Hindawi Publishing Corporation
Journal of Control Science and Engineering
Volume 2012, Article ID 424051, 14 pages
doi:10.1155/2012/424051

Research Article

Process Completing Sequences for Resource Allocation Systems with Synchronization

Song Foh Chew,[1] Shengyong Wang,[2] and Mark A. Lawley[3]

[1] *Department of Mathematics and Statistics, Southern Illinois University, Edwardsville, IL 62026, USA*
[2] *Department of Mechanical Engineering, University of Akron, Akron, OH 44325, USA*
[3] *School of Biomedical Engineering, Purdue University, West Lafayette, IN 47907, USA*

Correspondence should be addressed to Mark A. Lawley, malawley09@yahoo.com

Received 21 January 2012; Accepted 12 April 2012

Academic Editor: Jin-Shyan Lee

This paper considers the problem of establishing live resource allocation in workflows with synchronization stages. Establishing live resource allocation in this class of systems is challenging since deciding whether a given level of resource capacities is sufficient to complete a single process is NP-complete. In this paper, we develop two necessary conditions and one sufficient condition that provide quickly computable tests for the existence of process completing sequences. The necessary conditions are based on the sequence of completions of n subprocesses that merge together at a synchronization. Although the worst case complexity is $O(2^n)$, we expect the number of subprocesses combined at any synchronization will be sufficiently small so that total computation time remains manageable. The sufficient condition uses a reduction scheme that computes a sufficient capacity level of each resource type to complete and merge all n subprocesses. The worst case complexity is $O(n \cdot m)$, where m is the number of synchronizations. Finally, the paper develops capacity bounds and polynomial methods for generating feasible resource allocation sequences for merging systems with single unit allocation. This method is based on single step look-ahead for deadly marked siphons and is $O(2^n)$. Throughout the paper, we use a class of Petri nets called Generalized Augmented Marked Graphs to represent our resource allocation systems.

1. Introduction

In recent years, liveness-enforcing supervisory control has been an active area of research for resource allocation systems characterized by processes with highly ordered, linear workflows. This research has been motivated to a large degree by the need to control resource allocation in large, highly automated manufacturing systems, where process workflow is highly sequential and is typically prespecified in a product's process plan. In brief, a sequential resource allocation system (RAS) consists of a set of resources, each available at a finite level, and a set of processes that progresses through sequences of processing stages, with each stage requiring a predetermined set of the system resources. Furthermore, a process instance is allowed to advance to its next stage only when it has been granted the complete set of required resources and only then will it release the currently held resources that are not required for the following stage.

Because the resource allocation schemes discussed above are embedded in the operation of many technologically advanced systems, a complete understanding of their worst case behaviors is essential when devising operating logic for their control. Indeed, if resource allocation is not properly constrained, the sequential RAS will attain resource allocation states from which additional allocation-deallocation of some subset of resources is not possible. This situation is highly undesirable, because resource allocation stalls, the involved processes and the resources they hold are idle, and outside intervention to resolve and reset the system is required. Liveness enforcing supervision seeks to avoid these situations and maintain completely smooth operation by imposing an appropriate supervisory control policy.

Reveliotis et al. [1] present a taxonomy for sequential RAS based on the structure of the allocation requests associated with various processing stages. This taxonomy includes (i) single-unit (SU) RAS, which admits only linearly ordered process sequences with resource requests corresponding to standard unit vectors, (ii) conjunctive (C) RAS, which admits linearly ordered process flows with arbitrary resource requests, and (iii) disjunctive/conjunctive (D/C) RAS, which allows the process to use alternative workflow sequences. Lower-numbered classes in the taxonomy are specializations of the higher-numbered and therefore present simpler behaviors which are more easily analyzed and controlled. Indeed, many results on RAS liveness and the synthesis of tractable liveness enforcing supervisors (LES) have been developed for the SU-RAS class, see, for example, [2, 3] for seminal papers. Researchers have also addressed the problem in the context of the more general classes of D-RAS, C-RAS, and D/C-RAS, see [4, 5] for early results. An interesting discussion that provides a unifying perspective for many of these results, and also highlights the currently prevailing issues in the area, can be found in [6]. Additional recent reviews are provided in [7, 8].

In [9], Reveliotis et al. extends the taxonomy of [1] to include RAS with process synchronizations, that is, RAS where a process may consist of several subprocesses operating independently until some synchronization stage is attained, at which point subprocesses recombine through merging and splitting and then continue as a new set. We shall refer to this class of RAS as A/D-RAS (assembly/disassembly RAS), since, in the case of manufacturing, this class covers products with both assembly and disassembly in their specified workflow. We notice, however, that synchronization also commonly occurs in project management and business workflow scenarios where finite resources must be allocated to competing tasks, which must eventually merge and spawn successor tasks.

From the perspective of logical analysis and control, a major difference between the A/D-RAS and those addressed in the taxonomy of [1] is that we can no longer quickly be sure that the given level of resource capacities is sufficient to complete even a single process. More specifically, since a single process may consist of several concurrent and independently operating subprocesses, each requesting, using, and holding resources, there is no guarantee that resources are of sufficient capacity to allow these subprocesses to attain required synchronization states. In this paper, we refer to this issue as the "quasi-liveness" problem since, by definition, an underlying Petri net model of the A/D-RAS will be quasi-live if, for every transition of the net (including those representing synchronizations), there exists a sequence of transition firings (resource allocations) that enables that transition. In [9], it is established that the lack of quasi-liveness in the A/D-RAS can be explained by the presence of a particular type of deadly marked siphon in the underlying net dynamics and that testing quasi-liveness, a rather easy task for nets modeling the D/C-RAS, now becomes an NP-complete problem (cf. also [10] for a formal proof on the NP-completeness of the quasi-liveness problem in the considered RAS class). Thus, assessing process quasi-liveness raises important and novel research problems to be addressed for this RAS. For quasi-live processes, an additional issue is identifying sequences of resource allocations that enable the involved process synchronizations. Once such sequences have been identified, standard D/C-RAS deadlock avoidance policies can be implemented to control concurrent allocation of resources across several concurrently operating processes.

We note that in [11], Xie and Jeng also study resource allocation in systems with synchronizations by analyzing a class of ordinary Petri nets called extended resource control nets (ERCN). More specifically, they develop structural characterizations for the ERCN quasi-liveness and liveness that are based on the notion of empty siphons. In other work, Wu et al. [12] model assembly/disassembly processes using resource-oriented Petri nets. Based on the models, a deadlock control policy is proposed and proved to be computationally efficient and less conservative than the existing policies in the literature. Hsieh [13] develops a subclass of Petri net models called nonordinary controlled flexible assembly Petri nets with uncertainties for assembly systems and studies their robustness to resource failure. Hu et al. [14] proposes a class of Petri nets to study automated manufacturing systems with either flexible routes or assembly operations. Using structural analysis, the authors show that liveness of such systems can be attributed to the absence of under-marked siphons.

Our work, on the other hand, places more emphasis on the associated design and control problems, seeking first to find resource levels that guarantee quasi-liveness and then to find resource allocation sequences that enable synchronization transitions. In [15, 16], we model the A/D-RAS using a subclass of Petri nets known as Generalized Augmented Marked Graphs (G-AMG). Based upon the notion of reachability graph, we present an algorithm that determines the quasi-liveness of a process subnet by enumerating all execution sequences that are resource-enabled under the considered resource availability; if the net is quasi-live, there will be at least one sequence that leads to process completion. For a quasi-live process, the reachability graph provides complete information about the resource allocation sequences that can be used. Since the graph is exponential in size, it is generally necessary to select a smaller subset of sequences to use for supervision. Based on the work presented in [15, 16], Choi [17] develops a mixed integer program that selects a small subset of process completing sequences for the development of liveness enforcing supervisors. This defines a manageable set of realizable behaviors the system can exhibit. The subset is selected such that a performance controller, posed as a Markov decision process, has the greatest potential to optimize system performance.

In this paper, we seek to develop more tractable methods of identifying process completing sequences for certain subclasses. More specifically, we define a special case of G-AMG, called G-AMG$_A$, which models a RAS comprising only "assembly" or merging operations. For RAS modeled by G-AMG$_A$'s, we develop two necessary conditions for quasi-liveness which provide quick tests. We also develop a polynomial net reduction algorithm that can be used to compute resource levels sufficient to assure quasi-liveness.

We then turn our attention to the more restricted subclass of G-AMG$_A$, G-AMG$_{ASU}$, in which resource allocation is of the single-unit type. For this class, we develop resource bounds that guarantee polynomial quasi-liveness. We also present a polynomial algorithm for computing resource-feasible sequences when the resource bounds are met.

We organize the remainder of the paper as follows. Section 2 presents and discusses our A/D-RAS model. Section 3 develops the necessary conditions, the sufficient condition, and the net reduction algorithm for generating a process completing sequence for the G-AMG$_A$. Section 4 develops sufficient resource bounds along with a polynomial algorithm for generating process completing sequences in assembly systems with single unit resource allocation, G-AMG$_{ASU}$. Finally, Section 5 provides concluding remarks and discusses future research.

2. The G-AMG Model for the A/D-RAS

References [9, 12] formally define the G-AMG structure for the A/D-RAS. For completeness, the Appendix repeats this definition. Figure 1 provides an example of a G-AMG process net.

Note from Figure 1 that the net has an initial place, p_0, marked with a single token. This represents the uninitiated process. The initial transition, t_I, serves as the order release transition, which initiates production of the five subprocesses. The places of $t_I\bullet$, call this set P_I, hold the released subprocess orders. No resources are allocated to subprocesses in P_I, that is, t_I merely releases orders for the subprocesses it does not allocate resources. This is indicated by the zero need vector associate with places in P_I.

We use P_S to represent the set of places that model processing operations, typically those with nonzero resource need, and T_S to represent those transitions that allocate-deallocate resources. Thus, resource places are only connected to transitions in T_S. Note that the sequential logic underlying the execution of the set of subprocesses is expressed by the induced subnet $P_S \cup T_S$.

Places of P_S are labeled with resource need for three resource types. We do this to simplify the figure. In fact, each resource type has its own place (the set of resource places is P_R) and is marked with a number of tokens representing its capacity (we will denote the capacity or resource, r_i, as C_i). Consider Figure 2, illustrating the connectivity for resource r_1. The weight $W(r_1, t_1) = 1$ represents the number of units of r_1 requested by the subprocess at t_1. The needs of a process place $p \in P_s$ with respect to some resource $r_i \in P_R$, are expressed by the value of $u_i(p)$, where u_i is the p-semiflow introduced by item 5 of Definition A.11.

Note that resource types support the execution of the different requesting subprocesses in a reusable fashion, that is, their utilization does not diminish their capacity.

Firing of $t_7 \in T_S$ represents the completion of the process. This event deallocates all resources and places a token in the final completion place, $p_F \in P_F$. When this happens, the final transition, t_F, which signals process completion, is allowed to fire and a new process release is enabled. Only

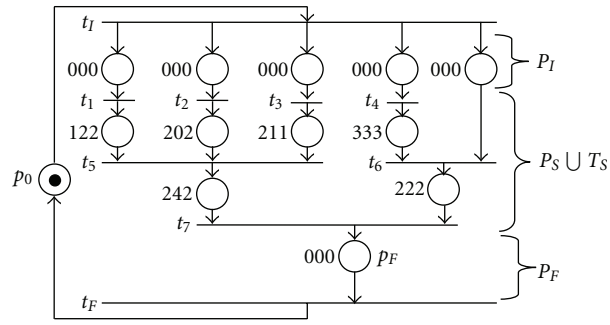

Figure 1: Process net with needs for three resource types.

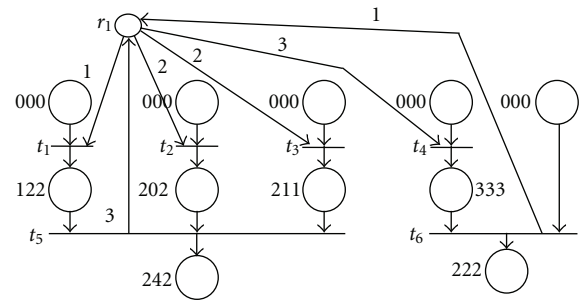

Figure 2: Token flow relation for resource 1.

places of P_F provide input to t_F, t_F is the only input of p_0, and p_0 is the only output of t_F. Also, p_0 is the only input of t_I, and t_I is the only output of p_0. Finally, t_I is the only input of places in P_I, and these places connect to transitions in T_S.

Since the process net (without resource places) is a marked graph, each place in $\{p_0\} \cup P_I \cup P_S \cup P_F$ has exactly one input and one output. This implies that processes can exhibit concurrency and synchronization but not choice. To be well-defined, we require that the process net be strongly connected. Finally, we will say that $P = \{p_0\} \cup P_I \cup P_S \cup P_F$, $T = \{t_I, t_F\} \cup T_S$, $N = P \cup T$ and $N_R = (P \cup P_R) \cup T$. To summarize, we have the following notation.

p_0: Initial process place. The initial marking of p_0 specifies the maximum number of concurrently executing processes.

P_I: Places that hold subprocesses ready to begin processing.

P_S: Places where processing occurs. These typically have associated resource needs.

P_F: Places holding the completed process.

P_R: The set of resource places.

P: $\{p_0\} \cup P_I \cup P_S \cup P_F$, all places except resource places.

t_I: The "order release" transition.

T_S: Transitions that allocate-deallocate resources and that synchronize, merge, or split subprocesses.

t_F: The "process completion" transition.

T: $\{t_I, t_F\} \cup T_S$ the set of transitions.

$W(r,t)$: The number of units of resource r requested at transition t.

N: $P \cup T$, the process net without resources.

N_R: $(P \cup P_R) \cup T$, the process net with resources.

As previously stated, the Appendix (Definition A.11) provides the formal definition.

As mentioned in the introduction, assessing the quasi-liveness of the G-AMG is NP-complete [10]. Thus, determining whether or not a given process has a sequence of transition firings (resource allocations) that enables t_F requires super-polynomial computation in the general case. Detailed discussions on quasi-liveness and related issues for the general case can be found in [12, 14].

In this paper, we investigate live resource allocation for assembly systems only; that is, we impose that for all $t \in T_S \subseteq N_R$, $t \bullet \cap P_S$ is a singleton. In Section 3, we develop conditions that provide quickly computable tests on quasi-liveness. In Section 4, we develop polynomial methods for resolving quasi-liveness and generating feasible resource allocation sequences for assembly systems with single unit resource allocation.

3. The G-AMG$_A$ Model for the A-RAS

This section develops results for the subclass, referred to as G-AMG$_A$, of A/D-RAS systems restricted to assembly only (A-RAS). In other words, systems in G-AMG$_A$ $\subseteq$ G-AMG have subprocess merging but no splitting. For this subclass of systems, N is restricted as follows: for all $t \in T_S$, $|t \bullet \cap P_S| = 1$. Thus, a transition (other than t_I) can perform no splitting operation; that is, there is no disassembly. For this subclass, we develop a set of quick tests for quasi-liveness based on necessary conditions and sufficient conditions. The necessary conditions are based on local tests of "place concurrence" for each synchronizing transition. If these conditions are not met, then the net is not quasi-live. If these tests do not indicate lack of quasi-liveness, we then perform a polynomial sufficiency test, that, if met, guarantees quasi-liveness and provides resource enabled execution sequences.

3.1. Necessary Conditions for A-RAS. Consider an N_R. Let T_{Synch} be the set of transitions that synchronize subprocesses, that is, $T_{\text{Synch}} = \{t \in T_S : |\bullet t \cap P_S| > 1\}$. For example, in Figure 1, $T_{\text{Synch}} = \{t_5, t_6, t_7\}$. We note that for each $t \in T_{\text{Synch}}$, all places in $\bullet t \cap P_S$ must be simultaneously marked for synchronization to occur. Further, there must exist sufficient remaining unallocated resources to fire the synchronization once these places are marked. For example, in Figure 1, for t_5 to be process enabled, it is necessary that the three subprocesses synchronized at t_5 are simultaneously allocated a total of three units of resource type, r_2. To resource enable t_5, one additional unit of r_2 is required. Thus, if the capacity of r_2 is less than three, t_5 cannot be process enabled, and if the capacity of r_2 is less than four, t_5 cannot be both process and resource enabled. Thus, as illustrated by this example, if there exists $t \in T_{\text{Synch}}$ and resource, r_i, such that $W(r_i,t) + \sum_{p \in \bullet t \cap P_S} u_i(p) > C_i$, where C_i is the capacity of r_i, then N_R cannot be quasi-live.

This is our first necessary condition that resource capacities must be sufficient to be both process enabled and resource enabled $t \in T_{\text{Synch}}$.

Further, note that transitions t_5 and t_6 must be fired to process-enable t_7. Since we fire only one transition at a time, these must be fired in some order. Suppose t_6 is fired before t_5. Then the subprocess at place $t_6 \bullet \cap P_S$ will be assembled and holding two units of r_3 after firing t_6. Then to fire t_5, subprocesses at $\bullet t_5 \cap P_S$ will need to be holding five units of r_3. Thus, r_3 must have at least seven units of capacity if t_6 fires before t_5.

On the other hand, if t_5 is fired before t_6, then the subprocess at place $t_5 \bullet \cap P_S$ will be assembled and holding two units of r_3. Then to fire t_6, the subprocess at $\bullet t_6 \cap P_S$ will need to be holding three units of r_3. Thus, r_3 must have at least five units of capacity if t_5 fires before t_6. Clearly, if r_3 has capacity four, the net is not quasi-live. If r_3 has capacity five or six, $t_5 t_6$ is resource enabled but $t_6 t_5$ is not. If r_3 has capacity seven or greater, both sequences are resource enabled.

More generally, suppose $t \in T_{\text{Synch}}$ has K subprocess input places, that is, $K = |\bullet t \cap P_S|$. Since N is a marked graph, each $p \in \bullet t \cap P_S$ will have only one input transition. Since N is assembly only, each $p \in \bullet t \cap P_S$ will have a unique input transition. Thus, to process enable t, these K transitions will have to be fired in some order.

Let $\bullet t \cap P_S = \{p_{(1)}, p_{(2)}, \ldots, p_{(K)}\}$ and let $\bullet \bullet t = \{t_{(1)}, t_{(2)}, \ldots, t_{(K)}\}$, where $\bullet p_{(1)} = \{t_{(1)}\}$, $\bullet p_{(2)} = \{t_{(2)}\}$, and so forth. Firing $t_{(1)}$ marks $p_{(1)}$, firing $t_{(2)}$ marks $p_{(2)}$, and so forth. When $\{p_{(1)}, p_{(2)}, \ldots, p_{(K)}\}$ are all marked, t is process enabled. With unlimited resources, there are $K!$ possible firing sequences for $\{t_{(1)}, t_{(2)}, \ldots, t_{(K)}\}$ that process-enable t (assuming each is fired only once). However, with finite resource capacities, some (possibly all) of the firing sequences might be infeasible. For example, in Figure 1, if r_3 has capacity six, then the firing sequence $t_6 t_5$ is not possible, although $t_5 t_6$ is.

Let σ_k be the set of partial firing sequences of $\{t_{(1)}, t_{(2)}, \ldots, t_{(K)}\}$ of length $k \leq K$ (again assuming that each transition will occur at most once in any sequence of σ_k). Note that $\sigma \in \sigma_k$ marks k places of $\bullet t \cap P_S$ and leaves $K - k$ unmarked. If there exists $k < K$ such that for every marked k-subset of $\{p_{(1)}, p_{(2)}, \ldots, p_{(K)}\}$, all input transitions to the unmarked $(K-k)$-complement are resource disabled, then N_R cannot be quasi-live.

Putting more formally, let S^k be a k-subset of $\{p_{(1)}, p_{(2)}, \ldots, p_{(K)}\}$, that is, $S^k \subseteq \{p_{(1)}, p_{(2)}, \ldots, p_{(K)}\}$ such that $|S^k| = k$. Note that there are $\binom{|\bullet t \cap P_S|}{k}$ total k-subsets of $\bullet t \cap P_S = \{p_{(1)}, p_{(2)}, \ldots, p_{(K)}\}$. If $\exists k < K$ such that for all $S^k \subseteq \bullet t \cap P_S$, for all $p \in (S^k)^c = (\bullet t \cap P_S) \setminus S^k$, $\exists r_i$ such that $\sum_{p \in S^k} u_i(p) + W(r_i, \bullet p) > C_i$, then N_R cannot be quasi-live.

This is our second necessary condition that resource capacities must be sufficient to fire all the input transitions to subprocess input places of $t \in T_{\text{Synch}}$.

Algorithm 1 checks these necessary conditions. The algorithm starts with a For loop that tests every synchronization

For every $t \in T_{\text{Synch}}$
  //Check for violations of the first necessary condition
  Find r_i such that $W(r_i, t) + \sum_{p \in \bullet t \cap P_S} u_i(p) > C_i$
  If successful, return Not Quasi-live
  //Check for violations of the second necessary condition
  Else $k = 1$
    While $k < K = |\bullet t \cap P_S|$
      subset_count = 0
      For each $S^k \subseteq \bullet t \cap P_S$
        place_count = 0
        For each $p \in (S^k)^c$
          Find r_i st $\sum_{p \in S^k} u_i(p) + W(r_i, \bullet p) > C_i$
          If successful, place_count++
        End For
        If place_count = $|(S^k)^c|$, subset_counter++
      End For
      If subset_count = $\binom{|\bullet t \cap P_S|}{k}$, return Not Quasi-live
      k++
    End While
  End For
  Return Unknown

Algorithm 1

transition for violations of the two necessary conditions. The first check is for necessary condition 1, where the resources required to process-enable plus the resources required to resource-enable the synchronization are compared to the resource capacities. If a violation is found, then the net cannot be quasi-live, and the algorithm terminates by returning not quasi-live.

If no violation of the first necessary condition is found, then the algorithm initiates a While loop for testing the second condition. The first step is to initialize a subset counter, which, for the given synchronization transition, counts the number of k-subsets of the process input places that violate the second necessary condition. If it found that all k-subsets violate the second necessary condition, that is, subset_count = total number of k-subsets, then the algorithm terminates by returning not quasi-live.

Note that the inner For loop determines whether a given k-subset violates the second necessary condition or not. It does this by checking all the places in the $(K\text{-}k)$-complement to see if their input transitions are resource enabled. If none is, then none of these places can be marked, and the synchronization cannot be process enabled by first marking the k-subset and then firing the input transitions of the $(K\text{-}k)$-complement. If this is true for a k-subset, then that k-subset violates the second condition and the counter, subset_count, is incremented. Again, if we find $k < K$ such that all k-subsets violate the second necessary condition, that is, subset_count = total number of k-subsets, then the algorithm terminates by returning not quasi-live.

Consider an example assembly system depicted in Figure 3. Assume that $\langle C_1, C_2, C_2 \rangle = \langle 2, 4, 1 \rangle$. We have both t_{10} and t_{12} in T_{Synch}. Checking t_{10} for the first necessary condition, we have $0 + (0 + 0 + 1) = 1 \leq 2 = C_1$ for r_1, $2 + (1 + 0 + 0) = 3 \leq 4 = C_2$ for r_2, and $0 + (0 + 1 + 0) = 1 \leq 1 = C_3$ for r_3, resulting in no violation. A similar check finds that t_{12} does not violate the first necessary condition. We now check t_{10} for the second necessary condition. For $S^1 = \{p_3\}$, we have $(0) + 0 = 0 \leq 2 = C_1$ and $(0) + 1 = 1 \leq 2 = C_1$ for r_1, $(1) + 0 = 1 \leq 4 = C_2$ and $(1) + 0 = 1 \leq 4 = C_2$ for r_2, and $(0) + 1 = 1 \leq 1 = C_3$ and $(0) + 0 = 0 \leq 1 = C_3$ for $r3$. Hence, there is no violation. Similar checks for $S^1 = \{p_6\}$ and $S^1 = \{p_9\}$ reveal that there is no violation. For $S^2 = \{p_3, p_6\}$, we have $(0 + 0) + 1 = 1 \leq 2 = C_1$ for r_1, $(1 + 0) + 0 = 1 \leq 4 = C_2$ for r_2, and $(0 + 1) + 0 = 1 \leq 1 = C_3$ for r_3; as a result, the condition is not violated. Similarly, $S^2 = \{p_3, p_9\}$ and $S^2 = \{p_6, p_9\}$ yield no violation. Therefore, t_{10} does not violate the second necessary condition. Likewise, t_{12} does not violate the second necessary condition either.

We note that Algorithm 1 enumerates all subsets of the input places for each synchronization transition, and thus, in the strictest sense, this check is of exponential complexity. However, we expect that the number of subprocesses combined at any synchronization will be sufficiently small so that the total computation of Algorithm 1 will be quite small in comparison to the complete enumeration of the reachability graph in [12, 14], and therefore the check is worthwhile.

If no violations of either of the necessary conditions are found, then the quasi-liveness remains unverified that is, we cannot say whether the net is quasi-live or not. In the following section, we will develop a sufficient condition for quasi-liveness and an algorithm, for generating a process completing sequence based on this sufficient condition.

3.2. Sufficient Condition Test for Quasi-Liveness of the A-RAS. This subsection develops a sufficiency test for the G-AMG$_A$ model. This test makes use of reductions performed on two types of structures contained in the G-AMG$_A$. In Figure 3, consider the three net segments: $\{t_I, p_1, t_1, p_2, t_2, p_3, t_{10}\}$, $\{t_I, p_4, t_4, p_5, t_5, p_6, t_{10}\}$, and $\{t_I, p_7, t_7, p_8, t_8, p_9, t_{10}\}$. These three represent the sequential processing steps of the three subprocesses marking places $\{p_1, p_4, p_7\} \subseteq P_I$ that synchronize at t_{10}. Any interaction between the three subprocesses is strictly limited to resource competition. Otherwise their processing up to t_{10} is independent, possibly concurrent, depending on resource capacities.

We note the following.

(1) The subprocess of p_1 requires a total allocation of $\langle 1, 1, 0 \rangle$ in order to reach p_3, where it will release the unit of r_1 and will hold the unit of r_2.

(2) The subprocess of p_4 requires a total allocation of $\langle 0, 1, 1 \rangle$ in order to reach p_6, where it will release the unit of r_2 and will hold the unit of r_3.

(3) The subprocess of p_7 requires a total allocation of $\langle 1, 0, 1 \rangle$ in order to reach p_9, where it will release the unit of r_3 and will hold the unit of r_1.

Thus, if we have sufficient resources to simultaneously allocate $\langle 1, 1, 0 \rangle$ to the first subprocess, $\langle 0, 1, 1 \rangle$ to the second, and $\langle 1, 0, 1 \rangle$ to the third, then we are sure that the three subprocesses can reach the synchronization stage.

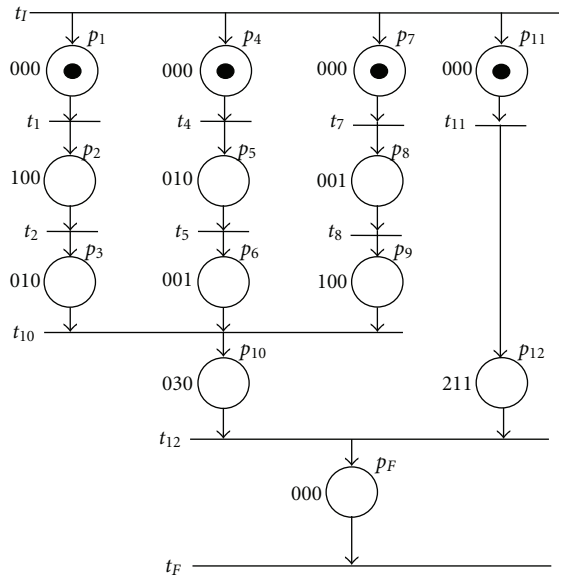

Figure 3: Example assembly system.

Thus, we say that if $\langle C_1, C_2, C_2\rangle \geq \langle 1,1,0\rangle + \langle 0,1,1\rangle + \langle 1,0,1\rangle = \langle 2,2,2\rangle$, then resource capacities are sufficient to process-enable the synchronization at t_{10}.

We refer to a structure such as $\{t_I, p_1, t_1, p_2, t_2, p_3, t_{10}\}$ as a Type-I structure; that is, a Type-I structure is a segment $\langle t_I, p_{(1)}, t_{(1)}, p_{(2)}, t_{(2)}, \ldots, t_{(k-1)}, p_{(k)}, t\rangle$ of N, where

(1) $p_{(1)} \in t_I\bullet$,

(2) $\{p_{(1)}, \ldots, p_{(k)}\} \subseteq P$ (recall, $P = \{p_0\} \cup P_I \cup P_S \cup P_F$),

(3) $\bullet t_{(j)} \cap P = \{p_{(j)}\}$,

(4) $t \in T_{\text{Synch}}$,

(5) $k > 2$.

The first condition states that $p_{(1)}$ is an output place of t_I; the second states that all places are nonresource places; the third states that none of the intermediate transitions are synchronizations; the fourth states that the last transition is a synchronization; and the last states there are at least three places in the structure.

Thus, a Type-I structure of N is a path in N with at least three places that begins with t_I, ends with a synchronization, and has the property that all intermediate transitions are not synchronizations.

Now, consider the Type-I structure, $\{t_I, p_1, t_1, p_2, t_2, p_3, t_{10}\}$, in Figure 3. Suppose we reduce it as follows:

(1) delete p_2 and t_2 and all corresponding edges;

(2) insert edge $\langle t_1, p_3\rangle$;

(3) set the resource need vector associated with p_3 to the component-wise maximum of the need vectors of all places in the Type-I structure, that is, the component-wise maximum of $\{\langle 0,0,0\rangle; \langle 1,0,0\rangle; \langle 0,10\rangle\} = \langle 1,1,0\rangle$.

Applying this reduction to the three Type-I structures in the example yields the resulting net shown in Figure 4. Note that the net now contains no Type-I structure.

More formally, let ρ_1 represent a Type-I reduction on net N, and let $\rho_1(N)$ be the resulting net. Then ρ_1 applies the following actions to N.

Resource Bound Update. For each Type-I structure $\langle t_I, p_{(1)}, t_{(1)}, \ldots, t_{(k-1)}, p_{(k)}, t\rangle$, assign $\Psi_i(p_{(k)}) = \max\{u_i(p_{(j)}) : j = 1 \ldots k\}, i = 1 \ldots |P_R|$, and let Ψ_k denote the vector $\langle \Psi_i(p_k) : i = 1 \ldots |P_R|\rangle$.

Net Reduction. Delete $\{p_{(2)}, t_{(2)}, \ldots, p_{(k-1)}, t_{(k-1)}\}$ and the associated arcs. Add arc $(t_{(1)}, p_{(k)})$.

Note that $\Psi_i(p_{(k)})$ retains the maximum usage of resource, r_i, along the Type-I structure. Thus, the resource bound associated with the undeleted place, $p_{(k)}$, will be the number of units of each resource required for the subprocess to reach the synchronization transition.

We note that all Type-I structures can be found in number of steps polynomial in places and transitions. We now proceed to our second reduction.

Now consider net segments $\{\langle t_I, p_1, t_1, p_3, t_{10}\rangle, \langle t_I, p_4, t_4, p_6, t_{10}\rangle, \langle t_I, p_7, t_7, p_9, t_{10}\rangle\}$ of Figure 4. We refer to this structure as a Type-II structure, that is, a set of at least two parallel segments, starting at t_I, with two intermediate places, and ending at $t \in T_{\text{Synch}}$.

More formally, a Type-II structure is composed of $m > 1$ parallel segments in N ending in $t \in T_{\text{Synch}}$:

1. $\langle t_I, p_{(11)}, t_{(11)}, p_{(12)}, t\rangle$
2. $\langle t_I, p_{(21)}, t_{(21)}, p_{(22)}, t\rangle$.

$\vdots$ $\vdots$

m. $\langle t_I, p_{(m1)}, t_{(m1)}, p_{(m2)}, t\rangle$

such that $\{p_{(11)}, \ldots, p_{(m1)}\} \subseteq t_I\bullet$ and $p_{(i1)} \neq p_{(j1)}$ for $i \neq j$.

A Type-II reduction, ρ_2, is similar to the Type-I reduction in that it applies a bound update and then a net reduction. We first illustrate the bound update and reduction and then state it more formally.

To understand the next bound update, consider the nets of Figure 5. Each place in (a) is labeled with resource need. To mark p_3, we require $\langle 121\rangle$ units for resources r_1, r_2 and r_3, thus, in (b), $\Psi_3 = \langle 121\rangle$. Similarly, $\Psi_6 = \langle 223\rangle$ and $\Psi_9 = \langle 412\rangle$ for places p_6 and p_9, respectively, as shown in (b).

Places in (b) are also labeled with their original resource needs, u_3, u_6, and u_9. Now, for p_3, p_6, and p_9, consider $\delta_i(p_k) = \Psi_i(p_k) - u_i(p_k)$. We refer to $\delta_i(p_k)$ as the "return" of resource, r_i, by the corresponding subprocess. Letting let δ_k denote the vector $\langle \delta_i(p_k): i = 1 \cdots |P_R|\rangle$, we have $\delta_3 = \langle 012\rangle$, $\delta_6 = \langle 213\rangle$, and $\delta_9 = \langle 410\rangle$, as shown in Figure 5(b).

Sort the places $\{p_3, p_6, p_9\}$ by decreasing return for r_1. Then we have ordered set $\langle p_9, p_6, p_3\rangle$ since $4 \geq 2 \geq 0$. In 5(a), if we first mark p_9, then p_6, and finally p_3 according to the firing sequence $\sigma = t_5t_6t_3t_4t_1t_2$, the following capacity constraints must be met (note that C is the resource capacity vector):

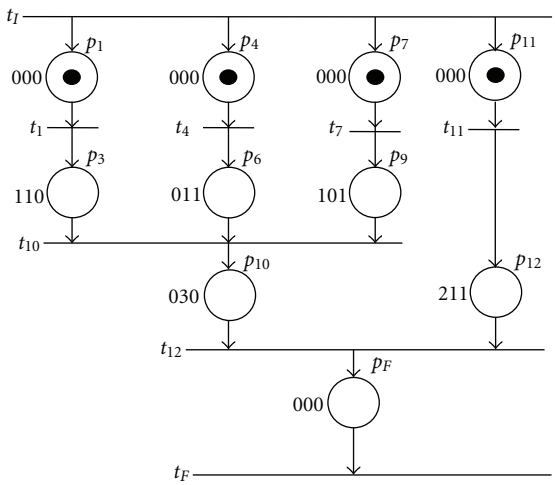

Figure 4: Resulting net with Type-I reductions.

$\Psi_9 = \langle 412 \rangle \leq C$,
$\Psi_6 + u_9 = \langle 223 \rangle + \langle 002 \rangle = \langle 225 \rangle \leq C$,
$\Psi_3 + u_9 + u_6 = \langle 121 \rangle + \langle 002 \rangle + \langle 010 \rangle = \langle 133 \rangle \leq C$.

Taking the component-wise max across these constraints yields $\langle 435 \rangle \leq C$. Thus, $\langle 435 \rangle$ is necessary and sufficient to execute $\sigma = t_5 t_6 t_3 t_4 t_1 t_2$. We will refer to σ as a "serialized" firing sequence, since it advances the Type-I subprocesses to the synchronization transition one at a time. In other words, a serialized firing sequence does not allow parallel Type-I subprocesses to process in parallel. We refer to the computed bounds as serialized bounds.

Note that if we sort $\{p_3, p_6, p_9\}$ in any other way, say $\langle p_6, p_3, p_9 \rangle$, we get a different serialized firing sequence for marking the places and a different set of resource bounds (in this case, $t_3 t_4 t_1 t_2 t_5 t_6$ and $\langle 533 \rangle$, resp.). The bound for r_1 can be no smaller, although the bounds for r_2 and r_3 might be tighter. This is established by the following lemma.

Lemma 1. *Let p_j and p_k be two places in a Type-II structure, where $\delta_i(p_j)$ and $\delta_i(p_k)$ are the returns of resource r_i for p_j and p_k. If $\delta_i(p_j) \geq \delta_i(p_k)$, then* $\max(\Psi_i(p_k), \Psi_i(p_j) + u_i(p_k)) \geq \max(\Psi_i(p_j), \Psi_i(p_k) + u_i(p_j))$.

Before going to the proof, note (recall) the following:

(1) $u_i(p_j)$ is the need (number of units held) of r_i at p_j;

(2) $\Psi_i(p_k)$ is the maximum need for r_i along the Type-I structure leading to p_k;

(3) given that the jth subprocess has advanced to p_j, $\Psi_i(p_k) + u_i(p_j)$ is a lower bound on the number of units of r_i required to advance the kth subprocess from its place in P_I to p_k;

(4) given that the jth and kth subprocesses are both at their initial places in P_I, $\max(\Psi_i(p_j), \Psi_i(p_k) + u_i(p_j))$ is a lower bound on the number of units of r_i required to first advance the jth subprocess to p_j and then the kth subprocess to p_k.

Proof. By assumption $\delta_i(p_j) \geq \delta_i(p_k)$. Further, $\Psi_i(p_j) \geq \delta_i(p_j)$, since the jth subprocess cannot return more of r_i than it is allocated.

Then, $\Psi_i(p_j) - \delta_i(p_k) \geq 0$;
$\Psi_i(p_j) - \Psi_i(p_k) + u_i(p_k) \geq 0$;
$\Psi_i(p_j) + u_i(p_k) \geq \Psi_i(p_k)$.
Thus, $\max(\Psi_i(p_k), \Psi_i(p_j) + u_i(p_k)) = \Psi_i(p_j) + u_i(p_k)$.
Now, since $u_i(p_k) \geq 0$, we have $\Psi_i(p_j) + u_i(p_k) \geq \Psi_i(p_j)$;
and $\Psi_i(p_j) + u_i(p_k) = \Psi_i(p_j) + \Psi_i(p_k) - \delta_i(p_k)$;
$\Psi_i(p_j) + u_i(p_k) \geq \Psi_i(p_j) + \Psi_i(p_k) - \delta_i(p_j)$;
(since $\delta_i(p_j) \geq \delta_i(p_k)$)
$\Psi_i(p_j) + u_i(p_k) \geq \Psi_i(p_k) + u_i(p_j)$.
Thus, $\max(\Psi_i(p_k), \Psi_i(p_j) + u_i(p_k)) \geq \max(\Psi_i(p_j), \Psi_i(p_k) + u_i(p_j))$. □

The point is to show that if we advance the subprocesses serially; that is, one at a time, from their places in P_I to their synchronization transition, in order of decreasing return of r_i, then we will minimize the need for r_i in the serial advancement.

We can now formally state the bound update and net reduction. To understand the subscripts, please refer to the definition of a Type-II structure given above. Our approach is to identify a critical resource, r_c, perhaps one that is most constraining or most expensive, and compute bounds for Type-II reductions using the returns for r_c as a sorting key in ordering the corresponding subprocesses.

Resource Bound Update for Critical Resource, r_c

For a Type-II structure
  Let $\delta_i(p_{(j2)}) = \Psi_i(p_{(j2)}) - u_i(p_{(j2)})$, $j = 1, \ldots, m$, $i = 1, \ldots, |P_R|$
  Sort $\{p_{(12)} \cdots p_{(m2)}\}$ by decreasing $\delta_c(p_{(j2)})$
  Let $\Gamma = \langle p^1, \ldots, p^m \rangle$ be the sorted set
  For $i = 1, \ldots, |P_R|$
    Set $\Psi_i(p_{(12)})$ to $\max\{\Psi_i(p^t) + \sum_{j=1}^{t-1} u_i(p^j) : t = 1, \ldots, m\}$
  End For
End For

Net Reduction. Delete $\{p_{(21)}, t_{(21)}, p_{(22)}; \ldots; p_{(m1)}, t_{(m1)}, p_{(m2)}\}$ and the associated arcs.

Subsequently, let $\rho_2(N)$ denote the net resulting from a Type-II reduction having been applied to N; that is, in $\rho_2(N)$ all Type-II structures have been reduced. Clearly, all Type-II structures in a net can be found in number of steps polynomial in places and transitions.

Let us now apply a Type-II reduction to the net of Figure 4. Assuming that r_1 is the critical resource, we obtain the resulting net depicted in Figure 6 (note that a new Type I structure has emerged).

Lemma 2 guarantees the computed bounds are sufficient for some serialized firing sequence.

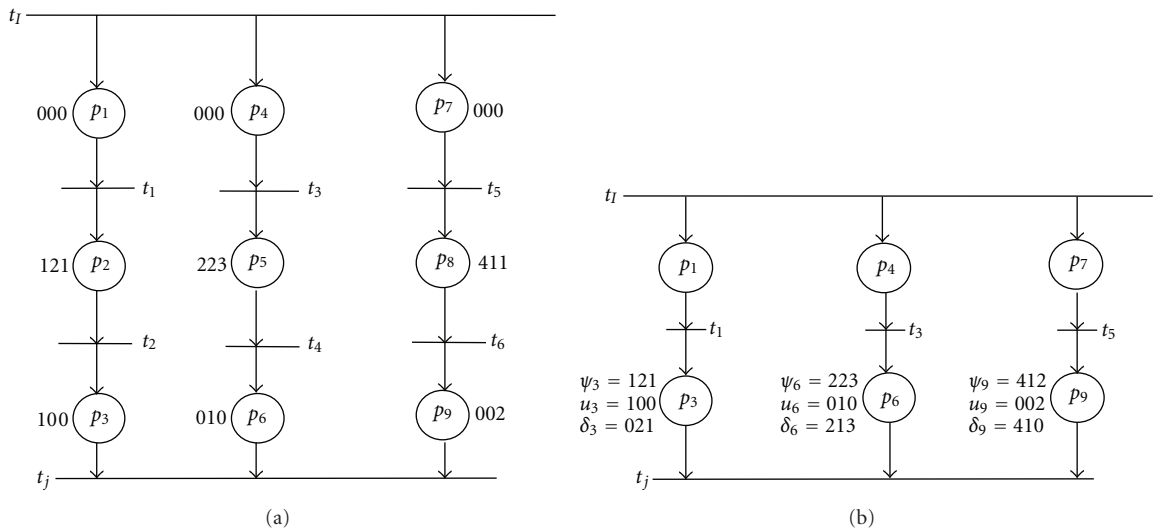

Figure 5: Example for Type-II bound update.

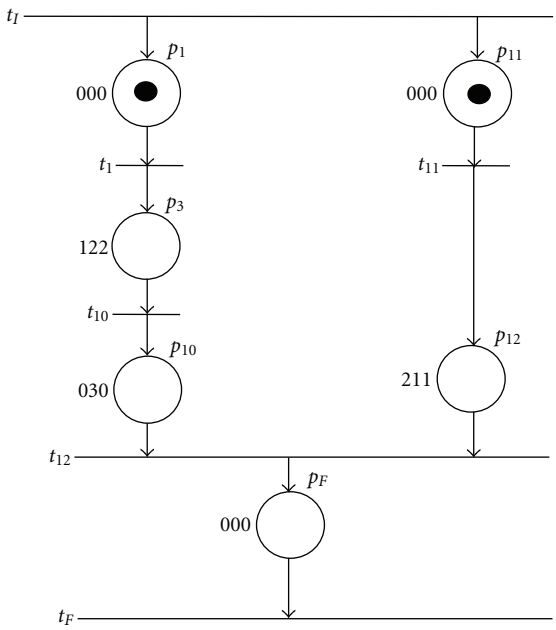

Figure 6: Resulting net with a Type-II reduction.

Lemma 2. *Suppose $N \in$ G-AMG$_A$ with m Type-I structures connecting t_I and $t_j \in T_{Synch}$, $\langle t_I, p_{(11)}, t_{(11)} \dots p_{(1k)}, t_j\rangle$, $\langle t_I, p_{(21)}, t_{(21)} \dots p_{(2n)}, t_j\rangle, \dots$, and $\langle t_I, p_{(m1)}, t_{(m1)} \dots p_{(mp)}, t_j\rangle$. Then $\Psi_{(1k)} = \{\Psi_h(p_{(1k)}) : h = 1, \dots, |P_R|\}$ in $\rho_2\rho_1(N)$ is a sufficient resource level to enable a firing sequence of N that marks $\{p_{(1k)}, p_{(2n)}, \dots, p_{(mp)}\}$.*

Proof. It is clear that $\Psi_{(1k)}$ in $\rho_1(N)$ enables $\sigma_1 = t_{(11)}t_{(12)} \dots t_{(1,k-1)}$ in N, $\Psi_{(2n)}$ in $\rho_1(N)$ enables $\sigma_2 = t_{(21)}t_{(22)} \dots t_{(2,n-1)}$ in N, and so forth. Now $\rho_1(N)$ will contain Type-II structure $\{\langle t_I, p_{(11)}, t_{(11)}, p_{(1k)}, t_j\rangle, \langle t_I, p_{(21)}, t_{(21)}, p_{(2n)}, t_j\rangle, \dots, \langle t_I, p_{(m1)}, t_{(m1)}, p_{(mp)}, t_j\rangle\}$. Before doing the Type-II reduction, we sort $\{p_{(1k)}, p_{(2n)}, \dots, p_{(mp)}\}$ based on the return of critical resource, r_c (perhaps arbitrarily chosen), $\delta_c(p) = \Psi_c(p) - u_c(p)$, and let $\langle p^1 p^2, \dots, p^m\rangle$ be the sorted set, in order of decreasing return. Then, if the resource capacities satisfy the following constraint set: $\{\Psi^1 \le C, \Psi^2 + u^1 \le C, \Psi^3 + u^1 + u^2 \le C, \dots, \Psi^m + u^1 + \cdots + u^{m-1} \le C\}$ in N, we can first fire σ^1 and mark p^1, next fire σ^2 and mark p^2, and so forth. Thus, by updating $\Psi_{(1k)}$ with the component-wise maximum of $\{\Psi^1 \le C, \Psi^2 + u^1 \le C, \Psi^3 + u^1 + u^2 \le C, \dots, \Psi^m + u^1 + \cdots + u^{m-1} \le C\}$ before the Type-II reduction, we assure that $\Psi_{(1k)}$ in $\rho_2\rho_1(N)$ is a sufficient resource level to enable the firing sequence $\sigma^1\sigma^2, \dots, \sigma^m$ in N. □

We will now establish some necessary properties for these reductions. We note that the reductions are defined on N and not on $N_R \in$ G-AMG$_A$. For the sake of brevity, we will use the notation "$N \in$ G-AMG$_A$ implies $\rho(N) \in$ G-AMG$_A$" to indicate that a reduction preserves the class defining structure of the process flow. Note that in the strictest sense, if $N_R \in$ G-AMG$_A$ and N is the corresponding process subnet, then $N \in$ G-AMG$_A$, since it represents a valid process flow with no resource requirements.

Lemma 3. *$N \in$ G-AMG$_A$ implies $\rho_1(N) \in$ G-AMG$_A$.*

Proof. Suppose N has no Type-I structure. Then, $\rho_1(N) = N_i$ and hence $N \in$ G-AMG$_A$. Suppose N has a Type-I structure $\langle t_I, p_{(1)}, t_{(1)}, p_{(2)}, \dots, t_{(k-1)}, p_{(k)}, t_{(k)}\rangle$. In $\rho_1(N)$, this structure is transformed to $\langle t_I, p_{(1)}, t_{(1)}, p_{(k)}, t_{(k)}\rangle$. Since $\{p_{(1)}, t_{(1)}, p_{(2)}, \dots, t_{(k-1)}, p_{(k)}\}$ are connected to the rest of N through t_I and $t_{(k)}$ only, the reduction is local and all other places, transitions, and arcs remain intact. Thus $N \in$ G-AMG$_A$ implies $\rho_1(N) \in$ G-AMG$_A$. □

Lemma 4. *$N \in$ G-AMG$_A$ implies $\rho_2(N) \in$ G-AMG$_A$.*

Proof. Suppose N has no Type-II structure. Then, $\rho_2(N) = N$ and thus $N \in$ G-AMG$_A$. Suppose N has a Type-II structure $\{\langle t_I, p_{(11)}, t_{(11)}, p_{(12)}, t_j\rangle, \langle t_I, p_{(21)}, t_{(21)}, p_{(22)}, t_j\rangle \cdots \langle t_I, p_{(m1)}, t_{(m1)}, p_{(m2)}, tj\rangle\}$. In $\rho_2(N)$, the m parallel sequences are transformed into the single sequence, $\langle t_I, p_{(11)}, t_{(11)}, p_{(12)}, t_j\rangle$. As before, all other places and transitions remain intact, and thus $N \in$ G-AMG$_A$ implies $\rho_2(N) \in$ G-AMG$_A$. □

The above two lemmas establish that ρ_i: G-AMG$_A$ → G-AMG$_A$, $i = 1, 2$. Note that for any $N \in$ G-AMG$_A$, $p_0 \in P$ and $P_I \cup P_s \cup P_F \neq \emptyset$, $\{t_I, t_F\} \subseteq T$, $\{(p_0, t_I), (t_F, p_0)\} \subseteq W$, $t_I \bullet \neq \emptyset$, $\bullet t_F \neq \emptyset$, and there is a path from t_I to t_F. Let $\aleph = \{\{p_0, p_{(1)}\}, \{t_I, t_F\}, \{(p_0, t_I), (t_I, p_{(1)}), (p_{(1)}, t_F), (t_F, p_0)\}, \{1, 0\}\}$. It is clear that $\aleph \in$ G-AMG$_A$ and that ρ_i will not affect $\aleph$, since $\aleph$ has no Type-1 or Type-2 structure. We refer to $\aleph$ as "irreducible."

Lemma 5. *If $N \in$ G-AMG$_A$ and $N \neq \aleph$, then there exists a Type-I or Type-II structure in N.*

Proof. Suppose that N is not irreducible. Then $T_S \neq \emptyset$. Suppose that there exists neither Type-I structure nor Type-II structure. Then, since no Type-I structure exists, every $t_u \in T_S$ is a synchronization. This implies that $|t_I \bullet| > 1$, otherwise there are no subprocesses to synchronize. Since no Type-II structure exists, for every pair $(p_j, p_k) \subseteq t_I\bullet$, $p_j \bullet \neq p_k\bullet$. This implies that for every $p_j \in t_I\bullet$, $\exists\, p_u \notin t_I\bullet$ such that p_j and p_u synchronize at $p_j\bullet$. Note that there must be a path from t_I to p_u, and the first node of this path, say p_v, must be in $t_I\bullet$. Further, the synchronization transition, $p_v\bullet$, must fire before p_u can be marked. Thus, for every $p_j \in t_I\bullet$ there exists $p_v \in t_I\bullet$ such that $p_v\bullet$ must be enabled and fired before $p_j\bullet$ can be enabled and fired. Since $t_I\bullet$ is finite, this implies a cyclic dependency among the transitions of $t_I \bullet \bullet$, which contradicts the implication of Definition A.11 that every cycle of N passes through p_0. □

With these results, the following theorems are now straightforward.

Theorem 1. *For every $N \in$ G-AMG$_A$, there is a finite sequence of reductions that maps N to irreducible form. Further, sequence length is $O(|P_S|)$.*

Proof. Suppose $N \in$ G-AMG$_A$ is not in irreducible form. Then, it can be reduced by the following algorithm, which will return the required sequence of reductions:

Set $\eta = N$, $\rho = \varepsilon$ (empty string)
While $\eta \neq \aleph$
  $\eta = \rho_2(\rho_1(\eta))$
  $\rho = \rho_2\rho_1\rho$ (concatenation)
End While
Return ρ

Note that if η is not irreducible, then $\rho_2(\rho_1(\eta))$ has fewer places than η. Since N has finite places, the While will terminate in a finite number of steps not larger than $|P_S|$ since each iteration will eliminate at least one place. □

In the following, we will let $(\rho_2\rho_1)^n(N)$ denote the net that results after the Type-I/Type-II reduction sequence has been applied n times.

Theorem 2. *For every $N \in$ G-AMG$_A$, let $\eta = (\rho_2\rho_1)^n(N)$ and suppose that p_j has survived at least one update to Ψ_j without being deleted. Then Ψ_j is sufficient to enable a firing sequence in N that enables $p_j\bullet$.*

Proof. Suppose p_j has been involved in Type-I and Type-II structures over the n reductions and is the surviving place of those reduced structures. By the induction hypothesis, Ψ_j is sufficient to enable a firing sequence in N that enables $t_j = p_j\bullet$, say σ. Note that in η, $\bullet p_j = \{t_I\}$ and p_j is in a Type-I structure (assuming $t_j \neq t_F$), since it is the lone input to t_j and $t_j \bullet \neq \emptyset$. (To see this, recall that since ρ_2 is performed after ρ_1, η has no Type-II structures.) Let $\langle t_I, p_j, t_j, p_{(2)}, t_{(2)} \ldots t_{(k-1)}, p_{(k)}, t_{(k)}\rangle$ be this Type-I structure in η. On subsequent Type-I reduction, $\Psi_h(p_{(k)}) = \max\{\Psi_h(p_j), \Psi_h(p_{(2)}), \ldots, \Psi_h(p_{(k)})\}$, for $h = 1, \ldots, |P_R|$, then $\{p_{(2)}, \ldots, p_{i(k-1)}\}$ will be deleted, along with corresponding arcs and arc $(t_j, p_{(k)})$ will be added. Thus, $\Psi_{(k)}$ will be sufficient for firing sequence $\sigma\tau = \sigma t_j t_{(2)} \ldots t_{(k-1)}$, which marks $p_{(k)}$.

Now, consider $\rho_1(\eta)$ with a Type-II structure $\{\langle t_I, p_{(11)}, t_{(11)}, p_{(12)}, t_j\rangle, \langle t_I, p_{(21)}, t_{(21)}, p_{(22)}, t_j\rangle \cdots \langle t_I, p_{(m1)}, t_{(m1)}, p_{(m2)}, t_j\rangle\}$, where $\sigma\tau_{(1)}$ marks $p_{(12)}$, $\sigma\tau_{(2)}$ marks $p_{(22)}$,..., and $\sigma\tau_{(m)}$ marks $p_{(m2)}$. Suppose we order $\{p_{(12)}, p_{(22)}, \ldots, p_{(m2)}\}$ by decreasing return, $\delta_j = \Psi_j - u_j$, and let $\{p^1 p^2, \ldots, p^m\}$ be the ordered set. Then, if we let $\Psi_{(12)}$ be the component-wise maximum of $\{\Psi^1, \Psi^2 + u^1, \Psi^3 + u^1 + u^2, \ldots, \Psi^m + u^1 + \cdots + u^{m-1}\}$, it is clear that $\Psi_{(12)}$ is sufficient to enable the firing sequence $\sigma\tau^1\tau^2 \ldots \sigma\tau^m$, and thus after the Type-II reduction, $\Psi_{(12)}$ for $p_{(12)}$ in $\rho_2\rho_1(\eta) = (\rho_2\rho_1)^{n+1}(\eta)$ is sufficient to enable firing sequence $\sigma\tau^1\tau^2, \ldots, \sigma\tau^m$ in N, which enables $p_{(1)}\bullet$. □

Algorithm 2 uses Type-I and Type-II reductions to compute resource levels sufficient to guarantee quasi-liveness. The algorithm starts with N, and for each process place, defines a bounding function, Ψ, for each resource. This bounding function is initialized to the resource need of the place. The While loop then updates the bounding function and applies reductions until the net is irreducible, at which point the resource bounds are returned.

More specifically, in the first For loop, the resource bound of the last place of each Type-I structure is updated with the maximum resource usage along the structure. Thus, the resource bound associated with the last place of each Type-I structure will be the number of units of each resource necessary for the subprocess to reach the synchronization transition. After these updates, the net reduction is applied.

After the Type-I reduction, if the net is not irreducible, at least one Type-II structure will be present. For each Type-II structure, say $\{\langle t_I, p_{(11)}, t_{(11)}, p_{(12)}, t_j\rangle, \langle t_I, p_{(21)}, t_{(21)}, p_{(22)}, t_j\rangle \ldots \langle t_I, p_{(m1)}, t_{(m1)}, p_{(m2)}, t_j\rangle\}$, the second For loop first updates the resource bounds of the place in the first path, $\langle t_I, p_{(11)}, t_{(11)}, p_{(12)}, t_j\rangle$, as illustrated and discussed above, and then deletes the other places.

Input: $N \in$ G-AMG$_A$ and critical resource, r_c
Output: Serialized bounds, sequence of place markings.
$\eta = N$, $\vartheta = \emptyset$ (ϑ is a last-in-first-out list)
For $p \in P_S \cup P_I \cup P_F$
  $\Psi_h(p) = u_h(p)$ for $h = 1, \ldots, |P_R|$
While $\eta \neq \aleph$
  For each Type-I structure $\langle t_I, p_{(1)}, \ldots, t_{(n-1)}, p_{(n)}, t_{(n)} \rangle$ in η
    $\Psi_h(p_{(n)}) = \max\{\Psi_h(p_{(1)}), \ldots, \Psi_h(p_{(n)})\}$, $h = 1 \ldots |P_R|$
End For
$\eta = \rho_1(\eta)$
For a Type-II structure $\{\langle t_I, p_{(11)}, t_{(11)}, p_{(12)}, t_j \rangle, \langle t_I, p_{(21)}, t_{(21)}, p_{(22)}, t_j \rangle \cdots \langle t_I, p_{(m1)}, t_{(m1)}, p_{(m2)}, t_j \rangle\}$ in η
  Sort $\{p_{(12)}, p_{(22)} \cdots p_{(m2)}\}$ by decreasing δ_c and let $\langle p^1, \ldots, p^m \rangle$ be the sorted set
  Insert $\langle p^1, \ldots, p^m \rangle$ into ϑ
  For $h = 1 \ldots |P_R|$
  Set $\Psi_h(p_{(12)})$ to
    $\max\{\Psi_h(p^t) + \sum_{j=1}^{t-1} u_j : t = 1, \ldots, m\}$
    End For
  End For
  $\eta = \rho_2(\eta)$
End While
Return $\{\langle \Psi_h : h = 1 \ldots |P_R| \rangle, \vartheta\}$

Algorithm 2

We note that before the bounds are computed for the Type-II reduction, the places in the Type-II structure are sorted. These sorted sets are saved on a last-in-first-out list and returned by the algorithm since they can be used to construct the serialized sequence which corresponds to the computed bounds. Thus, if the serialized bounds computed by Algorithm 2 are met, a serialized sequence can be easily constructed and, in the strictest sense, enumeration of the reachability graph need not occur. However, some additional enumeration and search might be desirable, since the serialized transition firings limit the concurrency of subprocesses.

Consider Figure 6. The resulting net is obtained after Type-I and Type-II reductions have been applied to the example assembly system of Figure 3. A Type-I structure, $\langle t_I, p_1, t_1, p_3, t_{10}, p_{10}, t_{12} \rangle$, can be further reduced, giving rise to $\langle t_I, p_{(11)}, t_{(11)}, p_{(12)}, t_{12} \rangle$ with $\Psi(p_{(12)}) = \langle 132 \rangle$, $u(p_{(12)}) = \langle 030 \rangle$ and $\delta(p_{(12)}) = \langle 102 \rangle$. We now have a Type-II structure: $\langle t_I, p_{(11)}, t_{(11)}, p_{(12)}, t_{12} \rangle$ and $\langle t_I, p_{11}, t_{11}, p_{12}, t_{12} \rangle$. Assuming that r_1 is a critical resource and hence a sorting key, the Type-II structure is reduced to $\langle t_I, p_{(11)}, t_{(11)}, p_{(12)}, t_{12} \rangle$ with $\Psi(p_{(12)}) = \langle 241 \rangle$. However, if r_2 is used as a sorting key and the subprocess $\langle t_I, p_{11}, t_{11}, p_{12}, t_{12} \rangle$ is executed first, then the Type-II structure is reduced to having $\Psi(p_{(12)}) = \langle 343 \rangle$. If $\langle C_1, C_2, C_2 \rangle = \langle 2, 4, 1 \rangle$, then the first Type-II reduction guarantees quasi-liveness.

We note that it is possible to compute a looser set of resource bounds that guarantees that any precedence feasible sequence of transition firings is resource enabled by replacing the interior of the third For loop with the following statement:

$$\Psi_h(p_{(1)}) = \sum_{j=1}^{k} \Psi_h\left(p_{(j)}\right), \qquad h = 1 \ldots |P_R|. \tag{1}$$

This sum guarantees that the maximum resource needs of the corresponding subprocesses can be met simultaneously, and thus every sequence of transition firings will be resource feasible. Clearly, in this case, enumeration is not required.

4. The G-AMG$_{ASU}$ Model for the ASU-RAS

This section studies the subclass of systems in G-AMG$_A$ where net places have single unit resource need; that is G-AMG$_{ASU} \subseteq$ G-AMG$_A$ is the subclass of G-AMG$_A$ where for all $p \in P_S$, $\sum_{h=1}^{m} u_h(p) \leq 1$, where $m = |P_R|$ and u_h being the p-semiflow in item 5 of Definition A.11 of the Appendix. We refer to this class as the ASU-RAS. For the ASU-RAS, we develop resource capacity bounds for which quasi-liveness is polynomial in the number of places and transitions in the underlying G-AMG$_{ASU}$. We also develop a fast method for finding sequences without developing the reachability graph of the underlying G-AMG$_{ASU}$.

More formally, we impose the following additional constraints on W of G-AMG$_A$.

Definition 6. G-AMG$_{ASU}$ is the class of nets obtained by placing the following constraints on G-AMG$_A$:

(1) for all $t_u \in P_I\bullet$, $\sum_{h=1}^{m} W(r_h, t_{iu}) = 1$,

(2) for all $t_u \in T_S$, $\sum_{h=1}^{m} W(r_h, t_u) \leq 1$,

(3) for all $t_u \in T_S$, if $\sum_{h=1}^{m} W(r_h, t_{iu}) = 1$, then $W(t_u, r_h) = \sum_{p \in \bullet t_{u \cap P_S}} u_h(p)$, for $h = 1, \ldots, m$,

(4) for all $t_u \in T_S \setminus \bullet P_F$, if $\sum_{h=1}^{m} W(r_h, t_u) = 0$, then

$$\sum_{h=1}^{m} W(t_u, r_h) = \left(\sum_{h=1}^{m} \left(\sum_{p \in {}^\bullet t_{u \cap P_S}} u_h(p) \right) \right) - 1, \quad \text{for } h = 1 \ldots m,$$

$$W(t_u, r_h) \in \left\{ \max\left\{ 0, \left(\sum_{p \in {}^\bullet t_{u \cap P_S}} u_h(p) \right) - 1 \right\}, \sum_{p \in {}^\bullet t_{u \cap P_S}} u_h(p) \right\}. \tag{2}$$

The first constraint says that a subprocess must be allocated a resource for its first processing step. The second says that no more than one unit of one resource type may be requested at a transition. The third says that when a unit of resource is allocated at a transition, all resources held by the requesting subprocesses must be returned. Finally, the fourth says that if a transition does not allocate a resource, then the return must be exactly one unit less than the number currently held (except for transitions in $\bullet P_F$, which release all resources). We have the following lemma.

Lemma 7. *For any $P \in P_S \setminus (P_I \cup P_F)$, the resource need vector is an m dimensional unit vector.*

Proof. For $P \in P_S \setminus (P_I \cup P_F)$ we consider the following exhaustive cases.

Case 1. Suppose $u_h(p) = 0$ for $h = 1, \ldots, m$. By (1), $P \notin P_I \bullet \bullet$, but there must be a path, say γ, from some $P_u \in P_I$ *to* P. The first transition of the path, $p_u\bullet$, allocates one unit of some resource to the corresponding subprocess. Thus, some transition along γ must deallocate all resources with no additional allocation. This violates (4).

Case 2. Suppose $u_h(p) = k > 1$ for some r_h. Either these k units of r_h are accumulated through at least k transitions or they result from insufficient resource release at the firing of a synchronization transition. By (3), when a resource is allocated to a set of requesting subprocesses, all resources held by those subprocesses must be released. Thus, resources cannot be accumulated through consecutive transitions firings. By (4), if no resources are allocated at a transition, the corresponding subprocesses must still return all resources held except one. Thus, $u_h(p) = k > 1$ for some r_h violates both (3) and (4).

Case 3. Suppose $u_h(p) = 1$ and $u_k(p) = 1$. By the logic of Case 2, this is impossible. □

Now, for $N_R \in$ G-AMG$_{\text{ASU}}$, the reversed subnet, N_R', as defined in Section 3, has splitting (disassembly) but no merging. In the following, we use N_R' to develop resource bounds that guarantee quasi-liveness and polynomial sequence enumeration for N_R. Note that $t_u \in T_{\text{Synch}}$ in N_R is a disassembly transition in N_R'. Let T_{Split} be the set of disassembly transitions in N_R'. Note that for $t \in T_{\text{Split}}$, $|\bullet t \cap P_{\text{Si}}| = 1$ and $\sum_{h=1}^{m} u_h(\bullet t \cap P_S) = 1$. If $u_h(\bullet t \cap P_S) = 1$, we refer to r_h as the "disassembly resource." Let $T^h_{\text{Split}} = \{t : t \in T_{\text{Split}}$ and $u_h(\bullet t \cap P_S) = 1$ in $N_R'\}$, $h = 1, \ldots, m$. The set, T^h_{Split}, collects all the disassembly transitions in N_R' that have r_h as the disassembly resource. Since each disassembly utilizes a single resource type, we have $T^u_{\text{Split}} \cap T^v_{\text{Split}} = \emptyset$ when $u \neq v$.

For $t \in T_S$, let $\Gamma(t)$ be the set of transitions in T_S reachable from t in paths of N_R' not containing p_0. Note that for $t \in T_S$, $\Gamma(t)$ identifies reachable transitions that occur later in the disassembly process. Let $LT^h_{\text{Split}} = \{t : t \in T^h_{\text{Split}}$ and $T^h_{\text{Split}} \cap \Gamma(t) = \emptyset\}$ and note that LT^h_{Split} represents the set of disassembly transitions that use r_h as the disassembly resource but have no reachable transition (without including p_0) that does the same. That is, these are disassembly transitions, which use r_h, that occur latest in the disassembly process. The following lemma guarantees that the total token count in the set of disassembly operation places requiring r_h is no greater than $|LT^h_{\text{Split}}|$.

Lemma 8. *Given an N_R', if $M_0(p0) = 1$ and $M_0(p) = 0$, text for all $p \in P_S \cup P_I \cup P_F$, then for every marking M_j such that $M_0[\rangle M_j$, $M_j(\bullet T^h_{Split} \cap P_S) \leq |LT^h_{Split}|$.*

Proof. Let N' be initially marked as given above. Note that N_i' is a strongly connected marked graph with every circuit containing the place p_0 initially marked with exactly one token. For $t_j \in |LT^h_{\text{Split}}|$, there exists a path, from t_F to t_j, say τ_j. Note that there exists a circuit passing through t_j, say γ_j, such that τ_j is a subpath of γ_j. For any other $t_k \in |LT^h_{\text{Split}}|$, t_j and t_k are mutually unreachable except through paths including p_0. As a result, t_j and t_k are not in a common circuit. This implies LT^h_{Split} circuits passing through elements of LT^h_{Split}. Further note that for $t_u \in T^h_{\text{Split}} \setminus LT^h_{\text{Split}}$, $\exists t_v \in LT^h_{\text{Split}}$ such that $t_v \in \Gamma(t_u)$; this implies that t_u and t_v belong to a common circuit. Thus, the number of circuits in which resource r_h is used as a disassembly resource is precisely LT^h_{Split}. By the fundamental property of marked graphs, $M_j(\bullet T^h_{\text{Split}} \cap P_S) \leq |LT^h_{\text{Split}}|$. □

Note that $|LT^h_{\text{Split}}|$ can be quickly and easily computed for each resource and will play an important role in developing an enumeration policy for N_R'. Now consider the following lemma.

Lemma 9. *Given an N_R', suppose $M_0(p_0) = 1$, $M_0(p) = 0$, for all $p \in P_S \cup P_I \cup P_F$, and that $M_0[\rangle M_k$. Define induced marking M_k^* as follows:*

$$M_k^* = \begin{cases} 0 & \forall\, p \in \bullet T_{Split} \cap P_S \\ 1 & p_0 \\ M_k(p) & \text{otherwise.} \end{cases} \tag{3}$$

If marking M_k^* is free of deadly marked siphons, then M_k is free of deadly marked siphons.

Proof. We prove this result by contradiction. Let s' be a deadly marked siphon in M_k. Then, there will exist another

siphon $s \subseteq s'$ which is deadly marked in M_k and minimal. The structure of N_R' implies that the minimal siphons containing place p_0 are the circuits of the marked graph, N'. This observation, when combined with the presumed structure for the initial marking M_0, implies that, for any marking $M_k \in R(N_R', M_0)$, $p_0 \notin s$. But, the construction of M_k^* implies that s does not increase its token content, and, therefore, it constitutes a deadly marked siphon for M_k^*. The last conclusion contradicts the working assumption and concludes the proof. □

The importance of the marking M_k^* is that its corresponding subprocesses are each strictly SU-RAS for at least one step. That is, any token in M_k^* is holding one unit of resource and requesting one unit of resource. When the requested unit is allocated, the held unit is released, and the token advances to its next place. The lemma guarantees that if there is no deadlock among the subprocesses of M_k^* (assuming the reduced resource capacity levels of M_k^*), then there is no deadly marked siphon in M_k. We will use this fact along with resource bounds to be computed from the results of Lemma 9 to develop a single step look-ahead enumeration policy for N_R' that is polynomial in net size. The policy is as follows.

Enumeration Policy Φ. Let σ_j be a firing sequence for N_R' such that $M_0[\sigma_j\rangle M_j$ and suppose t_u is enabled at M_j such that $M_j[t_u\rangle M_k$. Admit the extension $\sigma_j t_u$ only if the marking M_k^* is free of deadly marked siphons.

We note that detecting whether or not a marking has a deadly marked siphon is polynomial in the size of the net and is thus very fast. However, allowing markings only if they are free of deadly marked siphons does not guarantee policy correctness since we may admit markings from which deadly marked siphons are unavoidable. For our purposes, we will define policy correctness as follows.

Definition 10. An enumeration policy is "correct" if for any marking, M_j, admitted under the policy, there exists a sequence of transition firings, $\sigma_j \neq \varepsilon$, such that

(1) $M_j[\sigma_j\rangle M_0$,

(2) for any prefix of σ_j, say τ_k, where $M_j[\tau_k\rangle M_k$, M_k is admitted under the policy.

We, now, are in the position to prove the following.

Theorem 3. *For $t_j \in T_{Split}^h$ and $h = 1, \ldots, m$, let*

$$\begin{aligned} U_j^h &= \sum_{p \in t j^{\bullet} \cap P_S} u_h(p), \\ U_{\max}^h &= \max\left\{U_j^h : \; t_j \in T_{Split}^h\right\}, \\ B_h &= \left|LT_{Split}^h\right| \; + \; U_{\max}^h + \; 2. \end{aligned} \tag{4}$$

If for $h = 1, \ldots, m$, $C_h \geq B_h$, then Φ is correct.

Proof. Suppose that a marking, M_k, is accepted by Φ. Then M_k^* contains no deadly marked siphon and thus M_k contains no deadly marked siphon. Note that in M_k^*, the capacity of every resource is at least $U_{\max}^h + 2$, $h = 1 \ldots m$. Let Π be the set of subprocesses in M_k where $\Pi = \Pi_{ND} \cup \Pi_D$, $\Pi_{ND} \cap \Pi_D = \emptyset$. Π_D is the set of subprocesses at disassembly operations, that is, tokens marking $\bullet T_{\text{Split}} \cap P_S$, and Π_{ND} is the set of subprocesses not at disassembly. □

Case 1. Suppose that $\Pi_{ND} \neq \emptyset$ in M_k. Since there is no deadly marked siphon in M_k^*, there is no subset of Π_{ND} deadlocked in M_k^*. Thus, $\exists \pi_u \in \Pi_{ND}$ and enabled $t_v \notin T_{\text{Split}}$ such that firing t_v allocates a unit of resource r_h to π_u and causes π_u to release a unit of resource r_p.

Now suppose that $M_k[t_v\rangle M_g$ and that M_g contains a deadly marked siphon. Thus, M_g^* contains a deadly marked siphon, which implies a deadlock among processes of Π_{ND} in M_g^*. Because of the resource bounds, each deadlocked subprocess of M_g^* is blocked by at least two other deadlocked subprocesses of M_g^*.

To summarize, we have the following: (1) M_k^* has no deadlock among Π_{ND}, (2) $M_k^*[t_v\rangle M_g^*$, (3) t_v allocates a single unit of r_h to π_u and releases a single unit of r_p, (4) M_g^* has a deadlock among Π_{ND}, and (5) every deadlocked subprocess of M_g^* is blocked by at least two other deadlocked subprocesses of M_g^*.

It is clear that allocating r_h to π_u causes the deadlock, implying that r_h is a resource involved in the deadlock. Thus, in M_g^*, at least two units of r_h are allocated to subprocesses in Π_{ND}, and in fact, there must be another subprocess $\pi_a \in \Pi_{ND}$ requesting r_h at $t_a \notin T_{\text{Split}}$ in both M_k^* and M_g^*. Allocating r_h to π_a rather than π_u, that is, $M_k[t_a\rangle M_p$ cannot result in deadlock among processes of Π_{ND}. Hence neither M_p^* nor M_p contains a deadly marked siphon.

Case 2. Suppose that $\Pi_{ND} = \emptyset$ and $\Pi_D \neq \emptyset$ in M_k. There exist only subprocesses at disassembly operations. Thus, each resource has at least $U_{\max}^h + 2$ free units, $h = 1 \ldots m$. Sufficient resources are available to fire any transition of T_{Split}. Suppose $t \in T_{\text{Split}}$ is enabled in M_k and that $M_k\ [t\rangle Mg$. M_g^* contains no deadly marked siphon. To see this, note that if M_g has $\Pi_{ND} = \emptyset$, then each resource continues to exhibit at least $U_{\max}^h + 2$ free units, $h = 1 \ldots m$. If M_g has $\Pi_{ND} \neq \emptyset$, then each resource, r_h, $h = 1 \ldots m$, has at least 2 units of free capacity.

Thus, Enumeration Policy Φ guarantees resource-enabled sequences of transition firings that complete the disassembly process, N_R'. We are now ready to present Algorithm 3. It starts with N_R' in the initial marking and generates a firing sequence that completes the disassembly by using single step look-ahead for deadly marked siphons. The most computationally intensive step is the siphon check, which can be done in polynomial time, no worse than $O(|P_S| + |T_S|)$. By Theorem 3, the loop will require no more than $|T_S|$ iterations, since every iteration will identify an admissible transition, and thus the algorithm is $O(|T_S|^2)$. By returning the reversed sequence, we get the resource enabled assembly sequence for the assembly net, N_R. We note that the termination request computations of Algorithm 1 can easily be implemented in Algorithm 3.

Input: (N_R', M_0)
Output: $\sigma_j \neq \varepsilon$ such that $M_0[\sigma_j\rangle M_0$
Set $\sigma_j = t_F$, and fire t_F
Set $M_j = M_0[t_F\rangle$
Loop
  Find $t \in E_t(M_j)$ st $M_j\ [t\rangle M_k$, M_k^* contains no deadly marked siphon
    $\sigma_j = \sigma_j t$
    If $M_k = M_0$, return reverse (σ_j)
    Else $M_j = M_k$
End Loop

Algorithm 3

As an aside, we note that the converse of Lemma 9 is not true; that is, a deadly marked siphon in M_k^* does not imply a deadly marked siphon in M_k. In fact, it is easy to illustrate markings which are "safe" in the sense that the firing sequence can be extended to reach M_0 but for which the induced marking exhibits a deadly marked siphon and is rejected. Thus, the Enumeration Policy Φ is suboptimal in the sense that it rejects some transition firings that lead to "safe" markings. Further, even when the capacity bounds of Theorem 3 are in place, $N_R^{'}$ can exhibit markings with no deadly marked siphon but from which every sequence of transition firings leads to a marking with a deadly marked siphon. Thus, a policy that does single step look-ahead on the unaltered markings of $N_R^{'}$ is not correct. Finally, we note that since Theorem 3 applies to disassembly systems, when the specified bounds are in place, quasi-liveness is guaranteed and sequence enumeration is polynomial for the class of disassembly nets G-AMG$_{\text{DSU}}$.

5. Conclusion

In this paper, we developed models and algorithms for a class of Petri nets that support resource allocation in systems with synchronization and splitting operations. Our focus was on establishing quasi-liveness and enumerating process completing sequences. This is challenging since, for this class of systems, the quasi-liveness problem is NP-complete. Our tenet is that once quasi-liveness is established and a process completing sequence is generated, previously published liveness enforcing supervisors can be used to control the operation of these systems. For the general case, we proposed a breadth-first search algorithm that generates the reachability tree and computes minimal termination requests for each marking. We discussed the complexity of this approach as well as the need for selecting a smaller set of sequences for use in supervision. We then developed two special subclasses that for systems with assembly only, and for each class established that polynomial sequence enumeration is possible if the resource capacities meet certain bounds. The first subclass was assembly with conjunctive resource allocation. For this class, we developed a net reduction algorithm that reduces the net to a minimal form and, in so doing, computes a resource sufficiency bound for "serialized" firing sequences. The second special case was that of assembly with single unit resource allocation. For this class, we developed resource bounds and an enumeration policy that guarantees a process completing sequence in polynomial time. In current and future work, we are addressing liveness enforcing supervision for assembly/disassembly systems with unreliable resources, particularly those subject to degradation.

Appendix

Definition A.11. A G-AMG is a Petri net, $N = (P, T, W, M_0)$ such that

(1) $P = P_S \cup P_I \cup P_F \cup P_0 \cup P_R$, where $P_S \cap P_I \cap P_F \cap P_0 \cap P_R = \emptyset$;

(2) $T = T_S \cup T_I \cup T_F$, where $T_S \cap T_I \cap T_F = \emptyset$;

(3) $W : (P \times T) \cup (T \times P)Z^+ \rightarrow$ satisfies the following:

  (a) $(P \times T) \cup (T \times P) \rightarrow \{0, 1\}$ such that $(\{p_0\} \times (T_S \cup \{t_I, t_F\})) \rightarrow \{1\}$ for (p_0, t_I), and $\{0\}$ otherwise. Similarly, $((T_S \cup \{t_I, t_F\}) \times \{p_0\}) \rightarrow \{1\}$ for (t_F, p_0), and $\{0\}$ otherwise,

  (b) $(\{t_I\} \times P_I) \rightarrow \{1\}$ and $(\{t_I\} \times (P_S \cup P_F)) \rightarrow \{0\}$. Similarly, $(P_F \times \{t_F\}) \rightarrow \{1\}$ while $((P_S \cup P_I) \times \{t_F\}) \rightarrow \{0\}$,

  (c) $(P_R \times T_S) \cup (T_S \times P_R) \rightarrow Z^+$ and $(P_R \times (T_I \cup T_F)) \cup ((T_I \cup T_F) \times P_R) \rightarrow \{0\}$;

(4) the net generated by $P_S \cup P_I \cup P_F \cup \{p_0\} \cup T_S \cup \{t_I, t_F\}$ is a strongly connected marked graph such that every cycle contains $\{p_{0i}\}$;

(5) for all $r \in P_R$, and there exists a minimal integral p-semiflow, u_r, such that $\|u_r\| \cap P_R = \{r\}$, $\|u_r\| \cap \{P_0 \cup P_I \cup P_F\} = \emptyset$, $\|u_r\| \cap \{P_S\} \neq \emptyset$, and $u_r(r) = 1$;

(6) N is pure and strongly connected.

$M_0 : P \rightarrow Z^+$ with $M_0(p) \geq 1$, for all $p \in P_0 \cup P_R$ and $M_0(p) = 0$, otherwise.

References

[1] S. A. Reveliotis, M. A. Lawley, and P. M. Ferreira, "Polynomial-complexity deadlock avoidance policies for sequential resource allocation systems," *IEEE Transactions on Automatic Control*, vol. 42, no. 10, pp. 1344–1357, 1997.

[2] Z. A. Banaszak and B. H. Krogh, "Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 6, pp. 724–734, 1990.

[3] M. A. Lawley, S. A. Reveliotis, and P. M. Ferreira, "A correct and scalable deadlock avoidance policy for flexible manufacturing systems," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 5, pp. 796–809, 1998.

[4] K. Barkaoui, A. Chaoui, and B. Zouari, "Supervisory control of discrete event systems based on structure theory of Petri nets," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pp. 3750–3755, October 1997.

[5] W. Sulistyono and M. A. Lawley, "Deadlock avoidance for manufacturing systems with partially ordered process plans," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 819–832, 2001.

[6] S. A. Reveliotis, "Liveness Enforcing supervision for sequential resource allocation systems: state of the art and open issues," in *Synthesis and Control of Discrete Event Systems*, B. Cailaud, X. Xie, P. Darondeau, and L. Lavagno, Eds., pp. 203–212, Kluwer Academic Publishers, 2002.

[7] Z. Li, N. Wu, and M. Zhou, "Deadlock control of automated manufacturing systems based on Petri nets—a literature review," *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, no. 99, pp. 1–26, 2011.

[8] Z. W. Li, M. C. Zhou, and N. Q. Wu, "A survey and comparison of Petri net-based deadlock prevention policies for flexible manufacturing systems," *IEEE Transactions on Systems, Man and Cybernetics Part C*, vol. 38, no. 2, pp. 173–188, 2008.

[9] S. Reveliotis, "Structural analysis of resource allocation systems with synchronization constraints," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1045–1049, September 2003.

[10] E. Roszkowska and R. Wojcik, "Problems of process flow feasibility in FAS," in *CIM in Process and Manufacturing Industry*, pp. 115–120, Pergamon Press, 1993.

[11] X. Xie and M. Jeng, "ERCN-merged nets and their analysis using siphons," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 4, pp. 692–703, 1999.

[12] N. Wu, M. C. Zhou, and Z. W. Li, "Resource-oriented Petri net for deadlock avoidance in flexible assembly systems," *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, vol. 38, no. 1, pp. 56–69, 2008.

[13] F. S. Hsieh, "Robustness analysis of non-ordinary Petri nets for flexible assembly systems," *International Journal of Control*, vol. 83, no. 5, pp. 928–939, 2010.

[14] H. Hu, M. Zhou, Z. Li, and N. Wu, "Deadlock-free control of ratio-enforced automated manufacturing systems with flexible routes and assembly operations," in *Proceedings of the 6th Annual IEEE International Conference on Automation Science and Engineering (CASE '10)*, pp. 459–464, August 2010.

[15] S. Chew, M. Lawley, and S. Reveliotis, "Liveness enforcing supervision for resource allocation with complex workflows," in *Proceedings of the 9th IEEE Methods and Models in Automation and Robotics (MMAR '03)*, pp. 823–829, 2003.

[16] S. F. Chew, M. A. Lawley, and S. A. Reveliotis, "Liveness enforcing supervision for resource allocation systems with process synchronizations," in *Proceedings of the 42nd IEEE Conference on Decision and Control*, pp. 3735–3741, December 2003.

[17] J. Y. Choi, "The thinning problem," *Engineering Optimization*, vol. 42, no. 2, 2010.