Blocking Detection in Discrete Event Systems

Sherif Abdelwahed
sherif.abdelwahed@vanderbilt.edu
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN, USA

W. M. Wonham
wonham@control.toronto.edu
Department of Electrical Engineering
University of Toronto
Toronto, Ontario, Canada

Abstract

In this paper we present an approach for blocking detection in multiprocess DES. In the proposed approach, potential blocking states are first identified by examining shared transitions in the system components and then tested for their reachability.

1 Introduction

Blocking is a major problem in multiprocess discrete event systems (DES). Blocking originates from the nature of the synchronous communication between the components. Depending on the existence of possible transitions, blocking states can be described as either deadlock or livelock. Deadlock occurs when the system resources have been occupied in such a way that the set of system components cannot continue their activities. Livelock, on the other hand, occurs when the system is trapped in endless cycles that do not lead to successful termination. Static reachability analysis procedures provide a method for automatically detecting blocking situations in multiprocess systems. However, they usually require exhaustive searching of the state space of the system and hence suffer from the state explosion problem of multiprocess systems.

Several approaches have been proposed to avoid searching the entire state space of the system by exploiting certain regularities in the system structure. Partial order reduction [4] is an instance of these approaches in which the effect of representing concurrency is alleviated with interleaving. Symmetrical reduction is another approach in which symmetry in the system is traced back to its components and then exploited to reduce the complexity of the reachability analysis [5]. See [2] for a survey on reduction techniques for blocking detection in logical systems.

In this paper, an alternative approach is proposed to identify blocking states of both forms for multiprocess discrete event systems. In this approach only relevant parts of the system behavior are explored. In DES with relatively few shared events, the saving offered by this approach could surpass its overhead and the overall computation could be significantly less than direct reachability analysis procedures.

The paper is organized as follows. Section 2 introduces basic facts and notation used in this paper. In Section 3 we introduce the blocking problem in multiprocess discrete event systems. A procedure for deadlock detection problem is introduced in Section 4. A conceptually similar livelock detection procedure is presented in Section 5. Proofs of the propositions and theorems in this paper can be found in [1].

2 Preliminaries and Notation

Let Σ be an alphabet representing the events in the process under consideration. A *string* or word is a sequence of events. We will write Σ^+ for the set of all nonempty finite strings with events in Σ, and $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$, where $\epsilon \notin \Sigma$ denotes the empty string. A *language* over the alphabet Σ is any subset of Σ^*. The set of languages over Σ will be denoted $\mathcal{L}(\Sigma)$. A string $s' \in \Sigma^*$ is a *prefix* of $s \in \Sigma^*$, denoted $s' \leq s$, if there exists $u \in \Sigma^*$ such that $s'u = s$. The *prefix closure* of a language $H \subseteq \Sigma^*$, denoted $\overline{H}$, is the set of all strings in Σ^* that are prefixes of strings in H. The *complement* of a language $L \subseteq \Sigma^*$ is defined as $\Sigma^* - L$ and is denoted L^c.

An *automaton* is a 5-tuple structure $A = (Q, \Sigma, \delta, q_o, Q_m)$, where Q is a finite set of states, Σ is a finite nonempty set of events, $\delta : Q \times \Sigma \to Q$ is a (partial) *transition function*, $q_o \in Q$ is the *initial state*, and $Q_m \subseteq Q$ is a nonempty set of *marker states*. If $\delta(q, \sigma)$ is defined, then we say that σ is *eligible* at q in A. This can also be expressed by the map $Elig_A : Q \to Pwr(\Sigma)$ which assigns to each state in A the set of eligible events. The map δ is extended to strings in the usual way. We will write $\varrho_A(q)$ to denote the set of all states reachable from q in A. For a language $L \in \mathcal{L}(\Sigma)$, we will write $A(L)$ to denote the minimal automaton that generates L.

We will extend the above notation to handle multiprocess systems. Let I be the index set of a collection of processes. An *alphabet vector* over I is a set $\{\Sigma_i | i \in I\}$ of alphabets. In the following we will use bold letters to distinguish vector quantities. Let $\mathbf{\Sigma} = \{\Sigma_i | i \in I\}$ be an alphabet vector. The union of all alphabets in

$\boldsymbol{\Sigma}$ will be denoted $\alpha(\boldsymbol{\Sigma})$ or simply Σ. We will write Σ_s or $\alpha_s(\boldsymbol{\Sigma})$ for the set of shared (synchronous) events in $\boldsymbol{\Sigma}$, namely $\Sigma_s = \bigcup_{i \neq j} (\Sigma_i \cap \Sigma_j)$. A multi-process environment will be referred to as a *process space*.

A *language vector* over $\boldsymbol{\Sigma}$ is a set $\boldsymbol{L} = \{L_i \subseteq \Sigma_i^* | i \in I\}$. The set of all language vectors over $\boldsymbol{\Sigma}$ is denoted $\mathfrak{L}(\boldsymbol{\Sigma})$. The language L_i is called the ith *component* of $\boldsymbol{L}$. The decomposition (vector projection) map $\boldsymbol{P}_{\boldsymbol{\Sigma}} : \mathcal{L}(\Sigma) \rightarrow \mathfrak{L}(\boldsymbol{\Sigma})$ associates each language $L \in \mathcal{L}(\Sigma)$ with the language vector $\{P_i L \,|\, i \in I\}$ where $P_i : \Sigma^* \rightarrow \Sigma_i^*$ is the natural projection map that erases all events other than those of the ith component of $\boldsymbol{\Sigma}$. On the other hand, the composition (synchronous product) map $B_{\boldsymbol{\Sigma}} : \mathfrak{L}(\boldsymbol{\Sigma}) \rightarrow \mathcal{L}(\Sigma)$ associates each language vector with its synchronous product $\|\boldsymbol{L}$ as defined in [8]. To simplify notation we will write $\boldsymbol{P}B_{\boldsymbol{\Sigma}}$ to denote the composition $\boldsymbol{P}_{\boldsymbol{\Sigma}} \circ B_{\boldsymbol{\Sigma}} : \mathfrak{L}(\boldsymbol{\Sigma}) \rightarrow \mathfrak{L}(\boldsymbol{\Sigma})$, and $B\boldsymbol{P}_{\boldsymbol{\Sigma}}$ for the composition $B_{\boldsymbol{\Sigma}} \circ \boldsymbol{P}_{\boldsymbol{\Sigma}} : \mathcal{L}(\Sigma) \rightarrow \mathcal{L}(\Sigma)$.

3 Blocking in Multiprocess DES

A dynamic system is said to be *blocked* if it contains a reachable state from which the system cannot reach any marked state. These states are referred to as *blocking* states. In multiprocess environments, the composite system may contain blocking states even if its individual components are not blocked. In general, any vector language $\boldsymbol{L}$ in a process space $\boldsymbol{\Sigma}$ satisfies

$$\overline{B_{\boldsymbol{\Sigma}}(\boldsymbol{L})} \subseteq B_{\boldsymbol{\Sigma}}(\overline{\boldsymbol{L}})$$

where $\overline{\boldsymbol{L}} = \{\overline{L_i} \,|\, i \in I\}$. However, it is easy to see that the other direction of the inclusion does not hold in general. A language vector $\boldsymbol{L}$ is said to be *unblocked* or *live* if $\overline{B_{\boldsymbol{\Sigma}}(\boldsymbol{L})} = B_{\boldsymbol{\Sigma}}(\overline{\boldsymbol{L}})$. Clearly, $\boldsymbol{L}$ is live if the set $\{P_i^{-1} L_i \,|\, i \in I\}$ is *non-conflicting*, that is, satisfies

$$\overline{\bigcap_{i \in I} P_i^{-1} L_i} = \bigcap_{i \in I} \overline{P_i^{-1} L_i}$$

It is important to distinguish between two forms of blocking in discrete event systems. The first corresponds to the case when an unmarked reachable state cannot reach any other state. Such a state is referred to as a *deadlock* state. Otherwise, if a blocking state can reach unmarked states it is referred to as a *livelock* state. Clearly, the two forms are exhaustive, that is, a blocking state must be of either form but not both.

4 Deadlock Detection in Multiprocess Systems

Let $\boldsymbol{L}$ be a language vector in $\boldsymbol{\Sigma}$. A string $s \in B_{\boldsymbol{\Sigma}}(\overline{\boldsymbol{L}})$ is called *nonterminal* if it is not in $B_{\boldsymbol{\Sigma}}(\boldsymbol{L})$. A language vector $\boldsymbol{L}$ is said to be *deadlock-free* if the set of eligible events at every nonterminal string in $B_{\boldsymbol{\Sigma}}(\overline{\boldsymbol{L}})$ is not empty. Formally, $\boldsymbol{L}$ is *deadlock-free* if

$$(\forall s \in (B_{\boldsymbol{\Sigma}}(\overline{\boldsymbol{L}}) - B_{\boldsymbol{\Sigma}}(\boldsymbol{L})))(\exists \sigma \in \Sigma) \quad s\sigma \in B_{\boldsymbol{\Sigma}}(\overline{\boldsymbol{L}})$$

Deadlock can be detected by exploring the entire state space of the system and identifying those non-marked states with no eligible events. An alternative approach is to identify potential blocking states first and then to check their reachability. This approach will be referred to as *detect-first* approach. The key to this approach is given in the following result which provides a characterization for deadlock freeness in multiprocess discrete event systems. For a given subset J of I, let $\Sigma_{(J)}$ denote the set of events shared exclusively by the set of J components. Namely, $\Sigma_{(J)} = \cap_{j \in J} \Sigma_j - \cup_{i \in I - J} \Sigma_i$.

Theorem 4.1 The language vector $\boldsymbol{L}$ is deadlock-free if and only if

$$(\forall s \in B_{\boldsymbol{\Sigma}}(\overline{\boldsymbol{L}}) - B_{\boldsymbol{\Sigma}}(\boldsymbol{L}))(\exists J \subseteq I) \bigcap_{j \in J} Elig_{L_j}(P_j s) \cap \Sigma_{(J)} \neq \emptyset \qquad \square$$

The above theorem shows that deadlock can be traced by examining the set of eligible events at local strings generated by the system components. Clearly, if a deadlock occurs at a given string then it also occurs at the corresponding state in the system's automaton. Hence, a language vector $\boldsymbol{L}$ cannot be deadlocked after a string s if (and only if) there exists an asynchronous (local) event enabled at any of the vector projection components of this string. Consequently, in detecting global deadlocked states, we only need to consider those combinations of local states all of whose eligible events are shared. These combinations (global states) are further refined by testing them with respect to the above criterion to determine if they identify *potential* deadlock states in the composite system. Each potential deadlock state is then traced backward to check if it is reachable in the composite system. The following Proposition, shows that this search can also be performed by considering only the shared behavior of the system.

Proposition 4.1 Let $\boldsymbol{L}$ be a language vector in a process space $\boldsymbol{\Sigma}$. Then

$$B_{\boldsymbol{\Sigma}}(\boldsymbol{L}) = \emptyset \quad \Longleftrightarrow \quad B_{\boldsymbol{\Sigma}}(P_s(\boldsymbol{L})) = \emptyset.$$

$\square$

Therefore, a given potential deadlock state is reachable if and only if the corresponding state in the projected composite system is reachable. Testing reachability through the projected system, $P_s(B_{\boldsymbol{\Sigma}}(\boldsymbol{L}))$, can offer a computational advantage in loosely coupled systems, given that $P_s(B_{\boldsymbol{\Sigma}}(\boldsymbol{L})) = B_{\boldsymbol{\Sigma}}(P_s(\boldsymbol{L}))$, that is, synchronous product is commutative with the projection of shared events [1].

To further clarify the detect-first approach for deadlock detection, consider the language vector $\boldsymbol{L}$ represented by a set of automata $A_i = (\Sigma_i, Q_i, \delta_i, q_{oi}, Q_{mi})\}$, $i \in I$ where $A_i = A(L_i)$. Every automaton A_i is assumed trim and therefore live. Let $A = \|_{i\in I} A_i$ be the synchronous product automaton given by the tuple $(\Sigma, Q, \delta, q_o, Q_m)$ where $\Sigma = \cup_{i\in I}\Sigma_i$ and $Q = Q_1 \times Q_2 \times \ldots \times Q_n$ be the set of all possible states in A. Transitions in the automaton A are defined through the synchronous product operation in the usual way. Clearly, $\overline{L_m(A)} = B_{\boldsymbol{\Sigma}}(\boldsymbol{L})$. However $L(A)$ may not be equal to $\overline{B_{\boldsymbol{\Sigma}}(\boldsymbol{L})}$ due to possible blocking states in the composite system. Define $\texttt{PDS}(A)$ as the set of potential deadlock states in A as outlined by the last theorem, namely

$$\texttt{PDS}(A) = \{(q_1, \ldots, q_n) \in Q \mid (\forall J \subseteq I) \bigcap_{j\in J} Elig_{A_j}(\{q_j\} \cap (Q_j - Q_{jm})) \cap \Sigma_{(J)} = \emptyset\}$$

Therefore, the set $\texttt{PDS}(A)$ can be computed by testing only unmarked local states[1] where all eligible events are shared. The set of (actual) deadlock states in A is then equal to $\texttt{PDS}(A) \cap \texttt{Reach}(A)$, where $\texttt{Reach}(A)$ denotes the set of all reachable states in A. A direct backward reachability procedure can be used to determine if a given state $q \in Q$ is reachable in A.

For loosely coupled systems, testing that a given global state q belongs to the set $\texttt{Reach}(A)$ can be conducted more efficiently by considering only the shared behavior of A namely $P_s(A)$. The projection map P_s commutes with the synchronous product operation and therefore $P_s(A)$ can be computed indirectly as $\|_{i\in I} P_s(A_i)$. Based on Proposition 4.1, $q \in \texttt{Reach}(A)$ if and only if there exists $x \in X$ where $x \in \texttt{Reach}(P_s(A))$ and $q \in x$.

The complexity of the detect-first procedure depends to a large extent on the shared behavior of the system. The initial part of the procedure identifies those combinations of local states with all shared eligible events. Therefore the worst case complexity of this part is of the order of $|Q_1^s| \times \ldots \times |Q_n^s|$ where $Q_i^s \subseteq Q_i$ denotes the set of states in the ith component of the system having only shared eligible events. However, the typical complexity of this part is much smaller than the worst case as many combinations can be excluded during the test. For instance, if for some $J \subseteq I$ we have $\bigcap_{j\in J} Elig_{A_j}(q_j) \cap \Sigma_{(J)} \neq \emptyset$, then it follows from Theorem 4.1 that any global state containing the set $\{q_j \mid j \in J\}$ is deadlock-free. In fact, the efficiency of computing the set $\texttt{PDS}(A)$ can vary substantially depending on the order in which the set of local states is tested.

In the second part of the procedure, the reachability of potential deadlock states is checked. The worst case complexity of this part is on the order of the state size of the composite system, $|Q|$. However, this part would explore only those states leading to potential deadlock states. The actual overhead in this part is the tracing of unreachable potential deadlock states. However, the rate of early termination of such backward tracing can be high, particularly in loosely coupled systems. Also, as discussed earlier, a significant reduction of the complexity of the reachability test can be obtained by considering only the shared behavior of the system.

[1]This is based on the convention that $Elig_B(\emptyset) = \emptyset$ for any automaton B.

Example 4.1 The system shown in the figure below is a modified version of the Milner scheduler [6]. It consists of a set of simple components called cyclers. Figure 1 shows the automaton representation of the ith cycler, where $i \in [0 \ldots N]$ for a scheduler consisting of $N+1$ cyclers. In this system, state $1i$ is the initial and marker state for the first cycler ($i = 0$) while state $0i$ is the initial and marker state for the remaining cyclers ($i \in [1 \ldots N]$). The subscripts of the local components are taken as mod N, so that $x_{N+1} = x_0$.

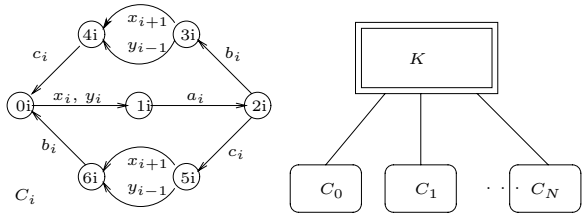
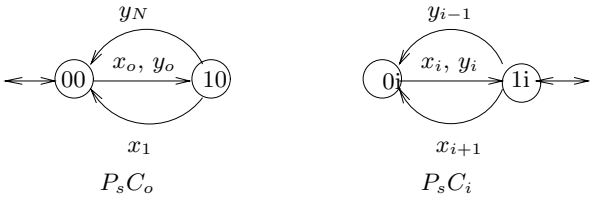


Figure 1: The process scheduler

Applying the detect-first procedure, potential deadlock states are selected from those states with all shared eligible events, that is, the set

$$Q^s = \{(q_{0r_o}, q_{1r_1}, \ldots, q_{Nr_N}) | (\forall j \in [0 \ldots n] \;\; r_j \in [0, 3]\}$$

The set Q^s can be tested thoroughly with respect to the definition of $\texttt{PDS}(A)$. However, working by eliminating those subsets of Q^s that do not correspond to potential deadlock states can be more efficient for this system. Clearly, any global state containing any of the combinations $(q_{i0}, q_{(i+1)3})$ or $(q_{i0}, q_{(i-1)3})$ for $i \in [0 \ldots N]$ cannot be a deadlock state. Excluding these cases will leave only two potential deadlock states, namely $(q_{0r}, q_{1r}, \ldots, q_{Nr})$ where $r \in [0, 3]$. Backward tracing will reveal that neither state is reachable. To reach this conclusion, the backward reachability test explores 2^{N+1} global states for the first potential-deadlock state ($r = 0$) and 3 global states for the second one (the composite system contains approximately 7^{N+1} states). Backward reachability can also be performed using the projection of the shared behaviour of the system components shown in the following diagram.



Potential deadlock states in the system correspond to the set of states $(q'_{0(r-1)}, q'_{1r}, \ldots, q'_{Nr})$ where $r \in [0,1]$ in the composite of the components projection. Checking the reachability of these states requires testing only two global states in the projected system. The complexity of computing the projections is of $O(N)$.

5 Livelock Detection in Multiprocess Systems

The difficulty in detecting livelock states stems from the dependency on the existence of deadlock states. Particularly, a livelock state can be defined recursively as a state with a reachable domain consisting entirely of a nonempty set of deadlock or livelock states. Detecting livelock states may therefore depend on identifying deadlock states. Livelock detection can be simplified by considering those states that can only reach livelock states. Note that, this is the only source of blocking if the system is deadlock-free. Given the assumption of finite state space, the recursive characterization of livelock leads to the conclusion that any livelock state must exist within a loop, or more precisely a clique, in the global system.

Let $A = (\Sigma, Q, \delta, q_o, Q_m)$ be a deadlock-free automaton. Based on the above description, a state $q \in A$ is a *livelock* state iff,

$$\varrho_A(q) \cap Q_m = \emptyset \ \wedge \ (\forall q' \in \varrho_A(q)) Elig_A(q') \neq \emptyset$$

The first condition characterizes also semi-livelock states, while the last condition distinguishes (strict) livelock states as those which can only lead to another livelock state. The automaton A is said to be *livelock-free* if it does not contain any livelock state.

Let X be a nonempty subset of Q in A. Let δ^X denote the restriction of δ to the states in X, and let $\varrho_A^X : X \to Pwr(X)$ be the reachability map that assigns each state $x \in X$ to those states in X that are reachable from x via transitions from δ^X. The set X is called a *clique* in A if every state in X can reach any other state in X via transitions from δ^X. Formally, a set of states X is a clique in A if $X \neq \emptyset$ and

$$(\forall x, x' \in X) \quad x' \in \varrho_A^X(x)$$

A state $x_o \in X$ is called an *input state* for the clique X if x_o is the initial state of A or

$$(\exists q \in Q - X)(\exists \sigma \in \Sigma) \quad \delta(q, \sigma) = x_o$$

In this case the transition (q, σ, x_o) is said to be an *input transition* for the clique X. Similarly, a state $x_m \in X$ is called an *output state* in X if it is a marker state in A or there exists a transition (q, σ, x_m) with $q \in Q - X$. In this case, the transition (q, σ, x_m) is then an *output transition* for the clique Q'. For a clique set X we will write X_o for its set of input states and X_m for its set of output states. Therefore, a system may enter a clique from one of its input states, then it can stay in the clique indefinitely or exit the clique from one of its output states. The tuple (X, X_o, X_m) will be referred to as a *clique structure* in A.

A clique is said to be reachable if any (and therefore all) of its states is reachable, and is said to be coreachable if any (and therefore all) of its states is coreachable. Note that if an automaton A is reachable then every clique structure in A must contain at least one input state, and if A is coreachable then every clique structure in A must contain at least one output state. A clique is said to be *maximal* if there is no other clique $X' = (X', X'_o, X'_m)$ in A such that $X \subseteq X'$, $X_o \subseteq X'_o$, and $X_m \subseteq X'_m$. A clique is said to be *terminal* if each reachable state from the clique is in the clique, that is, if $\varrho_A(X) = X$. It is easy to see that every terminal clique is maximal but the reverse is not generally true.

Proposition 5.1 Assume that the automaton A is deadlock-free. Then A is livelock-free if and only if every reachable terminal clique in A contains a marked state. □

Therefore, livelock states can be detected by testing the set of terminal cliques in the system[2]. The above result can be used as a basis for developing a detect-first procedure to livelock detection in multiprocess systems. First we need to identify the conditions in the system components that characterize a terminal clique in the composite system. To this end, let (X, X_o, X_m) be a clique structure in A. A *clique machine* is a tuple $M = (X, x_o, X_m)$ where $x_o \in X_o$. The transition function of this machine is δ^X. A clique machine $M = (X, x_o, X_m)$ is said to be maximal if there is no other clique machine $M' = (X', x'_o, X'_m)$ such that $x'_o = x_o$, $X \subseteq X'$, and $X_m \subseteq X'_m$, that is if it is not a submachine of any other clique machine. As mentioned earlier, if A is trim then every clique in A must have at least one input and one output state. Therefore, in this case, each clique is associated with at least one maximal clique machine.

Let $\boldsymbol{A}$ be a set of trim automata $\{A_i = (\Sigma_i, Q_i, \delta_i, q_{oi}, Q_{mi}) \mid i \in I\}$ over a process space $\boldsymbol{\Sigma}$ and let $A = \|A_i = (\Sigma, Q, \delta, q_o, Q_m)$ be their synchronous product. For a nonempty $J \subseteq I$ let $\boldsymbol{M} = \{M_j = (x_{jo}, X_j, X_{jm}) \mid (j \in J)\}$ be a set of clique ma-

[2]Finding all possible cliques in a given graph is a known NP-complete problem as it requires examining all possible subsets of the set of states [3].

chines in $\boldsymbol{A}$. The set $\boldsymbol{M}$ is said to be a *clique vector* if $M = \|\boldsymbol{M}$ is a clique machine, where $\|$ is the synchronous product operation[8] for the process space $\boldsymbol{\Sigma}$. The composite machine M is a tuple (x_o, X, X_m), where $x_o = \{x_{jo} \mid j \in J\}$ is called the *local input state* for M, $X \subseteq \times_{j\in J} X_j$ is the set of states reachable from x_o in M, and $X_m = X \cap \times_{j\in J} X_{jm}$. By definition, the set $\boldsymbol{M}$ is a clique vector if and only if there is a nonempty string connecting x_o to itself through states from X. Let $y \in \times_{i\in I-J} Q_i$ and $x \in X$. We will write (y, x) to denote the global state q corresponding to pair x and y. The tuple $(\boldsymbol{M}, y)$ will be referred to as a *clique module*. The clique module $(\boldsymbol{M}, y)$ is said to be *reachable* if

$$(x_o, y) \in \varrho_A(q_o)$$

That is, the global state (x_o, y) is reachable in the composite system. The clique module $(\boldsymbol{M}, y)$ is said to be *terminal* if

$$(\forall q \in Q)\ \ q \in \varrho_A(x_o, y) \implies (\exists x \in X) q = (x, y)$$

That is, any reachable global state from (x_o, y) must remain in the clique while preserving its y part. To satisfy this condition it is necessary that y be deadlocked with respect to M, i.e., there should be no eligible event at y that is also an eligible event at any $x \in X$. Clearly, this also requires that y have no asynchronous eligible events. Finally, $(\boldsymbol{M}, y)$ is said to be *marked* if

$$(\exists x_m \in X_m)\ \ (x_m, y) \in Q_m$$

That is, a the clique module contains a global marked state. Intuitively, a clique vector $\boldsymbol{M} = \{M_j = (x_{jo}, X_j, X_{jm}) \mid (j \in J)\}$ is a set of local cliques that compose to a clique in the global system independently of any other components in $I - J$. Therefore, for any $y \in \times_{i\in I-J} Q_i$ the tuple $(\boldsymbol{M}, y)$ corresponds to a potential clique in the global system. The tuple $(\boldsymbol{M}, y)$ only needs to be reachable to correspond to an actual clique, that is when (x_o, y) is reachable. The conditions for termination and marking can be formulated directly for this tuple through this correspondence.

Theorem 5.1 Let $\boldsymbol{A}$ be a vector of trim automata in $\boldsymbol{\Sigma}$ where $A = \|\boldsymbol{A}$ is deadlock-free. Then A is livelock-free if and only if every reachable and terminal clique module in $\boldsymbol{A}$ is marked. □

Therefore, livelock situations can be traced from reachable and terminal clique modules that are not marked. Identifying modules, however, requires identifying all cliques in the system components and examining the synchronous behaviour of all possible combinations of these cliques. In general, the number of cliques in a given automaton may exceed its number of states. However, many of these combinations are expected to be eliminated early by examining the criteria for a livelock situation. In summary, a set of clique machines $\boldsymbol{M} = \{M_j = (x_{jo}, X_j, X_{jm}) \mid j \in J\}$ and a set of states $y \in \times_{i\in I-J} Q_i$ in a vector automaton $\boldsymbol{A}$ identifies a livelock situation in the composite system $A = \|\boldsymbol{A}$ iff

1. $(\boldsymbol{M}, y)$ is a clique module: $M = \|\boldsymbol{M} = (x_o, X, X_m)$ satisfies $x_o \in \varrho_M(x_o)$
2. $(\boldsymbol{M}, y)$ is reachable in A: $(x_o, y) \in \varrho_A(q_o)$
3. $(\boldsymbol{M}, y)$ is terminal in A: Any state reachable from (x_o, y) state in A must be in the form (x, y) for some $x \in X$. That is, y is deadlocked with respect to the set of states X.
4. $(\boldsymbol{M}, y)$ is not marked: Every global state (x_m, y) must not be in Q_m, the set of marked states of the composite system.

A clique module is first identified by a composable set of cliques $\boldsymbol{M}$ and set of states y that is deadlocked with respect to the states in $\|\boldsymbol{M}$. The clique module $(\boldsymbol{M}, y)$ identifies a *potential livelock source* if it satisfies any of the above conditions. It becomes a *livelock source* if it satisfies all the above condition. A list of potential livelock sources is built and updated by testing the conditions above for reachability and marking. A clique module $(\boldsymbol{M}, y)$ is removed from the list when it fails any of these conditions.

The detection procedure can be enhanced by utilizing the reachability and coreachability information as they accumulate during the detection process. Another enhancement can be obtained by observing the inclusion of one clique machine into another in the system components. Formally, a clique machine $M = (x_o, X, X_m)$ is a submachine of another clique machine $M' = (x'_o, X', X'_m)$ if $x_o = x'_o$, $X \subseteq X'$, and $X_m \subseteq X'_m$. A clique machine is said to *maximal* if it is not a submachine of any other clique machine. Let M be a submachine of M' and let $\boldsymbol{M}$ be a clique vector containing M, and $\boldsymbol{M'}$ be the clique vector obtained from $\boldsymbol{M}$ by substituting M with M'. It is easy to see that if $\boldsymbol{M'}$ does not satisfy any one of the above conditions then $\boldsymbol{M}$ does not satisfy it either. And if $\boldsymbol{M'}$ satisfies all these conditions then it is a source of livelock and the overall set of states, including $\boldsymbol{M}$ irrespective of its status with respect to the above condition, needs to be examined. Therefore, we only need to consider the set of maximal clique machines in the detecting procedure. The detect-first procedure outlined above can be used to confirm that the system in Example 4.1 is livelock-free, simply because the only possible clique module is the whole system which must be marked if the composite system is a clique.

Example 5.1 In this example we consider the alternating bit protocol (ABP) as presented in [7]. The basic components of the protocol are shown in Figure 2.

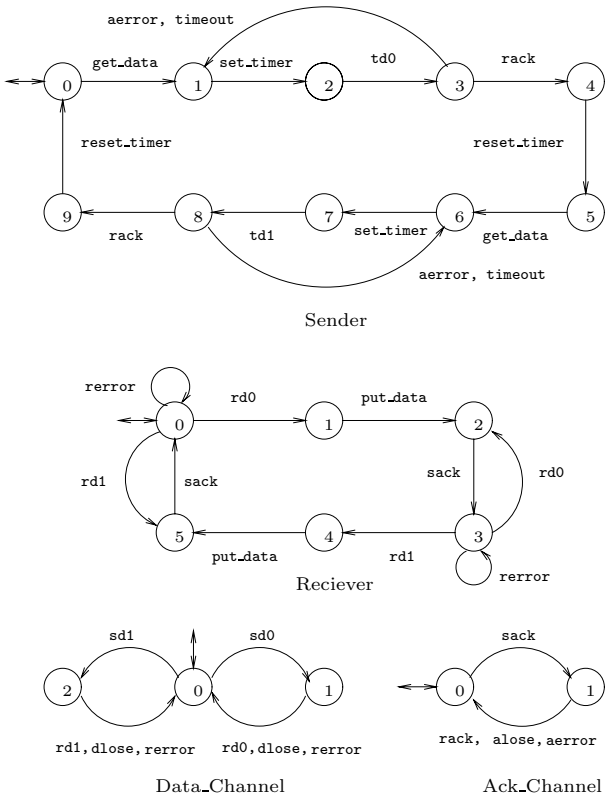


Figure 2: The alternating bit protocol model

Consider first the deadlock issue. The local sets of states with only shared eligible events are

$$\texttt{OSE}(\text{Sender}) = \{2, 7\}, \quad \texttt{OSE}(\text{D-Channel}) = \{0\},$$
$$\texttt{OSE}(\text{Receiver}) = \{0, 2, 3, 5\}, \quad \texttt{OSE}(\text{A-Channel}) = \{0\}$$

where $\texttt{OSE}(A)$ denote the set of states in A that have only shared eligible events. The set of potential deadlock states is then tested based on Theorem 4.1 from which we can confirm that the global states $(2, X, 0, X)$ and $(7, X, X, 0)$, where X denotes any state in the corresponding component, cannot correspond to a deadlock state. However, these two combinations are exactly all possible global states with only shared eligible events. This shows that the protocol is deadlock-free.

To detect livelock states in the protocol, the set of maximal clique machines in each component has to be identified. In the sender we can identify the following maximal clique machines.

$$S1 = (Q_1, 0, \{0\}), \quad S2 = (\{1, 2, 3\}, 1, \{3\}),$$
$$S3 = (\{1, 2, 3\}, 6, \{6, 7, 8\})$$

And in the receiver machine we have,

$$R1 = (Q_2, 0, \{0\}), R2 = (\{2, 3\}, 2, \{3\}), R3 = (\{3\}, 3, \{3\}),$$
$$R4 = (\{5, 0\}, 0, \{0\}), R5 = (\{5, 0\}, 5, \{0\})$$

In both the Data-Channel and the Ack-channel the only maximal clique is the machine itself. We denote these cliques as $D1$ and $A1$ respectively. First consider those combinations containing $S1$. Examining this clique will show that it needs to synchronize with events from the other three machines in order to remain as a loop in the composite structure. Therefore, we need to consider only clique combinations containing a clique from each component. The combinations $(S1, R1, D1, A1)$, $(S1, R4, D1, A1)$, and $(S1, R5, D1, A1)$ are excluded as they are always marked. The combinations $(S1, R2, D1, A1)$, and $(S1, R3, D1, A1)$ are also excluded as their local initial states $(0, 2, 0, 0)$, and $(0, 3, 0, 0)$ are not reachable. Testing the remaining combinations will reveal that the system is livelock-free. □

As shown in the above example, there are several ways a given clique vector $\boldsymbol{M}$ could be excluded from the list of potential livelock generators. The efficiency of detecting livelock using the detect-first approach depends to a large extent on the way the given conditions are tested. The detection procedure can be enhanced by conducting some tests before others. For instance, checking if a given clique vector $\boldsymbol{M}$ is reachable or terminal may be more efficient than testing if it is a module (computing $\|\boldsymbol{M}$) if the size of the clique sets in $\boldsymbol{M}$ is relatively large. The reverse can be true if the size of the clique sets in $\boldsymbol{M}$ is relatively small.

References

[1] S. Abdelwahed. *Interacting Discrete Event Systems: Modelling, Verification, and Supervisory Control*. PhD thesis, University of Toronto, 2002.

[2] J. Corbett. Evaluating deadlock detection methods for concurrent software. *IEEE Trans. on Software Engineering*, 22(3):1–22, 1996.

[3] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP Completeness*. W.H. Freeman and Company, New York, 1979.

[4] P. Godefroid. *Partial Order Methods for the Verification of Concurrent Systems*. PhD thesis, Univ. de Liege, 1994.

[5] C. E. McDowell. A practical algorithm for static analysis of parallel algorithms. *Journal of Parallel and Distributed Processing*, 6(3):515–536, June 1989.

[6] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer-Verlag, 1980.

[7] K. Rudie. *Decentralized Control of Discrete Event Systems*. PhD thesis, Univ. of Toronto, 1992.

[8] W.M. Wonham. *Notes on Control of Discrete-Event Systems*. ECE Department, University of Toronto, revised 1 July 2002. `http://www.control.utoronto.ca/DES`.