ij}$ has no outgoing $P_j^i$-transitions, so let $u_{ij}$ be $s_{ij}$. (End of case 1.)

*Case 2: There is at least one $P_j^i$-path starting in $s_{ij}$.* Hence there is at least one maximal $P_j^i$-path starting in $s_{ij}$. By the above discussion, this path is finite and ends in a state with no outgoing $P_j^i$-transitions. Let $u_{ij}$ be this state. Since there is a $P_j^i$-path starting in $s_{ij}$ and ending in $u_{ij}$ we conclude that $u_{ij}{\uparrow}i = s_{ij}{\uparrow}i$, since, by the pair-structure definition (5.2.1) a $P_j^i$-transition cannot change the truth value assigned to any atomic proposition in $\mathcal{AP}_i$. Also, $u_{ij}$ is reachable, since $s_{ij}$ is reachable and there is a path from $s_{ij}$ to $u_{ij}$. (End of case 2.)

Since $u_{ij}$ has no outgoing $P^i_j$-transitions, all of the moves in $P^i_j$ (we are assuming compact notation here) are disabled in state $u_{ij}$, so we have

$$u_{ij} \models \bigwedge a^i_j \in P^i_j \,.\, (\neg\{\!\!\{a^i_j.start\}\!\!\} \vee \neg a^i_j.guard), \tag{a}$$

since a move is disabled iff control is not at its start state or its guard evaluates to false. Since every local state of a process has at least one outgoing arc (Section 2), there exists at least one move $a^i_j$ in $P^i_j$ such that $u_{ij} \models \{\!\!\{a^i_j.start\}\!\!\}$. From this and (a), we have

$$u_{ij} \models \bigvee a^i_j \in P^i_j \,.\, (\{\!\!\{a^i_j.start\}\!\!\} \wedge \neg a^i_j.guard). \tag{b}$$

Finally, since $u_{ij}$ is reachable and $u_{ij}\!\uparrow\! i = s_{ij}\!\uparrow\! i$, we obtain from (b)

$$\bigvee s^0_{ij} \in S^0_{ij} \,.\, (\, M_{ij}, s^0_{ij} \models EF(\, \{\!\!\{s_{ij}\!\uparrow\! i\}\!\!\} \wedge (\bigvee a^i_j \in P^i_j \,.\, (\{\!\!\{a^i_j.start\}\!\!\} \wedge \neg a^i_j.guard))\,)\,).$$

By the sometimes-blocking state definition (6.7.3.1), we conclude that $s_{ij}\!\uparrow\! i$ is sometimes-blocking. □

LEMMA 6.7.4.1 (PROGRESS). *If*

(1) *the liveness assumption LV holds,*

(2) *for every reachable I-state s, $W_I(s)$ is supercycle-free, and*

(3) *v is a reachable I-state of $M_I$ such that $\bigwedge k \in I(\ell) \,.\, (M_{k\ell}, v\!\uparrow\! k\ell \models \neg r_\ell \wedge AFr_\ell)$ for some $\ell \in dom(I)$ and $r_\ell \in \mathcal{L}(\mathcal{AP}_\ell, \neg, \wedge)$, then*

$$M_I, v \models_\Phi AFex_\ell.$$

PROOF. We prove $\pi \models Fex_\ell$ for $\pi$ an arbitrary fullpath in $M_I$ such that $v$ is the first state of $\pi$ and $\pi \models \Phi$. By the definition of $\models_\Phi$, this establishes $M_I, v \models_\Phi AFex_\ell$. Now $W_I(s)$ is supercycle-free for every reachable $I$-state $s$, by assumption. Hence, by the deadlock freedom theorem (6.5.5.2), we have $M_I, S^0_I \models AGEXtrue$. Therefore, $\pi$ is infinite. Also, by propositional and temporal logic reasoning (and using $nblk_i \equiv \neg blk_i$)

$$\pi \models \bigwedge i \in dom(I) \,.\, (\overset{\infty}{G} nblk_i \vee (\overset{\infty}{F} nblk_i \wedge \overset{\infty}{F} blk_i) \vee \overset{\infty}{G} blk_i).$$

Hence we can partition $dom(I)$ into three sets $\psi_{nblk}, \psi_{ex}, \psi_{blk}$ such that

$$\pi \models \bigwedge i \in \psi_{nblk} \,.\, \overset{\infty}{G} nblk_i$$
$$\pi \models \bigwedge i \in \psi_{ex} \,.\, (\overset{\infty}{F} nblk_i \wedge \overset{\infty}{F} blk_i)$$
$$\pi \models \bigwedge i \in \psi_{blk} \,.\, \overset{\infty}{G} blk_i.$$

So, by CTL* semantics, $\pi$ has a suffix $\rho$ such that

$$\rho \models \bigwedge i \in \psi_{nblk} \,.\, Gnblk_i \tag{a}$$
$$\rho \models \bigwedge i \in \psi_{ex} \,.\, (\overset{\infty}{F} nblk_i \wedge \overset{\infty}{F} blk_i) \tag{b}$$
$$\rho \models \bigwedge i \in \psi_{blk} \,.\, Gblk_i. \tag{c}$$

We now have three cases, depending on which of $\psi_{nblk}, \psi_{ex}, \psi_{blk}$ contains $\ell$.

*Case 1:* $\ell \in \psi_{nblk}$. Since $\bigwedge k \in I(\ell) \,.\, (M_{k\ell}, v\!\uparrow\! k\ell \models \neg r_\ell \wedge AFr_\ell)$ by assumption, $v\!\uparrow\!\ell$ is sometimes-blocking by the sometimes-blocking proposition (6.7.3.4). Thus

$v \models blk_\ell$. Since $\rho \models nblk_\ell$, $P_\ell$ must have changed state along $\pi$, because $v$ is the first state of $\pi$, and $\rho$ is a suffix of $\pi$ (remember that $blk_\ell$ is a purely propositional formula). By the $I$-structure definition (5.3.1), $P_\ell$ must have been executed along $\pi$. Hence $\pi \models Fex_\ell$. (End of case 1.)

*Case 2:* $\ell \in \psi_{ex}$. Since $\pi \models \overset{\infty}{F} nblk_\ell \wedge \overset{\infty}{F} blk_\ell$, $P_\ell$ must change state infinitely often along $\pi$. Hence by the $I$-structure definition (5.3.1), $P_\ell$ must be executed infinitely often along $\pi$. Thus $\pi \models \overset{\infty}{F} ex_\ell$, which implies $\pi \models Fex_\ell$. (End of case 2.)

*Case 3:* $\ell \in \psi_{blk}$. First, a few definitions are needed. A subset $\psi$ of $dom(I)$ is *$I$-connected* if and only if $\bigwedge i, j \in \psi, i \neq j \,.\, (i\, I^+\, j)$ where $I^+$ is the transitive closure of $I$. An $I$-process $P_k$ *borders* $\psi$ if and only if $k \in I(i) - \psi$ for some $i \in \psi$. We let $border(\psi)$ denote the set of all bordering $I$-processes of $\psi$. Let $\eta$ be a maximal $I$-connected subset of $\psi_{blk}$. We call $\eta$ a *blk*-region. Consider the $I$-processes that border $\eta$. Clearly, no such $I$-process can be in $\psi_{blk}$, as $\eta$ would not be maximal. Hence, every bordering $I$-process must be in $\psi_{nblk}$ or in $\psi_{ex}$. It is clear that $\psi_{blk}$ can be partitioned into ($I$-disconnected) *blk*-regions. Let $\theta$ be a maximal $I$-connected subset of $dom(I)$ such that $\pi \models \bigwedge i \in \theta \,.\, (\overset{\infty}{F} ex_i)$. We call $\theta$ an *inf*-region. For an $I$-process $P_i$ such that $\pi \models \overset{\infty}{F} ex_i$, we define $inf(i)$ to be the *inf*-region which $P_i$ is a member of. For $\psi$ an arbitrary subset of $dom(I)$, if wait-for-graph $W_I(s)$ contains an edge $a_i^I \longrightarrow P_j^I$ such that $i \in \psi$ and $j \notin \psi$, then we say that $W_I(s)$ contains an edge *out of* $\psi$.

We now establish a series of assertions P2, P3, P4, which together allow us to establish $\pi \models Fex_\ell$.

> Let $P_j$ border some *inf*-region $\theta$. Then there exists a suffix $\rho'$ of $\rho$ such that, for every state $s$ along $\rho'$,
> $W_I(s)$ contains no edges into $P_j$. (P2)

Proof of P2. Since $P_j$ borders *inf*-region $\theta$, there exists $k \in \theta$ such that $j\, I\, k$. Since $j \notin \theta$, we have $\rho \models \overset{\infty}{G} \neg ex_j$, as otherwise $\theta$ would not be maximal. Thus there exists a suffix $\rho''$ of $\rho$ such that

$$\rho'' \models (\overset{\infty}{F} ex_k \wedge G \neg ex_j).$$

Now consider $\rho'' {\uparrow} jk$. By the path projection definition (5.1.2.1) and $\rho'' \models (\overset{\infty}{F} ex_k \wedge G \neg ex_j)$, we have $\rho'' {\uparrow} jk \models (\overset{\infty}{F} ex_k \wedge G \neg ex_j)$. Thus, by the path-mapping lemma (6.4.3), $\rho'' {\uparrow} jk$ is a fullpath in $M_{jk}$. Also, $\rho'' {\uparrow} jk \models Gex_k$, since all transitions of $\rho'' {\uparrow} jk$ are either $P_k^j$-transitions or $P_j^k$-transitions. Hence

$$M_{jk}, s{\uparrow} jk \models EGex_k, \qquad \text{(a)}$$

where $s$ is an arbitrary state along $\rho''$. Note that $s$ is a reachable state in $M_I$, since it is reachable from $v$, and $v$ is, by assumption, a reachable state in $M_I$. Now let $i$ be an arbitrary element of $I(j)$, and let $J = \{\{i, j\}, \{j, k\}\}$. By the state-mapping corollary (6.4.5), $s{\uparrow} J$ is reachable in $M_J$. Since $(s{\uparrow} J){\uparrow} jk = s{\uparrow} jk$, we have, by (a)

$$M_{jk}, (s{\uparrow} J){\uparrow} jk \models EGex_k. \qquad \text{(b)}$$

Letting $s_J = s{\uparrow}J$ in $LV$ (Definition 6.7.2.1), we get from (b)

$$\bigwedge a_i^J \,.\, (a_i^J \longrightarrow P_j^J \notin W_J(s{\uparrow}J)), \tag{c}$$

so there is no edge in $W_J(s{\uparrow}J)$ from $i$ to $j$. Since $i\,I\,j$ and $j\,I\,k$, we have $J \subseteq I$. So, by applying the wait-for-graph projection proposition (6.5.4.1) to (c), we obtain $\bigwedge a_i^I \,.\, (a_i^I \longrightarrow P_j^I \notin W_I(s))$. Since $i$ is an arbitrarily chosen element of $I(j)$, we conclude that $W_I(s)$ contains no edges into $P_j^I$. As $s$ is an arbitrary state along $\rho''$, setting $\rho'$ to $\rho''$ concludes the proof of P2. (End proof of P2.)

Let $\psi$ be an arbitrary subset of $\psi_{blk}$. If there exists a suffix $\rho'$ of $\rho$ such that,

for every state $s$ along $\rho'$, $W_I(s)$ contains no edge out of $\psi$,

then,

$$\rho \models \bigvee i \in \psi \,.\, \overset{\infty}{F} ex_i. \tag{P3}$$

Proof of P3. Assume otherwise. Thus there exists a suffix $\rho''$ of $\rho$ such that $\rho'' \models \bigwedge i \in \psi \,.\, G\neg ex_i$. Let $s$ be an arbitrary state along $\rho''$. By assumption, $W_I(s)$ is supercycle-free. Hence $W_I(s){\uparrow}\psi$ is supercycle-free. Thus, by the supercycle proposition (6.5.5.1) with $I := I{\uparrow}\psi$, some move node $a_{i_s}^I$ such that $i_s \in \psi$, must have no outgoing edges in $W_I(s){\uparrow}\psi$. By assumption, $W_I(s)$ contains no edge from a move in $\psi$ to an $I$-process outside $\psi$. Thus $a_{i_s}^I$ must have no outgoing edges in $W_I(s)$. Hence, by Observation 6.5.2.2, $s(a_{i_s}^I.guard) = true$. So, by the compact $I$-structure definition (5.4.3), $a_{i_s}^I$ can be executed, and thus $M_I, s \models en_{i_s}$.

Now let $t$ be the successor state to $s$ along $\rho''$. By assumption, $W_I(t)$ contains no edge out of $\psi$. Also, $W_I(t){\uparrow}\psi = W_I(s){\uparrow}\psi$ since no $I$-process in $\psi$ is executed along $\rho''$. Thus $W_I(t)$ contains no edges out of $a_{i_s}^I$. Hence, by Observation 6.5.2.2, $t(a_{i_s}^I.guard) = true$. We can inductively repeat this argument (e.g., for the successor state to $t$, and then the successor state to that state, etc…) to conclude that $u(a_{i_s}^I.guard) = true$ for every state $u$ which occurs after $s$ along $\rho''$. Thus $\rho'' \models \overset{\infty}{G} en_{i_s}$.

Now $\rho'' \models \overset{\infty}{G} blk_{i_s}$, since $i_s \in \psi$ and $\psi \subseteq \psi_{blk}$. Since $\rho'' \models G\neg ex_{i_s}$, we have $\rho'' \models \overset{\infty}{G}(blk_{i_s} \wedge en_{i_s}) \wedge G\neg ex_{i_s}$. Hence, by the weak blocking fairness definition (6.7.3.3), $\rho'' \models \neg\Phi$. Hence $\pi \models \neg\Phi$, since $\rho''$ is a suffix of $\pi$. But $\pi$ was chosen so that $\pi \models \Phi$. Hence the original assumption is false, and P3 is established. (End proof of P3.)

Let $j$ be an arbitrary element of $\psi_{blk}$. Then either

$P_j$ is a member of some *inf*-region, or

$P_j$ borders some *inf*-region. (P4)

Proof of P4. Assume otherwise. Since $j \in \psi_{blk}$, we have $j \in \eta$ for some *blk*-region $\eta$. Hence

$$\zeta = \eta - (\textstyle\bigcup_{\theta \text{ is an } \mathit{inf}\text{-region}} (\theta \cup \mathit{border}(\theta)))$$

is nonempty, since $j \notin \theta \cup \mathit{border}(\theta)$ for any *inf*-region $\theta$ by assumption, and thus $j \in \zeta$. By (P2), we have that there exists a suffix $\rho'$ of $\rho$ such that

for any $I$-process $P_k$ which borders an *inf*-region,
for every state $s$ along $\rho'$,
$W_I(s)$ contains no edges into $P_k$. (a)

Also, if for a state $s$ and an $I$-process $P_k$, $s \models nblk_k$, then $W_I(s)$ contains no edge into $P_k$ (see Observation 6.7.3.2). Thus, by definition of $\psi_{nblk}$ we have

for every state $s$ along $\rho'$,
$W_I(s)$ contains no edge into any $I$-process $P_k$ in $\psi_{nblk}$. (b)

Now consider an arbitrary member $P_k$ of *border*$(\zeta)$. Since every $I$-process is a member of exactly one of $\psi_{blk}$, $\psi_{ex}$, $\psi_{nblk}$, we have three (sub-)cases.

$k \in \psi_{blk}$. Since $\zeta \subseteq \eta$, and $k \in$ *border*$(\zeta)$, we have $k \in \eta$, since $\eta$ is a *blk*-region, (and $k \in \psi_{blk}$ by the case assumption) and all *blk*-regions are maximal, by definition. By assumption, $k \in$ *border*$(\zeta)$, so $k \notin \zeta$ by definition of *border*. Therefore, $k \in \eta - \zeta$. Since $\zeta = \eta - (\bigcup_{\theta \text{ is an } inf\text{-region}} (\theta \cup border(\theta)))$ by definition, we have $k \in \theta \cup border(\theta)$ for some *inf*-region $\theta$. Since $k \in$ *border*$(\zeta)$, there exists $P_m$ such that $k\, I\, m$ and $m \in \zeta$. However, if $k \in \theta$, then $m \in$ *border*$(\theta)$, contrary to the definition of $\zeta$. Thus we conclude that $k \in$ *border*$(\theta)$.

$k \in \psi_{ex}$. Therefore $\pi \models \overset{\infty}{F} ex_k$ (see case 2). So $k$ is a member of some *inf*-region $\theta$, by definition of *inf*-region. Since $k \in$ *border*$(\zeta)$, there exists $m$ such that $k\, I\, m$ and $m \in \zeta$. Thus $m \in$ *border*$(\theta)$, contrary to the definition of $\zeta$. We conclude that $k$ cannot be a member of any *inf*-region, and hence $k$ cannot be a member of $\psi_{ex}$.

$k \in \psi_{nblk}$. We do not need to infer anything for this case other than $k \in \psi_{nblk}$.

Considering the above three (sub-)cases, we have shown that every member of *border*$(\zeta)$ either borders some *inf*-region or is a member of $\psi_{nblk}$. Therefore, in $W_I(s)$, every edge out of $\zeta$ must have a target $P_k$ such that $P_k$ borders an *inf*-region or such that $P_k$ is in $\psi_{nblk}$. Thus, by (a) and (b), we conclude that

for every state $s$ along $\rho'$, $W_I(s)$ contains no edge out of $\zeta$.

Since $\zeta \subseteq \eta$, $\zeta$ is a subset of $\psi_{blk}$ by definition of *blk*-region. Thus, by (P3), $\rho' \models \overset{\infty}{F} ex_{k'}$ for some $P_{k'}$ in $\zeta$. Hence $k' \in \theta$ for some *inf*-region $\theta$. But this contradicts $k' \in \zeta$, by definition of $\zeta$. Hence the original assumption is false, and P4 is established. (End proof of P4.)

We can now establish $\pi \models F ex_\ell$. By the case 3 assumption, $\ell \in \psi_{blk}$. By (P4), we have

$P_\ell$ is a member of some *inf*-region $\theta$ or $P_\ell$ borders some *inf*-region $\theta$.

If $P_\ell$ is a member of $\theta$, then $\pi \models \overset{\infty}{F} ex_\ell$, and therefore $\pi \models F ex_\ell$. If $P_\ell$ borders $\theta$, then there exists $j \in I(\ell) \cap \theta$. Hence $\pi \models \overset{\infty}{F} ex_j$, and since $\pi$ is an infinite fullpath, we conclude by the path projection definition (5.1.2.1) that $\pi{\uparrow}\ell j$ is an infinite path (and therefore is a fullpath) and $\pi{\uparrow}\ell j \models \overset{\infty}{F} ex_j$. Moreover, the first state of $\pi{\uparrow}\ell j$ is $v{\uparrow}\ell j$. Since $\bigwedge k \in I(\ell) \,.\, (M_{k\ell}, v{\uparrow}k\ell \models \neg r_\ell \wedge AF r_\ell)$ by assumption (3) of the lemma, and $j \in I(\ell)$, we have $M_{\ell j}, v{\uparrow}\ell j \models \neg r_\ell \wedge AF r_\ell$. Thus, $M_{\ell j}, v{\uparrow}\ell j \models AF ex_\ell$, and so

$\pi{\uparrow}\ell j \models Fex_\ell$ since $\pi{\uparrow}\ell j$ is a fullpath and the first state of $\pi{\uparrow}\ell j$ is $v{\uparrow}\ell j$. Hence, by the path projection definition (5.1.2.1), we conclude $\pi \models Fex_\ell$. (End of case 3.)

Since we have established $\pi \models Fex_\ell$ in all three cases, the lemma is established. □

## C.6 The Generalized Large Model Theorem

THEOREM 6.8.1 (GENERALIZED LARGE MODEL). *Let $f_{k\ell}$ be an arbitrary formula of $FLCTL_{k\ell}$ (Definition 6.6.1). Let $s$ be an arbitrary reachable $I$-state, and, for all $i, j$ such that $i\,I\,j$, let $s_{ij} = s{\uparrow}ij$. If the liveness assumption LV holds, and $W_I(u)$ is supercycle-free for every reachable $I$-state $u$, then*

$$\bigwedge(i,j) \in I\,.\,(M_{ij}, s_{ij} \models \textstyle\bigwedge_{k\ell} f_{k\ell}) \textit{ implies } M_I, s \models_\Phi \textstyle\bigwedge_{k\ell} f_{k\ell}.$$

PROOF. By MPCTL* semantics, $M_{ij}, s_{ij} \models \bigwedge_{k\ell} f_{k\ell}$ is equivalent to $M_{ij}, s_{ij} \models f_{ij}$. Also by MPCTL* semantics, $M_I, s \models_\Phi \bigwedge_{k\ell} f_{k\ell}$ is equivalent to $M_I, s \models_\Phi \bigwedge(i,j) \in I\,.\,(f_{ij})$. This is equivalent, by CTL* semantics, to $\bigwedge(i,j) \in I\,.\,(M_I, s \models_\Phi f_{ij})$. Hence, the generalized large-model theorem is established if we can prove that

$$\bigwedge(i,j) \in I\,.\,(M_{ij}, s_{ij} \models f_{ij}) \text{ implies } \bigwedge(i,j) \in I\,.\,(M_I, s \models_\Phi f_{ij}) \qquad (*)$$

given the assumptions of the generalized large-model theorem. The proof is by induction on the structure of $f_{ij}$.

The proofs of the cases $f_{ij} = h_{ij}$, $f_{ij} = f'_{ij} \wedge f''_{ij}$, $f_{ij} = AGh_{ij}$, $f_{ij} = AG(p_i \Rightarrow AY_iq_i)$, $f_{ij} = AG(a_i \Rightarrow EX_ib_i)$, are verbatim identical to the proofs for the same cases respectively in the large-model theorem (6.6.2), and are thereby omitted here. Note that only the proof for the remaining case of $f_{ij} = AG(p_i \Rightarrow A[q_iUr_i])$ in Theorem 6.6.2 appealed to the assumptions $I = \{i_1, \ldots, i_K\} \times \{i_1, \ldots, i_K\} - \{(i,i) \mid i \in \{i_1, \ldots, i_K\}\}$ and $M_I, S_I^0 \models AGEXtrue$ in the antecedent of Theorem 6.6.2. We now give the proof for $f_{ij} = AG(p_i \Rightarrow A[q_iUr_i])$ in the generalized case.

$f_{ij} = AG(p_i \Rightarrow A[q_iUr_i])$. We will establish $M_I, t \models_\Phi (p_i \Rightarrow A[q_iUr_i])$ where $t$ is an arbitrary $s$-reachable state in $M_I$. If $M_I, t \models \neg p_i$ then $M_I, t \models_\Phi (p_i \Rightarrow A[q_iUr_i])$, and we are done. Otherwise $M_I, t \models p_i$, and we must establish $M_I, t \models_\Phi A[q_iUr_i]$. If $M_I, t \models r_i$ then we are done. Otherwise $M_I, t \models \neg r_i$, so $t{\uparrow}ij \models \neg r_i$ by the $I$-state projection proposition (6.3.1). Let $\pi$ be an arbitrary fullpath of $M_I$ starting in $t$ such that $\pi \models \Phi$. The antecedent is

$$M_{ij}, s_{ij} \models AG(p_i \Rightarrow A[q_iUr_i]). \qquad \text{(a)}$$

By assumption, $s{\uparrow}ij = s_{ij}$. Thus by the relativized state-mapping corollary (6.4.6) with $J = \{\{i,j\}\}$, $t{\uparrow}ij$ is a $s_{ij}$-reachable state in $M_{ij}$. So, by (a) we have

$$M_{ij}, t{\uparrow}ij \models (p_i \Rightarrow A[q_iUr_i]). \qquad \text{(b)}$$

Since $M_I, t \models p_i$, we conclude $M_{ij}, t{\uparrow}ij \models p_i$ by the $I$-state projection proposition (6.3.1). Together with (b), this yields

$$M_{ij}, t{\uparrow}ij \models A[q_iUr_i], \qquad \text{(c)}$$

and since $t{\uparrow}ij \models \neg r_i$, we have by (c) and CTL* semantics

$$M_{ij}, t{\uparrow}ij \models \neg r_i \wedge AFr_i. \qquad \text{(d)}$$

By the path-mapping lemma (6.4.3), $\pi{\uparrow}ij$ is a path in $M_{ij}$. Also, $\pi{\uparrow}ij$ starts in $t{\uparrow}ij$, and therefore, by (c) and CTL* semantics, we have $M_{ij}, \pi{\uparrow}ij \models [q_i U_w r_i]$ (we cannot conclude $M_{ij}, \pi{\uparrow}ij \models [q_i U r_i]$ because we have not shown that $\pi{\uparrow}ij$ is a fullpath). By the path projection lemma (6.3.3) with $J = \{\{i,j\}\}$, we have $M_I, \pi \models [q_i U_w r_i]$. It remains for us to establish $M_I, \pi \models F r_i$.

Since $j$ ranges over $I(i)$, we have, from (d), $\bigwedge j \in I(i) \,.\, (M_{ij}, t{\uparrow}ij \models \neg r_i \wedge AF r_i)$. By assumption, $s$ is reachable, and $t$ is $s$-reachable, so $t$ is reachable. Together with the assumptions of the theorem, we have satisfied the antecedent of the progress lemma (6.7.4.1) for $v = t$ and $\ell = i$. Thus, by the progress lemma (6.7.4.1), we have $M_I, t \models_\Phi AF ex_i$. Therefore, $P_i^I$ is eventually executed along $\pi$. By repeating this argument inductively, we conclude that either $P_i^I$ is executed repeatedly along $\pi$ until a global state $t'$ (along $\pi$) is reached such that $M_I, t' \models r_i$, (and hence $M_I, \pi \models F r_i$), or $P_i^I$ is executed infinitely often along $\pi$ (since $\neg r_i \wedge AF r_i$ continues to hold, and therefore the progress lemma can be applied repeatedly). In the latter case, we have that $\pi$ contains an infinite number of $P_i^I$-transitions, so by definition of path projection, $\pi{\uparrow}ij$ contains an infinite number of $P_i^j$-transitions, so $\pi{\uparrow}ij$ is a fullpath. Since $\pi{\uparrow}ij$ is a fullpath starting in $t{\uparrow}ij$, we have, by (c), $M_{ij}, \pi{\uparrow}ij \models [q_i U r_i]$. By the path projection lemma (6.3.3), we have $M_I, \pi \models [q_i U r_i]$, which implies $M_I, \pi \models F r_i$.

Since $M_I, \pi \models F r_i$ in both cases, and $M_I, \pi \models [q_i U_w r_i]$ has been established above, we have $M_I, \pi \models [q_i U r_i]$. Since $\pi$ is an arbitrary fullpath starting in $t$ such that $\pi \models \Phi$, we conclude $M_I, t \models_\Phi A[q_i U r_i]$. Hence $M_I, t \models_\Phi (p_i \Rightarrow A[q_i U r_i])$. Since $t$ is an arbitrary $s$-reachable state in $M_I$, we conclude $M_I, s \models_\Phi AG(p_i \Rightarrow A[q_i U r_i])$. Since $i$ ranges over $\{i_1, \ldots, i_K\}$, and $j$ ranges over $I(i)$, we have $\bigwedge (i,j) \in I \,.\, (M_I, s \models_\Phi AG(p_i \Rightarrow A[q_i U r_i]))$. Thus, we have established (*), which concludes the proof of the generalized large-model theorem. □

## D. A COMPACT REPRESENTATION FOR SYNCHRONIZATION SKELETONS

In this appendix, we provide a full discussion of the compact representation introduced in Section 5.4. Our discussion here is self-contained and so repeats some of the material in Section 5.4 (in particular, the definition of compact MP-synthesis is repeated, and so retains the number, 5.4.1, that it has in the main text).

Suppose $P_i^j$ (for every $j \in I(i)$) contains two arcs from $i$-state $s_i$ to $i$-state $s_i'$, e.g., $a_{i,1}^j = (s_i, B_{i,1}^j \rightarrow A_{i,1}^j, s_i')$ and $a_{i,2}^j = (s_i, B_{i,2}^j \rightarrow A_{i,2}^j, s_i')$. Then, by the MP-synthesis definition (5.1.1), $P_i^I$ contains $2^{|I(i)|}$ arcs from $i$-state $s_i$ to $i$-state $s_i'$, one arc for each element of the cartesian product

$$\{a_{i,1}^{j_1}, a_{i,2}^{j_1}\} \times \cdots \times \{a_{i,1}^{j_n}, a_{i,2}^{j_n}\}$$

where $\{j_1, \ldots, j_n\} = I(i)$. Thus $P_i^I$ is exponentially large in $K$ $(= |dom(I)|)$ in the worst case (since $|I(i)| = K - 1$ when $i\, I\, j$ for every $j$ in $dom(I) - \{i\}$), which defeats the purpose of MP-synthesis. We deal with this by defining a compact representation for processes in which there is at most one arc between any pair of (local) states, thereby avoiding the exponential blowup illustrated above.

Consider a pair-process $P_i^j$ which has two arcs from state $s_i$ to state $s_i'$, labeled with the synchronization commands $B_{i,1}^j \rightarrow A_{i,1}^j$, $B_{i,2}^j \rightarrow A_{i,2}^j$. In compact notation, we replace these two arcs by a single arc whose label is $B_{i,1}^j \rightarrow A_{i,1}^j \oplus B_{i,2}^j \rightarrow A_{i,2}^j$.

The symbol $\oplus$ is a binary operator which takes a pair of guarded commands as arguments. It is defined as follows:

$$(B_1 \to A_1) \oplus (B_2 \to A_2) \stackrel{\text{df}}{=} B_1 \ \vee \ B_2 \to \begin{array}{l} \textbf{if } B_1 \to A_1 \\ \Box\ B_2 \to A_2 \\ \textbf{fi} \end{array}$$

Roughly, the operational semantics of $B^j_{i,1} \to A^j_{i,1} \oplus B^j_{i,2} \to A^j_{i,2}$ is that if one of the guards $B^j_{i,1}, B^j_{i,2}$ evaluates to true, then the corresponding body $A^j_{i,1}, A^j_{i,2}$, respectively, can be executed. If neither $B^j_{i,1}$ nor $B^j_{i,2}$ evaluates to true, then the command "blocks," i.e., waits until one of $B^j_{i,1}, B^j_{i,2}$ evaluates to true. Note that the guarded commands which we use as arc labels are, in general, *partial commands*, i.e., they cannot be executed in every global state. A full treatment of the calculus of partial guarded commands is beyond our scope here. The reader is referred to Nelson [1989]. It is easily seen that $\oplus$ is commutative. To see that it is also associative, we note that

$$(B_1 \to A_1 \oplus B_2 \to A_2) \oplus B_3 \to A_3$$

and

$$B_1 \to A_1 \oplus (B_2 \to A_2 \oplus B_3 \to A_3)$$

have the same semantics, namely, if one of $B_1, B_2, B_3$ is true, then the corresponding body can be executed, and if none of $B_1, B_2, B_3$ are true, then the command blocks. Since $\oplus$ is commutative and associative, it can be extended to $n$ arguments using the indexed notation $\oplus_{\ell \in [1:n]}$. Thus, if $P^j_i$ contains $n$ arcs from $i$-state $s_i$ to $i$-state $s'_i$, with labels $B^j_{i,1} \to A^j_{i,1}, \ldots, B^j_{i,n} \to A^j_{i,n}$, then these $n$ arcs can be replaced by a single arc whose label is $\oplus_{\ell \in [1:n]} B^j_{i,\ell} \to A^j_{i,\ell}$.

We call an arc whose label has the form $\oplus_{\ell \in [1:n]} B^j_{i,\ell} \to A^j_{i,\ell}$ a *(pair) move*. In compact notation, a pair-process has at most one move between any pair of local states. The translation from compact notation back to normal notation is straightforward: simply replace every move by the corresponding set of arcs. The operational semantics is as follows. Assume that the current state is $(s_1, \ldots, s_i, \ldots, s_K, x_1, \ldots, x_m)$, and that $P_i$ contains a move from $s_i$ to $s'_i$ labeled by $\oplus_{\ell \in [1:n]} B^j_{i,\ell} \to A^j_{i,\ell}$. If a guard $B^j_{i,\ell}$ (for some $\ell$ in $[1:n]$) evaluates to true in the current state, then $(s_1, \ldots, s'_i, \ldots, s_K, x'_1, \ldots, x'_m)$ is a permissible next-state where $x'_1, \ldots, x'_m$ is a list of updated shared variables resulting from action $A^j_{i,\ell}$. A computation path is an infinite sequence of states where successive pairs of states are related by the above next-state relation. As before, omission of the guard $B^j_{i,\ell}$ from a guarded command means that $B^j_{i,\ell}$ is interpreted as *true*, and we write the command as $A^j_{i,\ell}$, while omission of the action $A^j_{i,\ell}$ from a guarded command means that the shared variables are unaltered, and we write the command as $B^j_{i,\ell}$. This is formalized by the following definition, which is a consequence of the pair structure definition (5.2.1) and the translation between compact and normal notation given above.

*Definition* 5.4.2 (*Compact Pair-Structure*). Let $i\,I\,j$. The semantics of $(S^0_{ij}, P^j_i \| P^i_j)$ in compact notation is given by the pair-structure $M_{ij} = (S^0_{ij}, S_{ij}, R_{ij})$ where

(1) $S_{ij}$ is a set of $ij$-states,

(2) $S^0_{ij} \subseteq S_{ij}$ gives the initial states of $P^j_i \| P^i_j$, and

(3) $R_{ij} \subseteq S_{ij} \times \{i, j\} \times S_{ij}$ is a transition relation giving the transitions of $P^j_i \| P^i_j$. A transition $(s_{ij}, h, t_{ij})$ by $P^{\bar{h}}_h$ is in $R_{ij}$ if and only if
   (a) $h \in \{i, j\}$,
   (b) $s_{ij}$ and $t_{ij}$ are $ij$-states, and
   (c) there exists a move $(s_{ij}{\uparrow}h, \oplus_{\ell \in [1:n]} B^{\bar{h}}_{h,\ell} \to A^{\bar{h}}_{h,\ell}, t_{ij}{\uparrow}h)$ in $P^{\bar{h}}_h$ such that there exists $m \in [1:n]$:
      (i) $s_{ij}(B^{\bar{h}}_{h,m}) = true$,
      (ii) $< s_{ij}{\uparrow}\mathcal{SH}_{ij} > A^{\bar{h}}_{h,m} < t_{ij}{\uparrow}\mathcal{SH}_{ij} >$, and
      (iii) $s_{ij}{\uparrow}\bar{h} = t_{ij}{\uparrow}\bar{h}$.
   Here $\bar{h} = i$ if $h = j$ and $\bar{h} = j$ if $h = i$.

Now an $I$-process $P^I_i$ is derived by "composing" all the moves of $P^j_i$ (as $j$ varies over $I(i)$) which have the same start and end states. Toward this end, we define the binary composition operator $\otimes$ on guarded commands. We say that a guarded command is *simple* iff it has the form $B \to A$ where $B$ is a guard and $A$ is a parallel assignment statement. Applied to a pair of simple guarded commands, $\otimes$ returns a simple guarded command whose guard is the conjunction of the guards of the operands and whose assignment statement is the parallel composition of the assignment statements of the operands.

*Definition* D.1 (*Simple Guarded-Command Composition*). The composition of two simple guarded commands is given by

$$(B_1 \to A_1) \otimes (B_2 \to A_2) \;=\; B_1 \wedge B_2 \to A_1 // A_2.$$

That is, if both guards are true, then both bodies can be executed in parallel. We note that, in general, it is possible that $A_1, A_2$ have a variable in common on their left-hand sides. Since $\otimes$ is a *syntactic* operator, this presents no problems in the definition of $\otimes$. However, the semantics of $A_1 // A_2$ is problematic in general. But, the only time when we need to assign a semantics to guarded commands containing $\otimes$ is in Definition 5.4.3 below, and we shall see that in this case, the possibility of assigning twice to the same variable within a parallel assignment does not arise. It is clear that $\otimes$, when applied to simple guarded commands, is commutative and associative, since both $\wedge$ and $//$ are. We now define general guarded commands as those that can be built up from simple guarded commands by applying $\oplus$ and $\otimes$.

*Definition* D.2 (*General Guarded Command*). General guarded commands are inductively defined as follows.

(1) A simple guarded command is a general guarded command

(2) If $G_1, G_2$ are general guarded commands, then so are $G_1 \oplus G_2$ and $G_1 \otimes G_2$

(3) The only general guarded commands are those that are generated by rules (1) and (2) above

In order to define the operational semantics of a general guarded command, we introduce a *normal form* for general guarded commands.

*Definition* D.3 (*Normal Form*). A general guarded command is in normal form iff it has the form $\oplus_{\ell \in [1:n]} G_\ell$, where each $G_\ell$ is a simple guarded command.

We note that the operational semantics of normal forms has been described informally above. The compact pair-structure definition (5.4.2) formalizes the operational semantics of normal forms by defining the pair-structures that are generated by pair-processes expressed in compact notation (whose arc labels are general guarded commands expressed in normal form). We can now define the application of $\otimes$ to general guarded commands in normal form as follows.

*Definition* D.4 (*Normal Form Composition*). The composition of two general guarded commands in normal form is given by

$$(\oplus_{\ell \in \varphi} G_\ell) \ \otimes \ (\oplus_{k \in \psi} G_k) \quad = \quad \oplus_{\ell \in \varphi, k \in \psi}(G_\ell \otimes G_k).$$

We recall that $G_\ell \otimes G_k$ is given by the simple guarded-command composition definition (D.1), since $G_\ell, G_k$ are simple guarded commands.

It remains to show that every guarded command can be expressed in normal form.

PROPOSITION D.5 (NORMAL FORM). *Every general guarded command can be expressed in normal form.*

PROOF. By structural induction over the definition of a general guarded command $G$.

*Base Case*. $G$ is simple. Then $G = \oplus_{\ell \in \varphi} G_\ell$ where $\varphi$ is a singleton set. This is in normal form.

*Induction Step*. $G = G_1 \oplus G_2$. By the induction hypothesis, $G_1$ and $G_2$ are in normal form. Let $G_1 = \oplus_{\ell \in \varphi} G_\ell$, $G_2 = \oplus_{k \in \psi} G_k$. Hence $G_\ell$ for all $\ell \in \varphi$ and $G_k$ for all $k \in \psi$ are simple guarded commands, by the normal-form definition (D.3). Since $\oplus$ is associative and commutative over simple guarded commands, we have $G_1 \oplus G_2 = (\oplus_{\ell \in \varphi} G_\ell) \oplus (\oplus_{k \in \psi} G_k) = (\oplus_{m \in \varphi \cup \psi} G_m)$. This last expression is in normal form.

*Induction Step*. $G = G_1 \otimes G_2$. By the induction hypothesis, $G_1$ and $G_2$ are in normal form. Let $G_1 = \oplus_{\ell \in \varphi} G_\ell$, $G_2 = \oplus_{k \in \psi} G_k$. Hence, by the normal-form composition definition (D.4), $G_1 \otimes G_2 = \oplus_{\ell \in \varphi, k \in \psi}(G_\ell \otimes G_k)$. Now $G_\ell$ for all $\ell \in \varphi$ and $G_k$ for all $k \in \psi$ are simple guarded commands, by the normal-form definition (D.3). Thus, by the simple guarded-command composition definition (D.1), $G_\ell \otimes G_k$ can be rewritten as a simple guarded command. Thus by the normal-form definition (D.3), $G_1 \otimes G_2$ can be expressed in normal form. $\square$

We finally recast the MP-synthesis definition (5.1.1) into compact form as follows.

*Definition* 5.4.1 (*Compact MP-Synthesis*). A *compact $I$-process $P_i^I$* is derived from the compact pair-process $P_i^j$ as follows:

$P_i^I$ contains a move from $s_i$ to $t_i$ with label $\otimes_{j \in I(i)} \oplus_{\ell \in [1:n]} B_{i,\ell}^j \to A_{i,\ell}^j$

iff

for every $j$ in $I(i)$: $P_i^j$ contains a move from $s_i$ to $t_i$ with label $\oplus_{\ell \in [1:n]} B_{i,\ell}^j \to A_{i,\ell}^j$.

The *initial state set* $S_I^0$ of the $I$-system is derived from the initial state $S_{ij}^0$ of the pair-system as follows:

$$S_I^0 = \{s \mid \bigwedge (i,j) \in I \,.\, (s{\uparrow}ij \in S_{ij}^0)\}.$$

Thus an $I$-process in compact notation has at most one move of the form $(s_i, \otimes_{j \in I(i)} \oplus_{\ell \in [1:n]} B_{i,\ell}^j \to A_{i,\ell}^j, t_i)$ between any pair of $i$-states $s_i, t_i$. Note that a move in a pair-process is simply a special case of a move in an $I$-process when $I$ is a single pair, e.g., if $I = \{\{i,k\}\}$, then $I(i)$ in $\otimes_{j \in I(i)}$ expands to the singleton $\{k\}$ giving a move in the pair-process $P_i^k$. In the sequel, we shall use $a_i^j, a_i^I$ to denote moves in $P_i^j, P_i^I$, respectively.

Returning to the example given at the beginning of this section, the pair of arcs $(s_i, B_{i,1}^j \to A_{i,1}^j, s_i')$, $(s_i, B_{i,2}^j \to A_{i,2}^j, s_i')$ in $P_i^j$ are replaced by the single move $(s_i, (B_{i,1}^j \to A_{i,1}^j \oplus B_{i,2}^j \to A_{i,2}^j),\ s_i')$, and the $2^{|I(i)|}$ arcs in $P_i^I$ are replaced by the single move $(s_i,\ \otimes_{j \in I(i)}(B_{i,1}^j \to A_{i,1}^j \oplus B_{i,2}^j \to A_{i,2}^j),\ s_i')$.

The translation of $I$-processes from normal to compact notation is most easily achieved via the translation given above for pair-processes, i.e., derive the corresponding pair-process (essentially applying MP-synthesis "in reverse"), perform the translation to obtain the compact pair-processes, and then derive the compact $I$-process using the compact MP-synthesis definition (5.4.1). Translating from compact to normal notation is achieved by applying the definition of the $\otimes$ operator, in effect expanding the general guarded command $\otimes_{j \in I(i)} \oplus_{\ell \in [1:n]} B_\ell \to A_\ell$ into normal form, and then creating one arc for each simple guarded command in the normal form. For example, if $I = \{\{i,k\},\{k,\ell\},\{\ell,i\}\}$, then

$$\otimes_{j \in I(i)}(T_j \to x_{ij} := j \oplus N_j \ \vee\ C_j \to \mathit{skip})$$

expands into normal form as follows. First, we expand the bound variable $j$ over its range $\{k,\ell\}$, thereby replacing the indexed form $\otimes_{j \in I(i)}$ by the infix form $\otimes$, to obtain

$$\begin{array}{l}(T_k \to x_{ik} := k \oplus N_k \ \vee\ C_k \to \mathit{skip}) \ \otimes \\ (T_\ell \to x_{i\ell} := \ell \oplus N_\ell \ \vee\ C_\ell \to \mathit{skip}).\end{array}$$

Now we apply the normal-form composition definition (D.4) to obtain

$$\begin{array}{l}(T_k \to x_{ik} := k \otimes T_\ell \to x_{i\ell} := \ell) \ \oplus \\ (N_k \ \vee\ C_k \to \mathit{skip} \otimes T_\ell \to x_{i\ell} := \ell) \ \oplus \\ (T_k \to x_{ik} := k \otimes N_\ell \ \vee\ C_\ell \to \mathit{skip}) \ \oplus \\ (N_k \ \vee\ C_k \to \mathit{skip} \otimes N_\ell \ \vee\ C_\ell \to \mathit{skip}).\end{array}$$

Finally, we apply the simple guarded-command composition definition (D.1) to all four occurrences of $\otimes$, to obtain

$$T_k \wedge T_\ell \to x_{ik} := k // x_{i\ell} := \ell \ \oplus$$

$$\begin{array}{l}(N_k \ \vee \ C_k) \wedge T_\ell \rightarrow skip//x_{i\ell} := \ell \ \oplus \\ T_k \wedge (N_\ell \ \vee \ C_\ell) \rightarrow x_{ik} := k//skip \ \oplus \\ (N_k \ \vee \ C_k) \wedge (N_\ell \ \vee \ C_\ell) \rightarrow skip//skip.\end{array}$$

It is easy to see that if $P_i^I$ had three neighbors instead of two, that the size of the final result would be eight instead of four. Generalizing, we see that if $|I(i)| = m$, then

$$\otimes_{j \in I(i)} \oplus_{\ell \in [1:n]} B_{i,\ell}^j \rightarrow A_{i,\ell}^j$$

expands into a term with size on the order of $n^m$. Thus the compact form provides an exponential savings in the size of the representation of the $I$-processes.

The operational semantics of compact $I$-processes is given by the following definition, which is a consequence of the $I$-structure definition (5.3.1) and the translation between compact and normal notation given above.

*Definition* 5.4.3 (*Compact $I$-Structure*). The semantics of $(S_I^0, P_{i_1}^I \| \ldots \| P_{i_K}^I)$ in compact notation is given by the $I$-structure $M_I = (S_I^0, S_I, R_I)$ where

(1) $S_I$ is a set of $I$-states,

(2) $S_I^0 \subseteq S_I$ gives the initial states of $P_{i_1}^I \| \ldots \| P_{i_K}^I$, and

(3) $R_I \subseteq S_I \times dom(I) \times S_I$ is a transition relation giving the transitions of $P_{i_1}^I \| \ldots \| P_{i_K}^I$. A transition $(s, i, t)$ by $P_i^I$ is in $R_I$ if and only if
   - (a) $i \in dom(I)$,
   - (b) $s$ and $t$ are $I$-states, and
   - (c) there exists a move $(s{\uparrow}i, \otimes_{j \in I(i)} \oplus_{\ell \in [1:n]} B_{i,\ell}^j \rightarrow A_{i,\ell}^j, t{\uparrow}i)$ in $P_i^I$ such that all of the following hold:
     - (i) for all $j$ in $I(i)$, there exists $m \in [1:n]$:
       $s{\uparrow}ij(B_{i,m}^j) = true$ and $< s{\uparrow}\mathcal{SH}_{ij} > A_{i,m}^j < t{\uparrow}\mathcal{SH}_{ij} >$,
     - (ii) for all $j$ in $dom(I) - \{i\}$: $s{\uparrow}j = t{\uparrow}j$, and
     - (iii) for all $j, k$ in $dom(I) - \{i\}$, $j \, I \, k$: $s{\uparrow}\mathcal{SH}_{jk} = t{\uparrow}\mathcal{SH}_{jk}$.

Given a pair-system $(S_{ij}^0, P_i^j \| P_j^i)$ in normal notation, we can apply Definitions 5.2.1, 5.1.1, and 5.3.1 to generate the pair-structure, the $I$-system, and the $I$-structure (for the generated $I$-system) respectively which correspond to $(S_{ij}^0, P_i^j \| P_j^i)$. Alternatively, if $(S_{ij}^0, P_i^j \| P_j^i)$ is given in compact notation, then we can apply Definitions 5.4.2, 5.4.1, and 5.4.3 to generate the pair-structure, the $I$-system (in compact notation), and the $I$-structure (for the generated $I$-system), respectively which correspond to $(S_{ij}^0, P_i^j \| P_j^i)$. It should be apparent that the pair, $K$-process structures generated by one set of definitions are identical to those generated by the other set of definitions. Thus, any result established in the sequel using one set of definitions will be equally applicable to the other set of definitions. Hence, in establishing a particular result, we will use whichever set of definitions is more convenient at the time. In particular, although some results will be established using normal notation, the implementation of MP-synthesis will be in compact notation, so as to avoid the exponential size $I$-processes which the normal notation may produce.

## E. CHECKING THE WAIT-FOR-GRAPH ASSUMPTION AND THE LIVENESS ASSUMPTION

### E.1 Checking the Wait-for-Graph Assumption

The wait-for-graph assumption $WG$ is mechanically checked as follows. As stated in Section 5 we are initially given a pair-system $(S^0_{k\ell}, P^\ell_k \| P^k_\ell)$. For technical convenience, we use the process indices $k, \ell$ rather than $i, j$. Using the pair-structure definition (5.2.1) we generate $M^r_{k\ell}$ (recall that a superscript $r$ denotes reachable states/structures—see Section 6.1) from $(S^0_{k\ell}, P^\ell_k \| P^k_\ell)$. We also translate $P^\ell_k, P^k_\ell$ to compact notation as shown in Section 5.4. From $M^r_{k\ell}$, we determine the set $S^r_k$ of reachable $k$-states in $M^r_{k\ell}$.[14] For every $t_k \in S^r_k$, we determine $|t_k.moves|$, using the compact form of $P^\ell_k$. Now the wait-for-graph assumption (Definition 6.5.4.2) specifies that $J$ has the form $\{\{j,k\},\{k,\ell_1\},\ldots,\{k,\ell_n\}\}$ where $n = |t_k.moves|$ and $k \notin \{j,\ell_1,\ldots,\ell_n\}$. Since no constraint is placed on the equality of members of $\{j,\ell_1,\ldots,\ell_n\}$, we must consider all possible cases for the form of $J$. We therefore define:

$$
\begin{aligned}
\mathcal{J}(t_k) \;=\; & \{J \mid J = \{\{j,k\},\{k,\ell_1\},\ldots,\{k,\ell_m\}\} \text{ and} \\
& \quad m \in [1:n] \text{ and} \\
& \quad j,k,\ell_1,\ldots,\ell_m \text{ are pairwise distinct}\} \\
\mathcal{J}'(t_k) \;=\; & \{J \mid J = \{\{k,\ell_1\},\ldots,\{k,\ell_m\}\} \text{ and} \\
& \quad m \in [1:n] \text{ and} \\
& \quad k,\ell_1,\ldots,\ell_m \text{ are pairwise distinct}\}
\end{aligned}
$$

$\mathcal{J}(t_k) \cup \mathcal{J}'(t_k)$ is the set of all the distinct forms of $J$ that must be considered when checking $WG$ for the $k$-state $t_k$, with $\mathcal{J}(t_k)$ containing all the forms of $J$ in which $j \notin \{\ell_1,\ldots,\ell_m\}$, and $\mathcal{J}'(t_k)$ containing all the forms of $J$ in which $j \in \{\ell_1,\ldots,\ell_m\}$. Note that $m$ is really the number of distinct indices in $\{\ell_1,\ldots,\ell_n\}$, so it ranges over $[1:n]$. For every $J$ in $\mathcal{J}(t_k)$, we generate $M^r_J$, using the compact MP-synthesis definition (5.4.1) and the compact $I$-structure definition (5.4.3). Then, for every $J$-state $t_J$ such that $t_J{\uparrow}k = t_k$ and $s_J \xrightarrow{k} t_J \in R_J$ for some reachable $J$-state $s_J$ of $M_J$, we evaluate

$$\bigwedge a^J_j \,.\, (a^J_j \longrightarrow P^J_k \notin W_J(t_J))$$

or

$$\bigvee a^J_k \in W_J(t_J) \,.\, (\bigwedge \ell \in \{\ell_1,\ldots,\ell_m\} \,.\, (a^J_k \rightarrow P^J_\ell \notin W_J(t_J))).$$

Using the wait-for-graph definition (6.5.2.1) and CTL* semantics, we can rewrite this as follows:

$$
\begin{aligned}
t_J \models \;& ( \;\bigwedge a^J_j \,.\, (\{\!\!\{a^J_j.start\}\!\!\} \Rightarrow a^J_j.guard_k) \\
& \;\vee \\
& \quad \bigvee a^J_k \,.\, (\{\!\!\{a^J_k.start\}\!\!\} \wedge (\bigwedge \ell \in \{\ell_1,\ldots,\ell_m\} \,.\, a^J_k.guard_\ell)) \\
& ).
\end{aligned} \qquad \text{(a)}
$$

Since the formula on the right-hand side of the $\models$ in (a) is purely propositional,

[14] This step, in effect, enforces our assumption (made in Section 6.1) that every $i$-state $s_i$ of $P^j_i$ is reachable in $M_{ij}$.

it is straightforward to evaluate (a) using the inductive definition for $\models$ supplied in Section 1. If, for any $t_k$, $J$, and $t_J$, (a) evaluates to false, then $WG$ does not hold. Likewise, for every $J$ in $\mathcal{J}'(t_k)$, we generate $M^r_J$. Then, we set $j$ equal to an arbitrarily chosen member of $\{\ell_1, \ldots, \ell_m\}$. Since $J$ is "radially symmetric" with respect to the $\ell_1, \ldots, \ell_m$ (i.e., $J$ is a "star" with $k$ as the center and $\ell_1, \ldots, \ell_m$ as the points), the particular choice of member of $\{\ell_1, \ldots, \ell_m\}$ makes no difference to the final outcome of the check. As in the case of $J \in \mathcal{J}(t_k)$, we evaluate (a), and if, for any $t_k$, $J$ and $t_J$, (a) evaluates to false, then $WG$ does not hold.

If, during the execution of the procedure outlined above, no $t_k$, $J$, and $t_J$ are found for which (a) evaluates to false, then $WG$ holds, and we can thus conclude that all $I$-systems are deadlock-free (provided that $W_I(s^0_I)$ is supercycle-free for every $s^0_I \in S^0_I$). We summarize the procedure given above in Figure 16, and compute its time complexity. The procedure clearly terminates, since the range of all loop variables is finite. Furthermore, upon termination, $WG$ holds if and only if $WGflag$ is set to "true."

We use the notation $|S|$ to denote the number of bits needed to represent $S$ using a "straightforward" encoding scheme. By definition of $M^r_{k\ell}$, we have that $|M^r_{k\ell}|$ is $O(|S^0_{k\ell}| + |S^r_{k\ell}| + |R^r_{k\ell}|)$. Since the pair-system is assumed to be nonterminating (Section 6.5), $R^r_{k\ell}$ is total, so we have $|S^0_{k\ell}| \leq |R^r_{k\ell}|$ and $|S^r_{k\ell}| \leq |R^r_{k\ell}|$. Hence $|M^r_{k\ell}|$ is $O|R^r_{k\ell}|$. We consider each step of the procedure and compute its worst-case time complexity. Step 0 can be performed in constant time. Step 1 requires the construction of $R^r_{k\ell}$. We generate $R^r_{k\ell}$ incrementally by starting with $S^0_{k\ell}$, selecting some state $s_{k\ell} \in S^0_{k\ell}$, and "expanding" $s_{k\ell}$ by computing all the transitions with source state $s_{k\ell}$ using the compact pair structure definition (5.4.2). We then mark $s_{k\ell}$ as "expanded" and repeat the process until all reachable states have been marked. Each reachable state (and each reachable transition) need be inspected only once. The "expansion" of each state using Definition 5.2.1 requires access to $P^\ell_k$ and $P^k_\ell$. Hence the time complexity of step 1 is $O(|P^\ell_k| \cdot |R^r_{k\ell}|)$.

From the definition of the compact notation given in Section 5.4, we see that the complexity of step 2 (converting $P^\ell_k, P^k_\ell$ to compact notation) is $O(|P^\ell_k|)$.

Step 3 can be performed by a single traversal of $R^r_{k\ell}$, and so its complexity is $O(|R^r_{k\ell}|)$.

The complexity of step 4.1 is $O(n)$, as we must count all the moves in $t_k.moves$. The complexity of step 4.2 is simply $O(\sum_{J \in \mathcal{J}(t_k)} |J|)$. Since $|J| = m + 1$, this can be rewritten as $O(\sum_{1 \leq m \leq n} m)$, i.e., $O(n^2)$.

We evaluate the complexity of step 4.3 starting with the most deeply nested iterations and proceeding outward. Now (a) can be rewritten as

$$(t_J{\uparrow}jk \models \bigwedge a^J_j \,.\, (\{\!\!\{ a^J_j.start \}\!\!\} \Rightarrow a^J_j.guard_k))$$

or

$$\bigvee a^J_k \,.\, (\,(t_J{\uparrow}j \models \{\!\!\{ a^J_k.start \}\!\!\}) \text{ and } (\bigwedge \ell \in \{\ell_1, \ldots, \ell_m\} \,.\, (t_J{\uparrow}j\ell \models a^J_k.guard_\ell))\,).$$

Hence the complexity of evaluating (a) is $O(sz2 \cdot |P^\ell_k| + m \cdot sz2 \cdot |P^\ell_k|)$, where $sz2$ is the number of bits needed to represent a single $k\ell$-state. This is just $O(m \cdot sz2 \cdot |P^\ell_k|)$. Since step 4.3.2 requires the inspection of all states in $S^r_J$ (but *not* all transitions in $R^r_J$), the complexity of step 4.3.2 is $O(m \cdot |S^r_J| \cdot |P^\ell_k|)$. Step 4.3.1 is carried out in an analogous manner to step 1. Each reachable state (and each reachable transition) in $R^r_J$ is inspected only once. The "expansion" of each state (using the compact

0. $WGflag := true$;
1. generate $M^r_{k\ell}$ from $(S^0_{k\ell}, P^\ell_k \| \ P^k_\ell)$;
2. translate $P^\ell_k$, $P^k_\ell$ into compact notation;
3. $S^r_k := \{t_k \mid \bigvee s_{k\ell} \in S^r_{k\ell} \,.\, (s_{k\ell}{\uparrow}k = t_k)\}$;
4. for all $t_k$ in $S^r_k$
   - 4.1 $n := |t_k.moves|$;
   - 4.2 $\mathcal{J}(t_k) := \{J \mid J = \{\{j,k\},\{k,\ell_1\},\ldots,\{k,\ell_m\}\}$ and $m \in [1:n]$ and $j,k,\ell_1,\ldots,\ell_m$ are pairwise distinct $\}$;
   - 4.3 for all $J$ in $\mathcal{J}(t_k)$
     - 4.3.1 generate $M^r_J$;
     - 4.3.2 for all $t_J$ such that $t_J{\uparrow}k = t_k$ and $s_J \xrightarrow{k} t_J \in R^r_J$ for some $s_J$
       if (a) evaluates to false, then $WGflag := false$;
   - 4.4 $\mathcal{J}'(t_k) := \{J \mid J = \{\{k,\ell_1\},\ldots,\{k,\ell_m\}\}$ and $m \in [1:n]$ and $k,\ell_1,\ldots,\ell_m$ are pairwise distinct $\}$;
   - 4.5 for all $J$ in $\mathcal{J}'(t_k)$
     - 4.5.1 generate $M^r_J$;
     - 4.5.2 set $j$ equal to an arbitrarily selected member of $\{\ell_1,\ldots,\ell_m\}$;
     - 4.5.3 for all $t_J$ such that $t_J{\uparrow}k = t_k$ and $s_J \xrightarrow{k} t_J \in R^r_J$ for some $s_J$
       if (a) evaluates to false, then $WGflag := false$

Fig. 16. Procedure to check the wait-for-graph assumption.

$I$-structure definition (5.4.3) with $I := J$ now) requires access to $P^J_k$ and $P^J_\ell$ (for $\ell \in \{j,\ell_1,\ldots,\ell_m\}$). Thus the complexity of step 4.3.1 is $O((|P^J_k| + m \cdot |P^J_\ell|) \cdot |R^r_J|)$. In compact notation, $P^J_k$ is derived by applying the operator "$\otimes_{\ell \in J(k)}$" to the label of every move in $P^\ell_k$. Hence $|P^J_k|$ is $O(|J| \cdot |P^\ell_k|)$, which is $O(m \cdot |P^\ell_k|)$ since $|J|$ is $O(m)$. $|P^J_\ell|$ (for $\ell \in \{j,\ell_1,\ldots,\ell_m\}$) is $O(|P^k_\ell|)$, since $P^J_\ell$ has only one $J$-neighbor, namely $P^J_k$. Hence the complexity of step 4.3.1 can be rewritten as $O(m \cdot |P^\ell_k| \cdot |R^r_J|)$. So the complexity of steps 4.3.1 and 4.3.2 combined is $O(m \cdot |P^\ell_k| \cdot |R^r_J| + m \cdot |P^\ell_k| \cdot |S^r_J|)$. Since $|S^r_J| \le |R^r_J|$, the first summand is not less than the second, so this is $O(m \cdot |P^\ell_k| \cdot |R^r_J|)$.

Now $R^r_J$ can be regarded as a "product" of the $m+1$ transition sets $R_{k\ell}$ for $\ell \in \{j,\ell_1,\ldots,\ell_n\}$, as given by the transition-mapping lemma (6.4.1). Thus, $|R^r_J|$ is $O(|R^r_{k\ell}|^{m+1}$. However, this counts the process $P^J_k$ $m+1$ times instead of once, so we can improve this upper bound to $O(|R^r_{k\ell}|^{m+1}/|P^\ell_k|^m)$. We rewrite this as $O(|R^r_{k\ell}| \cdot \alpha^m)$, where $\alpha = |R^r_{k\ell}|/|P^\ell_k|$. So the complexity of steps 4.3.1 and 4.3.2 combined is rewritten as $O(m \cdot |P^\ell_k| \cdot |R^r_{k\ell}| \cdot \alpha^m)$.

Step 4.3 iterates steps 4.3.1 and 4.3.2 over all $J$ in $\mathcal{J}(t_k)$, so its complexity is $O(\sum_{J \in \mathcal{J}(t_k)} m \cdot |P^\ell_k| \cdot |R^r_{k\ell}| \cdot \alpha^m)$. Now $J$ determines $m$, since $m =$ (the number of pairs in $J$) $-1$. So as $J$ varies from $\{\{j,k\},\{k,\ell_1\}\}$ to $\{\{j,k\},\{k,\ell_1\},\ldots,\{k,\ell_n\}\}$, $m$ will vary from 1 to $n$. Thus the complexity of step 4.3 is $O(|P^\ell_k| \cdot |R^r_{k\ell}| \cdot \sum_{1 \le m \le n} m \cdot \alpha^m)$, since $|P^\ell_k|$ and $|R^r_{k\ell}|$ are independent of $J$. $\sum_{1 \le m \le n} m \cdot \alpha^m$ is bounded from above by $n \cdot \sum_{1 \le m \le n} \alpha^m$, and $\sum_{1 \le m \le n} \alpha^m = (\alpha^{n+1} - 1)/(\alpha - 1)$, which is approximately $O(\alpha^n)$ when $\alpha \gg 1$, as is usually the case. Hence the complexity of step 4.3 is $O(n \cdot |P^\ell_k| \cdot |R^r_{k\ell}| \cdot \alpha^n)$. Finally, the complexity of steps 4.4 and 4.5 is easily seen to be at most equal to the complexity of steps 4.2, 4.3 respectively, since the only difference is that $J$ is smaller by one pair. Thus, the overall complexity of the body of step 4 (i.e., steps 4.1–4.5) is $O(n + n^2 + (n \cdot |P^\ell_k| \cdot |R^r_{k\ell}| \cdot \alpha^n))$. Now $n$ is bounded by the maximum branching within a single pair-process, so $|P^\ell_k| \ge n$. Hence the above is $O(n \cdot |P^\ell_k| \cdot |R^r_{k\ell}| \cdot \alpha^n)$.

The body of step 4 is repeated for every $k$-state $t_k$ in $S^r_k$. Let $b$ denote the maximum value of $n$ $(= |t_k.moves|)$ as $t_k$ ranges over $S^r_k$. Thus the complexity of step 4 is $O(b \cdot |P^\ell_k| \cdot |R^r_{k\ell}| \cdot \alpha^b \cdot |S^r_k|)$. The complexities of steps 1, 2, and 3 computed above are $O(|P^\ell_k| \cdot |R^r_{k\ell}|)$, $O(|P^\ell_k|)$, and $O(|R^r_{k\ell}|)$, respectively. These are all subsumed by the complexity of step 4, which therefore gives the overall complexity of the procedure for checking $WG$. Now $|S^r_k|$ is $O(|P^\ell_k|)$, since $S^r_k$ is a subset of the $k$-states of $P^\ell_k$. Thus, the overall time complexity can be rewritten as $O(b \cdot |P^\ell_k|^2 \cdot |R^r_{k\ell}| \cdot \alpha^b)$. Replacing $\alpha$ by $|R^r_{k\ell}|/|P^\ell_k|$, we obtain $O(b \cdot |R^r_{k\ell}|^{b+1}/|P^\ell_k|^{b-2})$ for the worst-case time complexity of the procedure for checking the wait-for-graph assumption.

Now each $|R^r_{k\ell}|$ is $O(|P^\ell_k|^2 \cdot 2^{|\mathcal{SH}_{k\ell}|})$, since a pair-system contains two pair-processes (which contribute $|P^\ell_k|^2$ to the size of $R^r_{k\ell}$), and a set of pairwise shared variables (which contributes $2^{|\mathcal{SH}_{k\ell}|}$ to the size of $R^r_{k\ell}$). Hence, the overall time complexity can also be written as $O(b \cdot (|P^\ell_k|^{2b+2} \cdot 2^{(b+1)|\mathcal{SH}_{k\ell}|})/|P^\ell_k|^{b-2})$, that is, $O(b \cdot |P^\ell_k|^{b+4} \cdot 2^{(b+1)|\mathcal{SH}_{k\ell}|})$.

Also, the space complexity of the procedure is $O(|R^r_J|)$, since $R^r_J$ is the largest of a fixed set of data structures that are used. This is $O(|R^r_{k\ell}|^{b+1}/|P^\ell_k|^b)$, or, alternatively, $O(|P^\ell_k|^{b+2} \cdot 2^{(b+1)|\mathcal{SH}_{k\ell}|})$.

### E.2 Checking the Liveness Assumption

The liveness assumption (6.7.2.1) is mechanically checked as follows. We introduce a "new"[15] atomic proposition $Q$ to $M_J$. We set $Q$ to *true* in every state $s_J$ of $M_J$ such that

$$M_{jk}, s_J{\uparrow}jk \models EGex_k$$

and to *false* in all other states of $M_J$.

Now $\bigwedge a^J_i . (a^J_i \longrightarrow P^J_j \notin W_J(s_J))$ is equivalent to $s_J \models noblock(i,j)$, where $noblock(i,j) \stackrel{\text{df}}{=} \bigwedge a^J_i . (\{\!\!\{a^J_i.start\}\!\!\} \Rightarrow a^J_i.guard_j)$. Thus, the liveness assumption can be rewritten as

$$\begin{array}{l} \text{for every reachable state } s_J \text{ in } M_J, \\ \qquad M_{jk}, s_J{\uparrow}jk \models EGex_k \ \text{ implies } \ M_J, s_J \models noblock(i,j). \end{array}$$

By definition of $Q$, we have $M_{jk}, s_J{\uparrow}jk \models EGex_k$ iff $M_J, s_J \models Q$. Hence the above is equivalent to

$$\begin{array}{l} \text{for every reachable state } s_J \text{ in } M_J, \\ \qquad M_J, s_J \models Q \ \text{ implies } \ M_J, s_J \models noblock(i,j). \end{array}$$

By CTL semantics, this is equivalent to

$$\begin{array}{l} \text{for every reachable state } s_J \text{ in } M_J, \\ \qquad M_J, s_J \models (Q \Rightarrow noblock(i,j)). \end{array}$$

Finally, we translate the quantification over all reachable states into CTL by prefacing the formula with the $AG$ modality, and evaluating it in all initial states:

[15] i.e., $Q \notin \mathcal{AP}_i \cup \mathcal{AP}_j \cup \mathcal{AP}_k$.

$$M_J, S_J^0 \models AG(Q \Rightarrow noblock(i,j)).$$

To determine the truth assignment to $Q$, we model-check $M_{jk}^r$ (the reachable part of $M_{jk}$) for the formula $AGEGex_k$. We employ the CTL model-checking algorithm of Clarke et al. [1986], which marks all states in $M_{jk}$ with all subformulae of $AGEGex_k$ that are true in the state. In particular, all states of $M_{jk}$ that satisfy $EGex_k$ will be so marked. We can then use this marking to assign the appropriate truth value to $Q$ in each reachable state $s_J$ of $M_J$. Finally, we model-check $M_J$ with respect to $AG(Q \Rightarrow noblock(i,j))$ to determine whether the liveness assumption holds or not, again using the model-checking algorithm of Clarke et al. [1986].

The algorithm of Clarke et al. [1986] has time complexity linear in both the structure and the formula being checked. Here we invoke it twice, once for $M_{jk}^r$ with respect to $AGEGex_k$ and once for $M_J^r$ with respect to $AG(Q \Rightarrow noblock(i,j))$. Since $|AGEGex_k|$ is constant, and $|M_{jk}^r| < |M_J^r|$ is easily seen to hold, the time complexity of model-checking $M_J^r$ dominates. Since $noblock(i,j)$ is quantified over all the moves of $P_i^J$, its size is $O(|P_i^J|)$. Hence, the liveness assumption can be model-checked in time $O(|M_J^r| \cdot |P_i^J|)$. Now $|M_J^r|$ is $O(|P_k^\ell|^3 \cdot 2^{2|\mathcal{SH}_{k\ell}|})$, since the $J$-system contains three processes and two sets of pairwise shared variables. Also $|P_i^J|$ is $O(|P_\ell^k|)$, since $|J|$ is fixed. Hence the overall complexity can be rewritten as $O(|P_k^\ell|^4 \cdot 2^{2|\mathcal{SH}_{k\ell}|})$. Note that the check must be made for the case $i \neq k$ and also for the case $i = k$.