Inappropriate handling of resource sharing is the major cause of deadlocks in Flexible Manufacturing System (FMS) modelled by Petri nets (PN). Zhou et al [ZHO 91] presented the deadlock-free condition (DFC) of a PN containing Sequential Mutual Exclusion (SME). This paper presents an alternative approach to explain SME and Parallel Mutual Exclusion (PME) in the context of the knitting technique and structural relationship, illustrates the application of S-Matrix to detect deadlocks in SMEs, generalizes the DFC for more complicated SMEs than the example in [ZHO 91], and discovers new DFCs. The structural matrix (S-Matrix) records the structural relationship among the processes and thus can be used to detect the violation of the synthetic rules upon a new generation of paths in the Petri net. To synthesize the SME, we have enhanced the algorithm to construct the S-Matrix.

Key words: FMS, Petri net, deadlock, liveness, boundedness, shared resource, PME, SME, S-matrix, S-Matrix, knitting rules, synthesis, deadlock-free condition.

---

1 This paper was partially supported by New Jersey Institute of Technology under grant No's 421190 and 421380.

2 The former name of the first author was Yuh Yaw, which has appeared in some of his earlier papers.
I. Introduction

The FMS offers a very promising approach to the increase of productivity through state-of-the-art manufacturing technology. The modeling and control of FMS are part of the great challenge to the professionals in engineering, computer science, mathematics, and management. PNs have been applied to the specifications, validations, performance analysis, control code generation, and simulation for automated manufacturing systems [COU 83, MAR 86, NAR 85, MUR 86, CHA 93].

A problem of using PNs [MUR 89] for modeling various systems is the large number of states generated. Synthesis can eliminate this problem by avoiding analysis. Various synthesis approaches have been proposed: top-down/bottom up [BER 86, DAT 84, SUZ 83, VAL 79], peer entity generation [RAM 86b], and knitting technique [YAW 87, CHA 93c-e 97, 98]. Most of them do not deal with PNs with shared resources. Zhou et al. [ZHO 91] developed a deadlock-free condition (DFC) of such PNs for synthesis.

Automated manufacturing systems often share resources such as machines, robots, and material transporters. There are two forms of resource sharing: P:ME and S:ME as proposed by [ZHO 91]. A P:ME models a resource shared by independent processes and an SME models sequentialization (synchronization) of PMEs. In a P:ME, any independent process may monopolize the use of a resource, thus creating an unfair situation. The SME eliminates this unfairness problem by sequentializing or synchronizing these independent processes. Because of the nature of resource sharing, processes in a system can enter a deadlock state by waiting for completion.

The PN models of SMEs have deadlocks due to either inappropriate structure of the net or incorrect allocation of shared resources. Zhou et al. [ZHO 91] discovered that a net with an SME can become nonlive and nonreversible when tokens are inappropriately distributed in the net. They derived the conditions under which a net containing SMEs is bounded, live and reversible. They pointed out that an algorithm is needed to detect SMEs automatically and the DFC should be generalized such that the shared resource can contain more than one token. This paper presents an alternative approach to automatically detect SME and explain SME and PME. Furthermore, it generalizes the DFC for SMEs more complicated than Zhou's work and discovers a new DFC. The major objectives of this paper are

1. Explain SME and PME in the context of the knitting technique/structural relationship.
2. Illustrate the application of S-Matrix to detect SMEs, and deadlocks in PMEs.
3. Synthesize PME and SME based on the knitting technique.
4. Generalization of DFC to more complicated SMEs than the example in [ZHO 91].
5. Discover new DFCs unvisited in [ZHO 91].

The knitting technique is a rule-based interactive approach. It expands PN in a structural fashion based on a set of rules. While it takes
exponential time complexity to determine marking properties; it may take polynomial time to determine structural properties such as structural liveness (SL) and boundedness (SB). It aims to find the fundamental constructions for building any PNs. There are two advantages of the knitting technique: (1) reduction of the complexity of synthesis as an interactive tool and (2) providing knowledge of which construction building which class of nets. It therefore opens a novel avenue to PN analysis.

Rather than refining transitions [RAM 86a], this technique generates new paths to a PN, N₁, producing a larger PN, N₂. The new generations are performed in such a fashion that all reachable markings in N₁ remain unaffected in N₂; hence all transitions and places in N₁ stay live and bounded, respectively. N₂ is live and bounded by making the new paths live and bounded. This notion is novel compared with other approaches and could synthesize more general PNs than others [CHA 93c].

Using the knitting technique, designers start with a basic process which is modeled by a set of sequentially-connected places and transitions with a home-place (defined in Section 3) marked with a certain number of tokens. The tokens may represent a number of resources which can be present in a system each time. Then parallel and exclusive processes or operations are added according to the system specification. Closed circuits for the operations are added according to the resources required by the operations involved. Since expansions are conducted among the nodes (transitions or places) in a global way, the knitting technique is thus called. This approach is easy to use due to the simplicity of the rules, and leads to the final well-behaved PN. Another advantage of this knitting technique is that it is easily adapted to computer implementation for rendering the synthesis of PNs performed in a user-friendly fashion [YAW 89].

Structural matrix (S-Matrix) is used in the knitting technique to record the structural relationships (concurrent, exclusive, sequential, cyclic, etc.) among processes and to detect rule violations upon a new generation from one node to another. Because the structural relationship also resembles their temporal relationship, S-Matrix is also called the temporal matrix (T-Matrix). There are four types of path generations; namely TT, PP, TP, and PT; depending on whether the generation is from a transition or place to a transition or place. One of the following three actions are taken depending on the structural relationship between the two nodes and the type of generation: (1) forbidden, (2) permitted, (3) permitted but more new generations are needed. For instance, a TT (PP)-generation is permitted between two sequential or concurrent (exclusive) processes, while TP (PT) generation is forbidden since it may create unbounded (nonlive) nets. The rules are complete in the sense that all possible generations are considered. Recently, we have upgraded the knitting technique by relaxing some forbidden
rules. For instance, we allow TT-generations between exclusive processes [CHA 92] and permit TP and PT generations [CHE 93a].

A TT-path from a transition $t_g$ to another transition $t_f$ creates a concurrent process. If $t_f$ fires earlier than $t_g$ in the original PN $N_1$, tokens must be inserted in the new path to avoid deadlocks. Such a generation is called a backward generation. Multiple PP-paths from a place to itself creates a PME. These PP-paths are mutually exclusive to each other. TT-generations between these exclusive processes can sequentialize or synchronize them; therefore generating an SME. Thus the creation of SME and PME can be automatically detected using the S-Matrix. Further TT-generations can be performed between transitions within an SME or in different SMEs. In both cases, the appropriate number of tokens may need to be inserted into the new paths to avoid deadlocks, since the above transitions are now sequential to (hence it may be a backward generation) each other. The corresponding conditions are more general than the DFC by Zhou et al. where a shared resource has only one token. Whether a TT-generation is within an SME, or is between different SMEs, or is not in any SMEs, is readily available from the S-Matrix. Thus the generalized DFC is applicable to complicated structures rather than simple structures such as that by Zhou et al. Further, we discover a new DFC for generations between different SMEs which was not envisioned by Zhou et al.

An algorithm was developed [YAW 89, CHA 93f] accordingly and to update the T-matrix with polynomial time complexity (since only structural information is needed). We have implemented this algorithm in an X-Window/Motif environment. It can synthesize protocols among multiple parties and has been applied to synthesis of communication protocols [YAW 87, CHA 94a] and automated manufacturing systems [CHA 93a]. In [CHA 93c], we have extended these rules to synthesize General Petri Nets (GPNs) whose arcs have multiple weights. This technique can synthesize well-behaved nets more general than free-choice (FC), extended free-choice (EFC), asymmetric-choice nets (AC) [MUR 89]. FC is a subset of EFC, which, in turn, is a subset of AC.

Other contributions in this direction are as follows. Esparza & Silva [ESP 91a] proposed two rules to synthesize live and bounded free-choice PNs. Their TT- and PP- handles [ESP 91b] are similar to the TT- and PP-paths in [YAW 87]. Similar to [YAW 87] and rephrased in another way, they also showed that circuits without TP- and PT-handles have good structural properties. However, they do not explicitly apply them to the synthesis using the notion of structural relationship. In [ESP 91a], Rule RF1 refines a macroplace by a state machine and RF2 adds an marking structurally

---

3. $\forall p \in P, |p| \leq 1 \Rightarrow (p') = \{p\}$

4. $\forall p_1, p_2 \in P, p_1 \cap p_2' \neq 0 \Rightarrow p_1' = p_2'$

5. $\forall p_1, p_2 \in P, p_1 \cap p_2' \neq 0 \Rightarrow p_1' \subseteq p_2'$ or $p_1' \supseteq p_2'$
implicit place (MSIP) to a free-choice net. RF1 corresponds to the PP rule in [YAW 87] to add paths between places. RF2 corresponds to the TT rule in [YAW 87] to add the paths between transitions and increase the degree of concurrency. However, one needs to decide whether the subnet can be reduced to a macroplace (in RF1) and whether a place is an implicit place (in RF2). The knitting technique [YAW 87] requires no checking on the paths generated; rather, it uses the S-Matrix to check automatically whether the generation points and joints satisfy certain constraints. This is simpler than checking many different subnets and a linear algebra equation for MSIP. In addition, the S-Matrix can record self-loops and find maximum concurrency with linear time complexity. Although [ESP 91b] added the RF3 rule to synthesize EFC nets, [ESP 91b] is unable to synthesize ACs.

Datta & Ghosh [DAT 84] also proposed two rules that are similar to the TT and PP rules [YAW 87]. Instead of using S-Matrix, they used the labeling rules to guide the augmentation. One can show that the synthesized nets are free-choice nets. These two techniques do not have explicit algorithms and the associated complexity. Chen, Tsai and Chao [CHE 93b] proposed a synthesis procedure to compose two modules with two TT paths. An algorithm to find P-invariant and its complexity was presented. All three techniques do not allow PT and TP generations such as those in Rules TT.3 and PP.2.

The knitting rules are useful for analysis and reduction [CHA 92]. For a given PN, we construct its S-Matrix and analyze mutual structural relationships among PSPs against the rules and DFCs. Any violation spots potential ill-designs. The reverse process of removing PSPs - reduction, according to rules should preserve the properties of the PN. Rather than reducing modules to transitions, we remove paths to reduce the PN. The distinct point of this approach is that while reducing, it can discover wrong designs and suggest how to fix the problem based on the knitting rules.

The rest of the paper is organized as follows. Section II introduces PME, SME and the DFC to motivate this work. Section III presents some preliminaries for understanding the S-Matrix. The algorithm for finding the S-Matrix is presented in Section IV. The guidelines for synthesis and some knitting rules and examples are presented in Section V. The synchronization rule for generating an SME and the rule for TT-generation involving SMEs are discussed in Sections VI and VII respectively, followed by its application to the synthesis of PMEs and SMEs in Section VIII. Section IX concludes this paper.

II. Examples of PME, SME and DFC

First we follow the same examples in [ZHO 91] to explain PME and SME, and in later sections, we will present other examples to demonstrate more complicated SMEs. An example of PME is shown in Fig. 1 where the two circuits \([p_1 \ t_1 \ p_3 \ t_3 \ p_1]\)
and \([p_2, t_2, p_4, t_4, p_5]\) form a 2-PME (a k-PME has k independent processes sharing a common resource). They model two independent processes sharing a common resource \(p_5\). In the initial marking, both can be executed; however, they cannot be executed simultaneously. Without the presence of \(p_5\), each of these two processes can iterate infinitely independent of each other.

Fig. 2(a) shows a PN which models a manufacturing system composed of three workstations (W1, W2, W3) and a shared robot for loading/unloading these workstations. The robot is modeled as a shared resource place \(p_6\) with \(M_0(p_6) = 1\), the three empty slots available in W2 are modeled by place \(p_5\) with \(M_0(p_6) = 3\), and the three fixtures with parts available are modeled by place \(p_3\) with \(M_0(p_3) = 3\) representing three initially unprocessed parts. Places \(p_1, p_2, p_3,\) and \(p_4\) model the four major steps, respectively. Initially \(p_1\) has three fixture parts in storage available to W1. The robot now loads a part from the storage for W1 to process in \(p_2\). By firing \(t_2\), the robot unloads W1 and move the fixture part to W3. In \(p_3\), W2 processes the part which is then moved to W3 by the robot. W3 processes the part to be unloaded by the robot and released to the storage.

The place \(p_6\) is shared by transitions \(t_1\) and \(t_3\). Hence they never proceed simultaneously and they are mutually exclusive. The single resource at \(p_6\) is utilized by process 1, \([p_6, t_1, p_2, t_2]\); and process 2, \([p_6, t_3, p_4, t_4]\), which are exclusive to each other without the presence of other nodes and arcs. And the net containing only these two processes is a 2-PME. This 2-PME can be sequentialized by adding processes \([t_2, p_3, t_5]\) and \([t_4, p_1, t_1]\). This results in a cycle \([p_1, t_1, t_2, t_3, t_4, t_4, t_4, p_1]\). By putting tokens into \(p_1\), process 1 gets executed before process 2. Hence they become sequentially mutual exclusive (SME) to each other—no longer just mutually exclusive to each other.

Remark: In Fig. 3(a), processes \([p_3, t_5, p_6, t_5, p_8]\) and \([p_3, t_4, p_7, t_6, p_8]\) form a 2-PME by sharing a common starting place \(p_3\), as when \(p_3\) holds a token, only one of them can be executed. They become SME by adding the circle \([t_3, p_5, t_4, p_4, t_3]\).

A Simple Example of DFC:

Careful observation of the PN in Fig. 2(a) reveals that if the initial marking of place \(p_1\), \(M_0(p_1) > M_0(p_3)\), the system will get into deadlock state. For instance, if the initial marking of \(p_1\), namely \(M_0(p_1) \geq 4\), the system comes to a total deadlock after a sequence of firings of \(t_1\) and \(t_2\). Fig. 2(b) presents a deadlock situation whose marking is \((0,1,3,0,0,0)\) after the firing of \(t_1,t_2,t_1,t_2,t_1,t_2,t_1\). if \(M_0(p_1) = 4\).

Viewing \(M_0(p_1)\) as the number of available jobs; the requirement \(M_0(p_1) \leq M_0(p_5)\) precludes large number of available jobs to be dispatched to the system in a fixed time interval. This is because the empty slots or buffers in the system are limited. Excessive arrivals of jobs or parts exhaust the shared resources in the system and cause deadlocks. Physically, workstations W1 have processed three
parts and W2 has three processed parts with no available empty slots. Now W1 is processing the fourth part along with the robot; however this part cannot enter into W2 to release the robot to be used by W3 to load a part from W2 - a deadlock.

**Observation**: From the above examples, one can conclude that an SME exists if a set of transitions are in a cycle and they share a common input place. Such a cycle is called a Cycle of Exclusive Synchronization (CES).

We adopt the following definitions by Zhou et al. [ZHO 91] defined A-place (or operation place) as a place p with \( M_0(p) = 0 \), B-place (or fixed resource place) as a place if \( M_0(p) \) is constant, and C-place (variable resource place) as a place with \( M_0(p) > 0 \) and \( M_0(p) \) is a variable. A path containing the A-place (B-place, C-place) is called an A-path (B-path, C-path). \( M_C \) and \( M_B \) denote the total number of tokens in the C-path and B-path, respectively. In the above example [Fig. 2(a)], \( p_z \) is an A-place with no tokens in the initial state. \( p_s \) is a B-place representing the fixed internal resource buffer and \( p_1 \) is a C-place representing the varied jobs available. The maximum number of jobs the system can handle are determined by the internal resource buffer size. The system will get into deadlock if the internal resource buffer size is not large enough to process the jobs available: in other words, the condition \( M_0(p_s) \geq M_0(p_1) \) must hold in order to avoid deadlocks. Zhou et al. indicated that this DFC is a sufficient condition only because it considers the case where the resource place holds only one token. They also pointed out that future works should be directed at generalizing PMEs and SMEs and developing algorithms to automatically verify whether a pair of arcs is properly added to \( N_1 \) such that the resultant \( N_2 \) still has a PME or an SME.

### III. Pseudoprocesses, Structural Relationship and S-Matrix

A PN's behavior depends on both its structure and marking. The former (latter) can be determined in a polynomial (exponential) time. For certain markings, the PN's properties depend only on its structure (i.e., in the synthesis using the knitting technique). Structurewise, a PN can be decomposed into a number of straight paths or elementary processes; all nodes of which are mutually sequential to each other. Such a path is called a pseudo-process (PSP). The structure of the PN can be captured by the relationship between all pairs of PSPs and is recorded in the S-Matrix. We now formally define home place, pseudo process and the related terms in the following.

**Definition (Home Place)**: Every \( p_h \) holding tokens initially in a synthesized net is defined as a *home place*. Every circuit contains at most one home place in a synthesized net using our rules. Without tokens in any home place, the net becomes deadlocked. Places \( p_1, p_{10}, p_{11}, \) and \( p_{26} \) in Fig. 4 are home places.

**Definition (Pseudo-Process, Generation Point, and Joint)**: A pseudo process (PSP) in a PN is a
directed elementary path in which any node (transition or place) has only one input node and only one output node except its two terminal nodes: the starting node is defined as the generation point and the end node as the joint.

**Definition (Virtual PSP)**: A virtual PSP (VP) is a two-node PSP.

It should be noted that a virtual PSP contains no node other than the generation point and joint. The examples of PSP and VP are presented after we define the structural relationship between two PSPs. Prior to that, we have

**Definition (Prime ts (Ps) of Two PSPs)**: Let \( t_s(P_1) \) be the starting transition [place] of two directed paths, \( P_1 \) containing \( P_2 \), and \( P_2 \) containing \( P_2 \), which do not share common PSPs. The prime \( t_s(P_s) \) of \( P_1 \) and \( P_2 \), denoted as \( t_s(P_s') \), is the \( t_s(P_s) \) such that any directed path from \( t_s(P_s) \) to \( P_1 \) or \( P_2 \) does not contain any other \( t_s(P_s) \).

**Definition (Structural Relationship between Two PSPs)**: The structural relationship between \( P_1 \) and \( P_2 \) is defined as

1) **Sequential**: \( P_1 \) and \( P_2 \) are sequential (‘SQ’ ) to each other if an elementary circle passes through both \( P_1 \) and \( P_2 \). If there are no tokens in the circle, then \( P_1 \) is cyclic, ‘CL’ to \( P_2 \), denoted as \( P_1 \leadsto P_2 \). Otherwise, consider the \( P \) on the circle from \( P_1 \) to \( P_2 \), if there are no tokens in the \( P \) from \( P_1 \) to \( P_2 \), then \( P_1 \) is sequentially earlier, ‘SE’ to \( P_2 \), denoted as \( P_1 \rightarrow P_2 \) and \( P_2 \) is sequentially later, ‘SL’ to \( P_1 \), denoted as \( P_2 \rightarrow P_1 \). Otherwise, \( P_1 \leftarrow P_2 \) and \( P_2 \rightarrow P_1 \). \( P_1 \) is sequentially previous (next), ‘SP’ (‘SN’) to \( P_2 \), denoted as \( P_1 \rightarrow P_2 \) if \( P_1 \rightarrow P_2 \) and the former’s joint (generation point) coincides with the latter’s generation point (joint).

2) **Concurrent (Exclusive)**: \( P_1 \) is concurrent (exclusive), ‘CN’ (‘EX’), to \( P_2 \), denoted as \( P_1 \parallel P_2 \), if they are not sequential to each other and there exists a \( t_s(P_s') \).

**Definition (Structural Relationship between Two Nodes)**: If two nodes are in the same PSP, they are sequential to each other. If they are in two different PSPs, their structural relationship follows that of the two PSPs.

In Fig. 4, \([p_3 t_3 p_1 t_1 p_2]\) is a PSP whose generation point and joint are \( p_5 \) and \( p_5 \), respectively. \([t_{13} p_{14} t_{13}]\) and \([t_{13} p_{33} t_{14}]\) share the same prime \( t_s = t_{13} \), and hence are concurrent to each other. The two VPs: \([p_{31} t_4]\) and \([p_{31} t_0]\) share the same prime \( p_s = p_{31} \) and, hence, are exclusive to each other. \([t_{21} p_{20} t_{22}] \rightarrow [t_{23} p_{25} t_{26} t_{27} p_{27} t_{23}]\). \([p_3 t_0 p_6 t_0] \rightarrow [p_3 t_3 p_1 t_1 p_2]\).

The structural relationship between two nodes is defined as follows.

**Definition (Structural Relationship between Two Nodes)**: If two nodes are in the same PSP, they are sequential to each other. If they are in two different PSPs, their structural relationship follows that of the two PSPs.

It is easy to see that if we execute a safe PN one iteration involving \( P_1 \), then \( P_1 \rightarrow P_2 \) implies that \( P_1 \) is executed earlier than \( P_2 \), \( P_1 \parallel P_2 \) implies that there is no need to execute \( P_2 \) to complete the iteration, and \( P_1 \parallel P_2 \) implies that
both $\Pi_1$ and $\Pi_2$ need to be executed to complete the iteration. Intuitively, two PSPs are sequential to each other if they are subject to an intra-iteration precedence relationship; i.e., if both are executed in one iteration, then one is executed earlier than the other. They are concurrent to each other if they can proceed in parallel. They are exclusive to each other if it is possible to complete one iteration with only one of them being executed.

Thus a PN consists of a set of PSPs which is enabled if it possesses a token; this PSP executes and the associated token disappears. New tokens are subsequently generated to enable other PSPs. The set of PSPs holding tokens represents the state of the PN. Thus a PN consists of a set of PSPs; their relationships are recorded by the structure Matrix (S-Matrix).

An entry $A_{ij}$ corresponding to row $i$ and column $j$ represents the relationship between pseudo-processes $i$ and $j$. For example, if $\Pi_i$ is concurrent to $\Pi_j$, then $A_{ij} = A_{ji} = \text{CN}$, where CN stands for "concurrent."

IV. Algorithm for Finding S-Matrix and Its Extension to Detect SME

S-Matrix is used for both analysis and synthesis. It records the structural relationship between any two PSPs in the PN, including PSPs which are involved in an SME. The DFC can then be applied to analyze deadlocks. During the course of synthesis, S-Matrix helps determine whether a new generation violates some rules or additional paths must be generated [CHA 98]. It also detects the creation of PMEs and SMEs automatically. In this section, we present the S-Matrix finding algorithm, prove its correctness, and give an example of this algorithm.

Note that 'SE' and 'SL' are marking dependent. For an arbitrary PN, a circle may contain more than one marked place (unlike the synthesis) which may render two PSPs mutually 'SE' to each other, by the definition of structural relationship. To simplify the matter we consider all sequential relationship as 'CL' as any two PSPs which are sequential to each other are also cyclic, by the reversibility property of knitting technique. Thus the S-Matrix contains only four kinds of entries, namely, 'CL', 'CN', 'EX' and 'SX' (defined later).

A systematic procedure such as depth-first or breadth-first search is preferred. Out of the structural relationships, sequential implies "depth" and "exclusive" or "concurrent" (in other words, branchings) implies "breadth". Hence, "depth-first" search implies searching for PSPs sequentially and breadth-first search searching branches. In this paper we concentrate on the "depth-first" search to construct the S-Matrix. The basic algorithm for finding the S-Matrix of a PN, without SME, is presented in Program List 1 and has been implemented on Sun Workstations, under X Window [CHA 94b].
1. for \( i = 1 \) to \( n \)
   \{ \( \text{checked} (\text{node}_i) = \text{FALSE} \) \}
   \{ for \( j = 1 \) to \( k \), \{ \text{marked} (\text{node}_j) = \text{FALSE} \} \}

2. \( \text{home\_place} = \text{an input\_place} \) of an enabled transition; \( j = 0 \);

3. Find the PSP containing \( \text{home\_place} \); \( \text{PSP\_number} = j \);

4. \( \text{current\_node} = \text{PSP\_joint} \); \( \text{start\_pso} = \text{PSP} \)

5. \( \text{while}(1) \)

5.1 \{ \( \text{while}(1) \) \}
   \{ if \( (\text{marked}(u_j) = \text{TRUE} \) \( \forall u_j \) of \( \text{current\_node} \) )
     \{ if \( (\text{PSP\_number} = 0) \) exit
     \} else \{ \text{current\_node} = \text{PSP\_generation\_point} \) with \( \text{PSP\_joint} = \text{current\_node} \) \}
   \} else break; /*\( \text{while}(1) \) of 5.1 */

5.2. If \( (j \neq 0) \) \( \text{start\_pso} = j++ \);
     If \( (\text{current\_node} = \text{place}) \) type = 'P' else type = 'T' ;

5.3. \( \text{while} (\exists \) at least a \( u_j \) of \( \text{current\_node} \) with \( \text{marked}(u_j) = \text{FALSE} \) )
   \{ pick one unmarked \( u_j \) of \( \text{current\_node} \) to find a new PSP;
     \text{marked}(u_j) = \text{TRUE} ; \text{PSP\_generation\_point} = \text{current\_node} ;
     /* Generate a new PSP */
     \} while (both the indegree & outdegree of \( \text{output\_node} \) == 1)
   \{ insert \( \text{output\_node} \) into the new PSP :
     \text{checked} (\text{output\_node}) = \text{TRUE} ; \text{output\_node} = \text{output\_node of output\_node} \} /* PSP
generation */
   \text{PSP\_joint} = \text{output\_node} ; \text{current\_node} = \text{PSP\_joint} ; \text{PSP\_number} = j++ \} /*\text{while} of 5.3 */

5.4. \( \text{end\_pso} = \text{PSP\_number} ; \)

5.4.1. \{ \( \text{for} (k = 0; k < \text{start\_pso}; k++) \) \}
   \{ If \( (\text{ckck}(k)) \) /* If \( (\Pi_k \in \text{a path from end\_pso\_joint to start\_pso\_generation\_point}) \) */
   \} for (\( i = \text{start\_pso} ; i < \text{end\_pso} ; i++ \) )
   \text{S}[k][i] = \text{S}[i][k] = \text{CL} ;
   else if \( (\Pi_k \in \text{another path from start\_pso\_generation\_point}) \)
Program List 1. Basic S-Matrix Finding Algorithm

The function check() used in Program List 1 is given in Program List 2. It should be noted that the two functions, trace_forward (node1, node2) and trace_backward (node1, node2), are used. The former returns 'TRUE' if
(a) tracing forward from node1, node2 can be found; and
(b) tracing forward from node1, node1 itself cannot be found without node2 on the tracing path.

The latter returns 'TRUE' if
(a) tracing backward from node2, node1 can be found; and
(b) tracing backward from node2, node2 itself cannot be found without node1 on the tracing path.

{if type == 'P' S[i][k] = S[k][i] = 'EX'
else S[i][k] = S[k][i] = 'CN'
else {find \( \Pi \), with \( \Pi \) . generation_point = = start_psp . generation_point
for (i = start_psp; i < end_psp; i++)
S[k][i] = S[i][k] = S[x][k] ;/* last else */
} /* for k */
for (\( \alpha = \) start_psp to end_psp)
for (\( \beta = \) start_psp to end_psp)
S[\( \alpha \)][\( \beta \)] = S[\( \beta \)][\( \alpha \)] = 'CL';

5.4.2 current_node = PSP . generation_point} /* while(1) of line 5 */

{(if (trace_forward (n1, \( \Pi \) . generation_point,) == 'TRUE' ) && (trace_backward (\( \Pi \) . joint, n2) == 'TRUE'))
return(TRUE) /* \( \Pi \) is on a directed path from \( n_2 \) to \( n_1 \) */
else return(FALSE)} /* check */
Program List 2. Function check() Called in Program List 1

The algorithm starts by picking one PSP containing an input place of an initially enabled transition, continues finding new PSPs and updating the S-Matrix. At any stage, we find new PSPs by picking a path whose two end nodes have been traced. The relationship between each PSP on the newly picked path and a traced PSP depends on the relative locations of the generation (starting) point and the joint (end) for this traced PSP with respect to the starting and the ending nodes of the new path. It is easy to find the structural relationship between the PSP on the newly picked path (called a new PSP) and the PSP's near the starting or the ending nodes of the new path. Based on these relationship, the structural relationship between a traced PSP far away from the newly picked path to the latter can be inferred from the existing relationship. We have the following lemma proving the correctness of the S-Matrix finding algorithm without SME, Steps 1 to 5.4.2.

Lemma 1. If all output_nodes of current_node have been marked and current_node is the generation_point of \(\Pi_0\), then all nodes have been checked.

Proof: In the S-Matrix Finding Algorithm, the depth-first search is applied. The searching for PSP and checking the nodes with indegree = outdegree = 1 goes as deep as possible. When it cannot go any further, it backtracks to the generation point of any PSP under consideration for searching, \(g'\). The depth-first searching keeps going from \(g'\). Further backtracking may occur to the nodes in the path from \(g'\). Only when all the searching from all the nodes, other than \(g'\), in this path have been conducted, the procedure backtracks to \(g'\) again. In addition, only when there is no more unmarked output node from \(g'\), \(g'\) is checked. Hence when the procedure backtracks to the generation point of \(\Pi_0\) and there is no more output nodes unmarked, all the nodes in the PN have been checked. \(\Box\)

Lemma 2. Function check can determine whether a PSP is on the path from the joint of ending PSP to the generation point of starting PSP of the current path.

Proof: By the aforementioned property of functions trace_forward and trace_backward, if a PSP, \(\Pi_i\), is such that \(\text{trace_forward}(n_i, \Pi_i, \text{generation_point}) = \text{trace_backward}(\Pi_i, \text{joint}, n_i) = \text{TRUE}\), it is obviously on the path from \(n_i\) to \(n_s\).

Lemma 3. The structural relationship between the PSPs \(\Pi_i\) on the current path, from start_psp to end_psp, and any other PSP \(\Pi_j\) which is (1) not 'CL' to these PSPs, and (2) not in another path from the start_psp, is the same as that between \(\Pi_i\) and the PSP \(\Pi_k\) whose generation point is the same as the generation point of the start_psp, i.e., \(S(i, j) = S(j, k)\).

\[ \Pi_k, \text{generation_point} = \Pi_{\text{start_psp}, \text{generation_point}} \]

Proof: As shown in conceptual Petri net in Fig. 10, the PSPs satisfying the above two conditions are on \(P\). Without loss of generality, the generation
points of the first psp of P_a and the current
start_psp are a transition and a place, respectively.
The structural relationship between the PSPs in the
path from start_psp to end_psp and the PSPs in P_a
is the same as that between \( \Pi_x \) whose generation
point is the same as the generation point of
start_psp and the PSPs in P_a. \( \square \)

Lemma 4. Step 5.5.1 of the S-Matrix Finding
Algorithm is correct.
Proof: Depending on relative position of \( \Pi_i \) with
respect to the starting (\( n_s \)) and ending (\( n_e \)) nodes of
a path, there are three cases:

1) Both the generation point and joint of \( \Pi_i \)
are on a directed path from \( n_s \) to \( n_e \). As
shown in Fig. 10, paths \( P_a, P_s, P_r, \) and \( P_b \)
belong to this case. Because the PSPs in \( P_a \)
satisfy the two conditions of Lemma 3, hence
the structural relationship between the PSPs of
\( P_a \) and those in the current path \( P_c \), by
Lemma 4, is the same as that between the
PSPs in \( P_a \) and the PSP whose generation
point is the same as the generation point of
the start_psp. The PSPs in \( P_s, P_r, \) and \( P_b \)
are cyclic to the PSPs of the current path.

2) Both the generation point and joint of \( \Pi_i \)
are on a directed path from \( n_s \) to \( n_e \). The
PSPs of this case are as shown in \( P_a \) and
current_path of Fig. 10. The PSPs in \( P_a \) are
either concurrent or exclusive to the PSPs in
the current path, depending on whether the
generation point of the starting PSP of this
path is a transition or place. The PSPs of the
current_path are all cyclic to each other.

3) The generation point \( \Pi_i \) is on a directed
path from \( n_s \) to \( n_e \) and the joint \( \Pi_i \) is on a
directed path from \( n_e \) to \( n_s \). The PSPs of this
case are as shown in the \( P_a \) of Fig. 10. \( P_a \)
has not been traversed yet. Hence the S-
Matrix entries regarding this path is to be
determined later. When the program begins
to traverse this path, this path becomes the
current path.

Step 5.4.1 determines the ‘CL’ PSPs (part of
Case 1) first and then determines the ‘EX’ or
‘CN’ PSPs for the PSPs on another paths
starting from the generation point of the start_psp
of the current path (part of Cases 2 and 3), by the
definition of exclusiveness and concurrency
between two PSPs. For the PSPs not belonging to
these two categories (the remaining part of Case 1),
the entries are the same as those with a PSP whose
generation point is the same as the generation point
of the start_psp of this new path. This is true
because those PSPs are on the path started from
another node cyclic to the current path. Finally, the
relationship between the PSPs on the current path
(the remaining part of Case 2) are all ‘CL’, by
the property of depth first search. The property of
depth-first search guarantees that all the PSPs are
traversed. Hence all the S-Matrix entries are
computed correctly. \( \square \)

Theorem 1. The S-Matrix finding algorithm is
correct.
Proof: Initially, in Step 1, all the nodes are not marked nor checked. In Step 2, a home place which is not on the synchronizing cycle is selected. In Step 3, the PSP containing the home place is found, using the method in Step 5.4, as \( \Pi_0 \). In Step 4, the current node is made the joint of \( \Pi_0 \). Step 5 is a while loop which guarantees that all the transitions and places in the PN are visited/checked. Step 5.1 is based on the nature of depth-first approach, which goes as deep as possible and; and when it cannot go any further, backtrack to the generation point of the current PSP and do the depth-first search for PSPs from there. Step 5.2 is used to determine the 'CN' and 'EX' entries, for the PSPs between the current path and the path started from the same node, in the S-Matrix to be used in Step 5.4.1, for the structural relationship between the PSPs in the new path and those in the existing path started from the same node. Step 5.3 contains two nested while loops. Function marked is used to mark the output node(s) from the current node. If all the output node from a node are marked, the backtracking to the generation point of the PSP this node belongs to is executed. The inner while loop is used to search for the nodes in the same PSP, of which, both the indegree and outdegree of the nodes other than the generation point and the joint are ones, per the definition of pseudoprocess. The depth-first search approach is implemented because (1) the PSP is formed starting from the generation point, and (2) each time the joint of the PSP is found, this joint becomes the generation point of the PSP including an unmarked output node of this node as the second node of the PSP. This series of depth-first searching continues until a marked node is encountered because, by Lemma 1, if a marked node is encountered, all its output node(s) are marked. The end_PSP is the last PSP of this path (series of depth-first PSP searching), as conducted in Step 5.4. The correctness of Step 5.4.1 is proved in Lemma 4. Step 5.4.2 is for backtracking to the generation point of the current PSP, before starting another depth first search of the PSPs. The algorithm continues searching, in the depth first sense, for PSPs until all output nodes of current_node have been marked and current_node is the generation_point of \( \Pi_0 \) because, by Lemma 2, all the nodes in the PN are marked. The correctness of the algorithm is thus proved.

Let \( n \) and \( m \) be the number of nodes and PSPs in the PN, respectively. The time complexity of this algorithm, as proved in the following lemma, is \( O(n^3) \).

Lemma 5. The time complexity of the S-Matrix Finding Algorithm is \( O(n^3) \).

Proof: The number of PSPs in the PN, \( m \), is \( O(n) \). The number of execution time of Step 5.3 is \( O(n^2) \). Step 5.4.1 is executed \( O(m^2) \) times. The while loop of Step 5 is executed \( O(n) \) times. Hence the total time complexity of this algorithm is \( O(n^3) \). 

Assuming that the SMEs are synchronized at the control transition in each one of the sequential exclusive processes, we develop the enhancing part of S-Matrix Finding Algorithm for SME as...
The enhancing part of S-Matrix Finding Algorithm is given in the following:

/* Detect whether there exists sequential exclusive 'SX' between PSPs */
6. Find all VPs which satisfy the following conditions:
6.1. They share a common generation point which is a place, denote such a place as CGP;
6.2. Find the joints (which must be transitions) of all these VPs, denoted as t's.

Divide them into a serial of subsets, these transitions in each subset must be cyclic to each other and in some elementary cycle.

Merge two subsets into one if they share a common transition:
6.3. For each subset, find the subnet by deleting all PSPs from the net which are not in some elementary cycles connecting the transitions in the subset;
6.4. This subnet must be live, bounded and reversible;
7. Let \( t_1, t_2, ..., t_k \) be the set of transitions in order in the subsets;

For each \( t_i \) (\( i = 1, 2, ..., k \)), repeat the following steps:

7.1. Find an input PSP \( \Pi_i \) with \( \Pi_i . generation \_point \) = CGP and output PSP \( \Pi_f \) and a PSP \( \Pi_e \) ∈ subnet with

\[
\Pi_e . generation \_point \Rightarrow t_i \text{ such that } S[i][e] = 'CL' \text{ and } S[i'][e] = 'CL' \text{ or 'CN'};
\]

7.2. \( \Pi \_number = \Pi i'; Y_i = \{ \Pi_i, \Pi_r \}; \) while (\( \Pi \_number = 'CL' \) \( \Pi_r \))

\[
\{ Y_i = Y_i + \Pi \_number; \text{number} = \text{number} + 1 \}
\]

7.3. For all PSPs \( \Pi_i \) and \( \Pi_j \) such that \( \Pi_i \in Y_i \) and \( \Pi_j \in Y_j \),
If \( S[i][i] = 'CN' \) then \( S[i][i] = S[i'][i] = 'SX' \);

Program List 3. The Enhancing part of S-Matrix Finding Algorithm for SME

The correctness of Program List 3 is proved in the following theorem:

**Theorem 2.** The algorithm of updating the S-Matrix for SMEs is correct.

**Proof:** Steps 6.1, 6.2, and 6.3 find all the synchronizing cycles, which are live, bounded, and reversible by themselves, for the corresponding SMEs. Step 7 first finds the control transition in each of the sequential exclusive processes. Using Step 7.1, for each sequential exclusive process \( i \), the VP from the decision place to the control transition \( t_i \), \( \Pi_{i1} \) and the next PSP, \( \Pi_{i2} \), are found since \( \Pi_{i1} \) 'CL' \( \Pi_e \) and \( \Pi_{i2} \) 'CN' \( \Pi_e \). In Step 7.2, \( \Pi_{i1} \) and \( \Pi_{i2} \) are first included in process \( i \). In the second part of Step 7.2, all the other PSPs, \( \Pi_{i3}, \Pi_{i4}, ..., \), are included in process \( i \) because of the use of depth-first approach and backtrack property in the S-Matrix finding.
algorithm, and the fact that the increase of PSP numbering in each of the sequential exclusive processes in the SME stops at the common ending place. Using Step 7.3 and the fact that the $\Pi_{i1} \in$ process $Y_i$ and the $\Pi_{j1} \in$ process $Y_j$ of the same SME are exclusive to each other, the 'SX' relationship among the PSPs of different processes of the SME can be determined. The correctness of the algorithm is thus proved.

The corresponding time complexity is given in the following Lemma 6.

**Lemma 6.** The time complexity of the detection of SME is $O(n^2)$.

**Proof:** Select a joint $t_j$ of a VP and find the elementary cycle the joint is in. Trace along the cycle starting from the above joint, find the first transition $t_2$ which is a joint of another VP with the same CG. Continue this tracing, we find all $t_i$'s in the cycle which are the joints of VPs with the same CG. The total time complexity involved in this step is $O(n^2)$. It takes $O(n)$ amount of time to find $Y_i$ of $t_i$. Therefore, it takes totally $O(n^2)$ of time to find all $Y_i$'s. It takes $O(n^2)$ of time for the update of entries from 'CN' to 'EX'. Hence the total time complexity of this update of entries due to SME is $O(n^2)$.

The above algorithm is included in the S-Matrix searching algorithm so that the obtained S-Matrix has 'SX' entries. Hence, the S-Matrix helps identify PMEs and SMEs. If the entry values in the S-Matrix corresponding to a set of PSPs are 'EX' ('SX'), then these PSPs are part of a SME (SME). Given this S-matrix, we can find the B-places and C-places. Based on their markings, we can verify whether the net is deadlock-free or not assuming the same PN as the one in [ZHO 91].

**Example of the S-Matrix Search Algorithm**

The S-Matrix for the PN in Fig. 3(a) is shown in Fig. 3(f). An example of the application of this algorithm is described as follows:

1. In the initial marking, $t_1$ is enabled. Hence, pick $p_1$ as a home place. The PSP containing $p_1$ is $\Pi_0 = [t_5 t_7 t_1 t_1 t_3 t_2 t_3]$. Mark $t_7$ and check $p_8$ since $t_7$ is the only output node to $p_8$. The current node is $p_3$.

2. $t_3$ and $t_4$ are two output nodes of $p_1$. Pick $t_3$, $t_3$ is not a simple node (which has only one input and only one output node). Hence $\Pi_1 = [p_3 t_3]$ and Mark $t_3$.

3. $p_4$ and $p_6$ are two output nodes of $t_3$. Pick $p_6$, which is a simple node. Its output node $t_5$ is not simple. Hence $\Pi_2 = [t_5 p_6 t_3]$. Mark $p_6$.

4. The output node $p_8$ of $t_5$ is not simple. Hence $\Pi_3 = [t_5 p_8]$ and end_psp = $\Pi_3$. Mark $p_8$ and check $t_5$. $p_8$ has been checked. Hence $S[i][j] = 'CL' \; \forall \; i, j \in \{0,1,2,3\}, i \neq j$.

Backtrack to $t_5$ which has been checked. So backtrack again to $t_3$ and pick its unmarked output node $p_7$ which is a simple node. Mark $p_7$. Its output node $t_4$ is not a simple node and $\Pi_4 = [t_3 p_7 t_4]$. Check $t_4$ since all its output arcs have been traced.

5. Pick an untraced arc $t_4$ $p_9$ and find a new
\[ \Pi_5 = [t_4 \ p_9 \ t_3]. \] t_5 has been marked, so update the S-Matrix. \[ \text{start}_{\text{ps}} = \Pi_4. \]

\[ \text{end}_{\text{ps}} = \Pi_5 \ o \ (\Pi_4, \ Pi_5) \ o \ (t_1, \ Pi_3) \ o \ (\Pi_0, \ Pi_1, \ Pi_4) \ o \ Pi_6. \] Mark p_9.

6. Pick an untraced arc \( t_4p_7 \) and find a new \( \Pi_6 = [t_4 \ p_7 \ t_6 \ p_8]. \) \( \Pi_6 \ o \ (\Pi_5, \ Pi_2, \ Pi_3), \)

\[ (\Pi_0, \ Pi_1, \ Pi_4) \ o \ Pi_6. \] Mark p_7.

7. Pick an untraced arc \( t_4p_4 \) and find a new \( \Pi_7 = [t_4 \ p_4 \ t_3]. \) Hence \( (\Pi_0, \ Pi_1, \ Pi_2, \ Pi_3, \ Pi_4) \ o \ Pi_7, \) and \( (\Pi_5, \ Pi_6) \ o \ Pi_7. \) Mark p_4 and check t_4.

8. Backtrack to p_3. Pick the remaining untraced arc \( p_3 \ t_4; \) \( \Pi_8 = [p_3 \ t_4] \) and \( t_4 \) has been marked and is on a circle. \( \Pi_8 \ \text{EX} \) \( (\Pi_0, \ Pi_4), \) and \( \Pi_8 \ \text{CL} \)

\[ (\Pi_0, \ Pi_2, \ Pi_3, \ Pi_5, \ Pi_6, \ Pi_7). \] Check p_3.

9. VPs \( \Pi_1 \) and \( \Pi_8 \) share a common generation point, \( p_3, \) and have joints \( t_3 \) and \( t_4, \) respectively. \( \{\Pi_4, \ Pi_7\} \) is a live, bounded and reversible sub-net of the PN.

10. Since \( \text{S}[1][4] = \text{CL} \) and \( \text{S}[2][4] = \text{CL}, \) \( Y_1 = \{\Pi_1, \ Pi_2\}. \) Similarly, \( \text{S}[8][7] = \text{CL} \) and \( \text{S}[5][7] \neq \text{CL}, \)

\[ Y_2 = \{\Pi_8, \ Pi_3\}. \]

11. \( Y_1 = \{\Pi_1, \ Pi_2, \ Pi_3\} \) and \( Y_2 = \{\Pi_8, \ Pi_5, \ Pi_6\}. \) \( \Pi_1 \ \text{EX} \)

\( \Pi_8, \Pi_{1\_\text{generation\_point}} = \)

\( \Pi_8, \Pi_{1\_\text{generation\_point}} = p_8, \) and thus 

\( (\Pi_1, \Pi_2, \Pi_3) \ \text{SX} \) \( (\Pi_8, \Pi_5, \Pi_6). \)

12. All PSPs are found and stop.

V. The Knitting Technique

Zhou et al [ZHO 91] developed the DFC to upgrade current PN analysis for shared resources. However, they did not explicitly apply the DFC to synthesis as we shall in this section. We will explain how to synthesize PMEs and SMEs. As a result, the DFC is generalized. Here we present the guidelines for the rules used in the knitting technique followed by examples. In the next section we apply the knitting technique to synthesize PMEs and SMEs and extended generalized DFCs to arbitrary paths.

To understand the knitting technique in an intuitive fashion, we present the guidelines for generating new arcs.

A. Guidelines for Synthesis

Given a synthesized net, \( N^1, \) a set of new paths are generated using some synthesis rules to form another net, \( N^2. \) The marking in the NP upon the generation is called its initial marking. The synthesis rules should be such that all transitions and places in \( N^2 \) are also live and bounded. This can be ensured by

1) Inhibiting any intrusion into normal transition firings of \( N^1 \) prior to the generations. This guarantees that neither unbounded places nor dead transitions would occur in \( N^1 \) since it was live and bounded prior to the generations.

2) Inhibiting any dead transitions and unbounded places in the NP.
There are two kinds of intrusion. One is to change the marking of \( N_1 \): the other is to eliminate some reachable markings. In order for \( NP \) to be live, it must be able to get enough tokens from \( p_g \) or by firing \( t_g \). When tokens in \( NP \) disappear, the resultant marking of the subnet \( N_1 \) in \( N_2 \) must be reachable in \( N_1 \) prior to the generation of \( NP \). Further, the marking of \( NP \) must be reversible in \( N_2 \).

The second intrusion may occur when the joint is a transition which may not be enabled in case of backward \( TT \) generation; hence causing some markings in \( N_1 \) unreachable. Thus, if the joint is a transition, the \( NP \) must be able to get tokens from a generation point within each iteration.

Based on the concept of no intrusion, the rules are constructed according to the following guidelines:

1) From \( M_0 \), each \( t_g \) must be potentially firable (always or has fired \( \forall \) firing sequence \( \sigma \) in \( N_1 \) enabling the joint which is a transition).

2) (a).These tokens must disappear from \( NP \) before it gets unbounded tokens, and (b).return to \( N_1 \) to reach a marking in \( M_2 \). The marking of the subnet \( N_1 \) in \( M_2(N_1) \) of all possible sets of such \( M_2 \) are identical to those in \( N_1 \) without the new paths.

Guidelines (1) and (2) guarantee no intrusion. Guideline (2) guarantees no dead transitions and no unbounded places in the \( NP \) (since all tokens can disappear from the \( NP \)). This paper only deals with ordinary \( PN's \); the rules for general \( PN's \) where arcs may carry multiple weights are discussed in [CHA 93c].

To relieve the reader from the burden of understanding the complete set of rules, we strip down the concepts to a bare minimum and rely on one example to explain the necessary terminology and rules. To simplify the discussion, we first present the terminology, rules, and examples for non-SMEs followed by the SMEs.

**B. Rules for Non-SMEs**

This section provides the terminology, rules and an example of synthesized PNs with no SMEs.

Definition (Local Exclusive Set) : A local exclusive set (LEX) of \( \Pi_j \) with respect to \( \Pi_k \), \( X_{ik} \), is the maximal set of all PSPs which are exclusive to each other and are equal to or exclusive to \( \Pi_j \), but not to \( \Pi_k \). That is, \( X_{ik} = LEX(\Pi_j, \Pi_k) = \{ \Pi' : \Pi' \neq \Pi_j \text{ or } \Pi_k \} \), \( \forall \Pi_1, \Pi_2 \in X_{ik} \).

There is a general rule:

An isolated basic process can be generated at any time.

The TT- and PP-Rules are listed as follows:

**TT Rules** :

For a \( \Pi \) from \( t_g \in \Pi_g \) to \( t_j \in \Pi_j \) generated by the designer,

1) TT.1 If \( t_g \cup t_j \) and \( t_g \cap t_j = \emptyset \) or only one
of them is in a circle which was solely generated using Rule PP.1,
then display "Disallowed, delete this PSP and generate another PSP" and return.

2) If (t_k \leftrightarrow t_i or t_k = t_j and without firing t_i, there
does not exist a firing sequence \sigma to fire t_k,
then insert a token in a place of \Pi'.
If \Pi_g = \Pi', then display "You may
generate another \Pi" and return.

3) TT.3
a) TT.3.1 Generate a TP-path from a
transition t_k of each \Pi_g in X_g to a place
p_k in the \Pi'.
b) TT.3.2 Generate a virtual PT-path from
the place p_j to a transition t_i of each \Pi_j
in X_g.

4) TT.4 If there was no path from t_k to t_i prior
to this generation or both are in a circle
which was generated using the PP rule, then
(1) generate a new TT-path from t_k to t'_i such
that t_k, t_i, t'_k, and t'_i are in a circle, (2) apply
Rule TT.3, and (3) Go to step 2.

**PP Rules:**
For an \Pi' from p_k \in \Pi_g to p_j \in \Pi_j generated
by the designer,

1) PP.1 If p_k \parallel p_j, then display "Disallowed,
generate another \Pi" and return.

2) PP.2
a) PP.2.1 Generate a TP-path from a
transition t_k of \Pi' to a place p_j of each
\Pi_j in C_g.

b) PP.2.2 Generate a virtual PT-path from
a place p_k of each \Pi_g in C_g to the
transition t_k of \Pi'.

In [CHA 93c], we showed that some
synthesized nets using this approach are beyond
the "assymmetric-choice nets" [MUR 89].

**An Example**

See Fig. 4, initially, using the general rule,
we generate an isolated/only basic process, [p_1 t_1 p_2
t_2 p_3 t_3 p_4 t_4 p_5 t_5 p_6]. Then another basic process,
[p_11 t_11 p_12 t_12 p_13 t_13 p_14 t_14 p_15 t_15 p_16 t_16 p_17 t_17 p_11],
is added, using the general rule. Then, by Rule
PP.1, the following four PSPs are generated: [p_2 t_6
p_7 t_7 p_4], [p_3 t_8 p_6 t_9 p_5], [p_12 t_18 p_18 t_19 p_19 t_20 p_12],
and [p_12 t_21 p_20 t_22 p_21 t_23 p_22 t_24 p_23 t_25 t_17]. Two
backward TT-paths, [t_9 p_8 t_8] and [t_23 p_25 t_26 t_27
p_27 t_21], is then generated with a token respectively
added to p_8 and p_26, by Rule TT.2. A pair of
forward- and backward-generations, [p_15 t_30 p_10]
and [p_16 t_28 p_13], are then generated. They form a
circle. Another three forward TT-paths, [t_13 p_28 t_33
p_29 t_13], [t_13 p_33 t_14] and [t_23 p_24 t_24], are generated.
Using Rule TT.3, PSP [t_1 t_30 t_13] and VPs [t_2 p_30],
[p_30 t_18], and [p_30 t_31] are generated. By Rules TT.4
and TT.3, PSP [t_14 p_31 t_4] and VPs [t_19 p_31], [t_25 p_31],
and [p_31 t_6] are generated. Using Rule PP.1, PSP
[p_16 t_29 t_25] is generated. Using Rule PP.2, PSP [p_29
p_31 t_32 t_22] and VPs [p_14 t_31], [p_33 t_31] and [t_32 t_24]
are generated. It should be noted that, during the
course of any generations, the PN in Fig. 4 is well-
behave, that is, the PN is live, bounded, reversible, and conservative.

The following observation is also in [CHA 93c].

Observation 1: Every circle of a synthesized net contains at most one marked place.

We explain this observation below. In the basic process, only the home place holds tokens. Afterwards, only a backward TT-generation would add tokens to the newly formed cycles. If there is no backward TT-generation, the observation holds. Otherwise, multiple marked places in a cycle could only occur when we add tokens upon a backward TT-generation to the $\Pi^w$ and another PSP in the circle already holds tokens. This is not possible because prior to the generation, $t_5$ ($t_j$) is enabled. An example of this observation is shown in Fig. 4.

Another example shows the use of knitting technique to synthesize a PME easily is given as follows:

Example of Synthesizing PME: As shown in Fig. 1. We can first generate a basic process, $[p_1 t_1 p_3 t_3 p_3 t_1]$, followed by a backward TT-generation, $[t_3 p_3 t_1]$ with a token added to $p_3$. We then generate a PP-path, $[p_3 t_2 p_4 t_4 p_4]$, followed by another backward TT-generation, $[t_4 p_2 t_2]$, with a token added to $p_2$. The result is Fig. 1.

VI. Synchronization Rule for Generating SMEs

We first define the following terms for the description of synchronization rules:

Definition (Decision Place): A decision place is the common generation place of a set of exclusive PSPs.

Definition (Control Transitions): The control transitions are the output transitions of a decision place.

For example, in Fig. 3(e), $p_3$ is a decision place and $t_3$ and $t_4$ are control transitions.

The synchronic distance is a concept closely related to the degree of mutual dependence between two events in a condition/event system. The definition is as follows:

Definition (Synchronic Distance): The synchronic distance between two transitions $t_1$ and $t_2$ in a PN $(Z, M_0)$ is defined by $d_{t_1 t_2} = \text{Max} \{ \sigma(t_1) - \sigma(t_2), \sigma \in L(M) \}$, where $\sigma(t_i), i = 1, 2,$ is the number of times transition $t_i$ appears in $\sigma$. $L(M)$ is the set of all firing sequences for the net with the initial marking $m \in R(M_0)$. The synchronic distance between $\Pi_1$ and $\Pi_2$ follows that between $t_1$ in $\Pi_1$ and $t_2$ in $\Pi_2$.

For example, consider the net shown in Fig. 2(d) without the two paths: $[t_4 p_1 t_1]$ and $[t_2 p_3 t_3]$. $d_{t_4} = \infty$. Note that the above synchronic distance is similar to the token capacity concept in [ZHO 91] or the deviation bound in [SIL 87].

Now a TT-path, $[t_4 p_2 t_2]$, is generated from $t_4$ in $\Pi_2 = [p_6 t_3 t_4 p_4 t_6]$ to $t_1$ in $\Pi_1 = [p_6 t_1 p_2 t_6]$. $\Pi_1$ and $\Pi_2$, as shown in Fig. 2(d). This is referred to as "exclusive TT interaction" in [YAW 87,88a,88b,89] which forbids such a generation for...
the following reason. Each time \( t_4 \) fires, it injects a token into \( p_1 \). After an infinite number of such firings, \( p_1 \) gets unbounded. Therefore, a single TT-path generated between these two exclusive transitions with synchronic distance \( d = \infty \) is insufficient.

The two alternative PSPs, however, can still be synchronized by generating a path with a token, \([t_2; p_3; t_3]\), from \( t_2 \) to \( t_3 \). The resulting PN is bounded, live and reversible. Note that without the token in \( p_1 \), the net becomes dead.

For exclusive processes, synchronization paths must be generated between the control transitions of the exclusive PSPs. Summarizing the above discussion, we have the following rule for synchronization:

**Synchronization Rule**: If a TT-path is generated from \( t_g \) to \( t_j \), \( t_k \) and \( t_j \) is a control transition, then generate another TT-path from \( t'_{g} \) to \( t'_{j} \) such that if \( t_{j} \) is control transition then \( t'_{j} \) is also a control transition with the same decision place.

**VII. Rules for Generations Involving SMEs**

This section discusses rules for nets with SMEs. Since SME manifests a new structure, new rules are required for TT generations within an SME and among SMEs to maintain the completeness of the knitting rules. There are three cases of TT-generations, namely, within an SME, between two SMEs and between an SME and an LEX. In addition to 'CN', 'EX' and 'SQ', \( t_5 \) may be 'SX' to \( t_j \) which corresponds to the generation within an SME. For the case of 'EX', we may apply the synchronization rule again and will not be considered further. For 'SX' and 'CN', we can still apply Rule TT.3 between LEX1 and LEX2 without disturbing the well-behavedness. The synchronic distance among PSPs in LEX1 or LEX2 is, however, no longer infinite. We term such LEX1 and LEX2 as SME1 and SME2 respectively. Thus an SME is not only a term indicating a specific behavior, but also a set of PSPs that are mutually exclusive but sequentialized with each other, having a finite synchronic distance. In other words, no process in SME, unlike PSPs in an LEX, can fire infinitely without firing any other processes in the SME. Rule TT.3 may be relaxed. In other words, not all PSPs in SME1 or SME2 may participate in the generation. With synchronic distance \( d = \infty \), this may create deadlocks or unbounded places since \( t_g \) and \( t_j \) may never execute which is not true for finite \( d \).

Relaxing Rule TT.3 creates new rules and also generates more classes of nets, allowing us to uncover more aspects behind well-behaved nets and enhancing the approach of synthesis-based analysis. In the rest of this paper, whether Rule TT.3 is relaxed or not, we uniformly say that the TT-generations are applied from \( X_{g_j} \) to \( X_{g_j} \). Therefore, when Rule TT.3 is relaxed, \( X_{g_j} \) (\( X_{g_j} \)), changing its original meaning somewhat, may not equal SME1 (SME2).

PSPs in both SME1 and SME2 are executed
sequentially and some TT-path within the
generation may be a backward one. We need to
determine whether and how many tokens need to
be added. As a result, a new DFC is discovered.

A. TT-Generations within An SME

In this subsection, we study the first case, i.e.,
the TT-generations within an SME and its
application to resource sharing in an FMS. As a
result, the DFC by Zhou et al is generalized. The
remaining two cases will be investigated in the
next subsection. Again, there are two types of
generations: forward and backward. Similar to
Rule TT.2, backward generation entails tokens
inserted into new. The difference is that insertion
of a single token may not avoid the deadlock. An
equality is shown in Fig. 2(e) where a TT-path is
generated backward from $t_2$ to $t_1$. Tokens must be
inserted into $p_5$ such that $M(d_{r}) \leq M(p_5)$ in
order to avoid deadlocks. For instance, if $M(d_{r}) = 4$ and $M(p_5) = 3$, the system comes to a total
deadlock after the firing sequence $\sigma = t_1, t_2, t_3, t_2, t_1$
$t_1, t_1$, the marking is $(0, 1, 3, 0, 0, 0)$. The deadlock can
be removed by adding one more token to $p_5$ which
enables $t_2$ to fire, removing the token trapped in $p_3$.
In general, the least number of tokens to be
inserted depends on the maximum marking at both
the decision place and $p_5$ which determines $d_{r}$. Using the example in Fig. 2(a), in which $p_D = p_6$, it
is easy to see that the least number of tokens $n^\ast$ (if
$> 0$) to be inserted into $\Pi^\ast$ for $N^2$ to be well-behaved when $t_{r}$ fires later than $t_{i}$ is

$$n^\ast = \text{MAX}(d_{r} - \text{MAX}[M(p_D)] + 1, 0).$$

which is a generalized version of the DFC in [ZHO 91c]. Note that $n^\ast$ corresponds to the marking in $p_8$,$d_{r}$ to that in $p_c$, and MAX[M(p_D)] to that in the shared resource place (i.e., robot).

Calculating $d_{r}$ and MAX[M(p_D)] is, in
general, a reachability problem. It is, however, not
so for synthesized nets because we calculate them
in an incremental fashion. Upon each new
generation, $d_{r}$ is determined when the corresponding SME is first formed and equals the
number of tokens inserted into $\Pi^\ast$. MAX[M(p_D)]
equals the minimum marking of all home places
which are 'SE' to $p_D$ which can be updated
upon each backward TT-generation.

Lemma 7: \[
\text{MAX}(M(p_D)) = \min_{p_{hi} \rightarrow p_D} M(P_{hi})
\]

Proof: MAX(M(p_D)) can be achieved by disabling
all output transitions of $p_D$ and continue firing transitions in the PN until no more transitions are
firable. Only those tokens in home places $p_{hi}$ which
are 'SE' to $p_D$ can flow to $p_D$. Because $p_{hi}$ are
'CN' to each other, the maximum tokens that can flow to $p_D$ is the minimum marking of all such
$p_{hi}$. 

When there are tokens in $\Pi^\ast$, $t_{i}$ can fire as
that in $N^1$ prior to the generation. Because the
generation does not alter input and output sets of
transitions of all places in $N_i$, the reachable
markings of $N^1$ remain the same as that prior to the
generation. Since $N^1$ was well-behaved, so is $N^2$ as
long as there are tokens in $\Pi^\ast$. In order for $N^2$ to
remain well-behaved, we must make sure that $N^2$
be live even if all tokens in \( \Pi^w \) have been consumed by never firing \( t_g \). Thus, the tokens in \( p_D \) always are consumed to fire \( t_f \). After all tokens in \( \Pi^w \) have been consumed, \( t_f \) cannot fire and tokens are trapped in the input place of \( t_f \). When all tokens in \( p_D \) are consumed and get trapped, none of the control transitions \( t_c \) can fire, neither can \( t_g \) to supply tokens to \( \Pi^w \) to fire \( t_f \). Thus, the PN comes to a deadlock and no longer reversible. The above trapping stops when the marking of the C-place turns to zero. Under this situation, if \( p_D \) still has tokens, we can fire one output transition of \( p_D \) to fire subsequently \( t_g \) and to make \( t_f \) firable again. And there are no deadlocks.

Lemma 8: If there are no tokens in \( \Pi^w \), the maximum tokens trapped in the input place of \( t_f \) equals the minimum of \((M(p_D), d_g)\).

Proof: easy to see. \( \square \)

Theorem 3: Assuming new paths have been correctly generated, the least number of tokens inserted into the \( \Pi^w \) for the TT-generation within an SME is shown in Eq. (1).

Proof: Each time \( t_f \) fires once, it consumes one token from \( \Pi^w \), the C-place, and \( p_D \). Because \( N^d \) is reversible, a token generated by the firing of \( t_f \) eventually returns to \( p_D \). Assume \( d_g \geq n^w \), after \( t_f \) fires \( n^w \) times, there will be no tokens in \( \Pi^w \) and the number of tokens of the C-place reduces from \( d_g \) to \( d_g - n^w \). To avoid deadlocks, when all tokens in the C-place are consumed, there must be at least one token in \( p_D \). Thus we have

\[
\text{MAX}[M(p_D)] \geq d_g - n^w + 1
\]

which is equivalent to

\[
n^w \geq d_g - \text{MAX}[M(p_D)] + 1
\]

Because \( n^w \) must be greater than zero, we have

\[
n^w \geq \text{MAX}(d_g - \text{MAX}[M(p_D)] + 1, 0) \quad \square
\]

The condition in Eq. (1) is also a necessary condition as we can assume the contrary and prove that \( N^2 \) is deadlocked. Note that prior to the sequentialization, an SME is an LEX. We first define the full and partial SME as follows:

Definition (Full SME, Partial SME): If all the control transitions of a decision place are 'SX' to each other, the SME is a full SME; otherwise it is a partial SME.

For the TT-generation between an SME/LEX and another SME/LEX, when the SME is partial, that is, the exclusive processes are not all synchronized, there is no ordering of executions of PSPs in the SME and other PSPs corresponding to the same decision place, \( p_D \). As a consequence, \( t_g \) (\( t_f \)) may never fire and thus cause deadlock (unboundedness). Hence, Rule TT.3 must be applied. In the sequel, we assume that all SMEs under consideration are full.

B. TT-Generations from SME1 to SME2

This kind of generations create new DFCs unvisioned in [ZHO 91]. Since PSPs within an SME are exclusive to each other, it seems that Rule TT.3 is still applicable. The only difference is that not all PSPs in the SME may be involved in the
generation. Consider two SMEs, each containing a pair of PSPs with synchronic distance \( d = 1 \). In what follows, we first show, in Lemma 9, that the ratio of firing number of \( t_g \) to that of \( t_j, \frac{\sigma_g}{\sigma_j} \), should asymptotically be 1 after \( t_g \) or \( t_j \) fires infinite times.

The TT-generation must be performed to have enough tokens injected into the NP to fire \( t_j \), to avoid deadlock, and no persistent accumulation of tokens in the NP, to avoid unboundedness. Based on this, we have the following:

**Lemma 9**: After a TT-generation between two full SMEs, the necessary condition for preserving well-behavedness is \( k = \lim_{\sigma_g \to \infty} \frac{\sigma_g - \sigma_j}{\sigma_g} = 0 \). If \( k > 0 \) (\( k < 0 \)), then the synthesized net is unbounded (deadlocked), where \( \sigma_g (\sigma_j) \) is the total number of firings of \( t_g \)'s (\( t_j \)'s).

**Proof**: If \( k > 0 \), then as \( \sigma_g \to \infty \), infinite number of tokens are accumulated in the NP, the synthesized net is unbounded. Similarly, if \( k < 0 \), then the NP needs to have infinite tokens in the initial marking to support \( t_j \)'s to fire infinite number of times; implying that the synthesized net has a deadlock. Hence the necessary condition for preserving well-behavedness is \( k = 0 \).

We then discuss the following two cases of synchronic distances among PSPs in an SME. (1) \( d = 1 \) and (2) \( d > 1 \).

**Case 1**: \( d = 1 \)

In this case, PSPs in an SME execute sequentially and alternatively. We let \( \tau_g (\tau_j) \) be the total number of firings of PSPs in SME1 (SME2) and \( a \) and \( b \) be the number of PSPs in SME1 and SME2, respectively.

Based on Lemma 9, we have the following theorem.

**Theorem 4**: For TT-generations from SME1 to SME2, if neither \( a \) nor \( b \) is a factor of the other, then Rule TT.3 cannot be relaxed.

**Proof**: Consider the TT-generation from a PSP, \( \Pi_g \), in SME1 to \( r \) PSPs in SME2 and infinitely large \( \tau_g = \tau_j \) which is divisible by both \( a \) and \( b \). Then, because the synchronic distance of PSPs in each SME is one, \( \Pi_g \) fires \( \sigma_g = \frac{\tau_j}{a} \) times and the total number of firings of the \( r \) PSPs in SME2 is \( \sigma_j = r * \frac{\tau_j}{b} \). We have \( k = \lim_{\sigma_g \to \infty} \frac{\sigma_g - \sigma_j}{\sigma_g} = \frac{\tau_j}{a} \left( 1 - \frac{r}{b} \right) \neq 0 \) for any integer \( r \) since neither \( a \) nor \( b \) is a factor of the other. Because the above \( \tau \) and \( \sigma \) are infinitely large, finite changes of \( \tau \) would not alter the above conclusion of \( k \neq 0 \). Thus, we have to apply Rule TT.3 to preserve well-behavedness. The same conclusion applies when the TT-generations are from arbitrary \( n_g \) PSPs in SME1 to arbitrary \( n_j \) PSPs in SME2.

Thus, Rule TT.3 can only be relaxed when \( a \) or \( b \) is a factor of the other. Otherwise, it has to be applied to all PSPs in SME1 and SME2; i.e., \( X_{\Pi} = \text{SME1} \).
and \( X_{g'} = \text{SME2} \). As will be shown later, this theorem also holds for the case when \( d > 1 \). The proof of Theorem 4 also leads to the following:

**Theorem 5** : If \( \frac{b}{a} \) is an integer, then applying Rule TT.3 by making \( X_{g'} = \Pi_x \) and \( X_{g'} \) a set containing arbitrary \( \frac{b}{a} \) PSPs of SME2 preserves the well-behavedness.

**Proof**: By making \( r = \frac{b}{a} \) of Theorem 4, we have \( k = 0 \).

Similarly, we have the following corollary:

**Corollary 1** : If \( b \) is a factor of \( a \), then applying Rule TT.3 to \( \Pi_j \) of SME2 and a set containing arbitrary \( \frac{a}{b} \) PSPs of SME1 preserves the well-behavedness.

In the above discussion, we consider only the case of \( X_{g'} \) being only a PSP in SME1. We can apply such generation a number of times so that \( X_{g'} \) can have more than one PSP. Thus, for the rest of this paper, all theorems and lemmas will only concern with such a generation.

Note that tokens may have to be inserted in the NP to avoid deadlock. This happens when a PSP in \( X_{g'} \) is about to execute but there is no token in the NP to support such an execution because the PSP in \( X_{g'} \) fires too late to supply tokens. The number of tokens to be inserted into the NP depends on the \( X_{g'} \) and \( X_{g} \) selected. The following lemma helps one develop procedure token Added().

**Lemma 10** : If no tokens need to be inserted into \( \Pi^w \) generated from \( \Pi_{g'} \) to \( X_{g'} = \{ \Pi_{s_x} \} \), \( x = 1, 2, ..., r \), then \( \forall x, j_x > (x - 1) \cdot a + y \) for the case of \( r = \frac{b}{a} \) being an integer.

**Proof**: In order to have no insertion of tokens, every \( x \)-th execution of \( \Pi_{g'} \) must occur earlier than \( \Pi_{s_x} \) of \( X_{g'} \). The \( x \)-th execution of \( \Pi_{g'} \) coordinates with the execution of \( \Pi_{s_x} \) to complete one iteration where \( j_x = (x - 1) \cdot a + y \). If \( j_x > j'_x \), then \( \Pi_{s_x} \) executes later than that of \( \Pi_{g'} \), and hence, later than the \( x \)-th execution of \( \Pi_{g'} \).

For the case of \( r = \frac{a}{b} \) being an integer, similar results can be obtained. Let \( X_{g'} = \{ \Pi_{g'} \mid y = 1, 2, ..., r \} \) and \( X_{s_y} = \{ \Pi_{s_y} \} \). Again PSPs in \( X_{s_y} \) must execute earlier than \( \Pi_{s_y} \) which is just the reverse of the case of \( b > a \). That is, \( g_y < g'_y \) (note the sign changes from \( > \) to \( < \)). All the rest discussions remain the same as those in Lemma 4.

**Lemma 11** : If no tokens need to be inserted into \( \Pi^w \) generated from \( X_{g'} = \{ \Pi_{g'} \mid y = 1, 2, ..., r \} \) and \( r = \frac{b}{a} \) to \( X_{s_y} = \{ \Pi_{s_y} \} \), then \( \forall x, g_y < (y - 1) \cdot a \)
+ x for the case of \( r = \frac{b}{a} \) being an integer.

Procedure Token_Added() for \( b > a \) is presented below. Let \( X_{gj} = \{ \Pi_{gj} \} \) and \( X_{jg} = \{ \Pi_{jg} \} \), \( x = 1, 2, \ldots \). We scan \( j_x \) in the increasing order of \( x \). Each time the deficiency of a token is detected, the integer variable "Token_Added" is incremented by one. At the end of the procedure, "Token_Added" represents the total number of tokens to be added.

```
Token_Added (Xgi, Xjg) {
    Token_Added = 0;
    for {x = 1; x < r + 1; x++}
    {
        if (\( j_x < ((x - 1 - Token\_Added) \ast a + y) \))
            Token\_Added++;
    }
}
```

Similarly, procedure Token_Added() for the case of \( \frac{a}{b} \) being an integer exactly parallels that for the case of \( \frac{b}{a} \) being an integer except the sign change (from \(<\) to \(>\)) in the "if" statement. We prove the correctness of the procedure as follows.

**Theorem 6:** Procedure Token_Added() calculates the minimum number of tokens to be added to maintain the well-behavedness.

Proof: Consider first the case of \( \frac{b}{a} \) being an integer. The proof for the case of \( \frac{a}{b} \) being an integer is similar. The procedure checks each \( \Pi_{\theta} \) starting from \( x = 1 \). If it cannot be executed and the synthesized net comes to a deadlock, then the procedure adds a token to the NP. At the end, the number of tokens added is the minimum. We then prove that the condition to add a token at the \( x \)-th step is \( j_x < ([x - 1 - \text{Token\_Added}] \ast a + y) \) as shown in the procedure where "Token_Added" is a temporary variable indicating the minimum number of tokens to be added at the end of the procedure. It also indicates the number of tokens added to the NP prior to the \( x \)-th step. At the beginning of the \( x \)-th step, we need \( x \) tokens to support the executions of all \( \Pi_{j\theta}, \theta = 1, 2, \ldots, x \). But the NP already has "Token_Added" tokens, thus \( \Pi_{gj} \) must have executed \( (x - 1 - \text{Token\_Added}) \) times earlier than the execution of \( \Pi_{jx} \). The \( (x - \text{Token\_Added}) \)-th execution of \( \Pi_{gj} \) adds a token into the NP to be consumed by the execution of \( \Pi_{jx} \). The former must coordinate with the execution of \( \Pi_{jx} \) to complete one iteration where \( j'_{x} = (x - 1 - \text{Token\_Added}) \ast a + y \). As the proof of Lemma 4, if \( j_x > j'_{x} \), then the above token can be consumed by the execution of \( \Pi_{jx} \). Otherwise a token must be added at the \( x \)-th step. The theorem is thus proved.

Thus, procedure token_Added() determines the least number of tokens to be inserted to avoid deadlocks and therefore represent a new DFC different than that by Eq. (1) and by [ZHO 91]. As an example, consider the case (Fig. 5) \( a = 2, b = 4, d_1 = d_2 = 1 \), and new paths are generated, using Rule TT.3, from \( \Pi_{g2} \) to \( \Pi_{j1} \) and \( \Pi_{j2} \). Thus we
have \( y = 2, j_1 = 1 \), and \( j_2 = 2 \). We start from \( j_1 (x = 1) \). Initially, \( \Pi_{g1} \) and \( \Pi_{j1} \) must both execute to avoid deadlock. \( \Pi_{g2} \), however, cannot execute because \( \Pi_{g2} \) has not executed to inject a token into the NP. Thus, we add a token to the NP and set the variable \( \text{token}_\text{Added} = 1 \). Next consider \( \Pi_{j2} (x = 2) \) and the token to be injected by executing \( \Pi_{g2} \) is to support the execution of \( \Pi_{j1} \), rather than \( \Pi_{j2} \) if \( \text{token}_\text{Added} = 0 \). Thus we compare \( j_2 = 2 \) to \((x - 1 - 1) \ast a + y = 2 \) rather than \((x - 1) \ast a + y) \); thus there is no need to add a token. No more \( j_2 \) needs to be considered; hence, the total number of tokens to be inserted into the NP is one.

When more than one PSP in \( X_g \) are needed, we can apply the above algorithm to each set of \{1 process in SME1, r processes in SME2\}.

Case 2: \( d > 1 \)

In this case, although the PSPs in SME1 and SME2 do not execute alternatively, Theorem 3 still holds. The argument is as follows. Let \( d \) be the synchronic distances among PSPs in SME1, then the range of execution times of \( \Pi_{gy} \) (or the case of \( \Pi_{gy} \)) is from \( \frac{\tau_j}{a} - d \) to \( \frac{\tau_j}{a} + d \).

When \( \tau_g \to \infty \), the range collapses to a single point of \( \frac{\tau_j}{a} \) since \( d \) is finite. Similar conclusion applies to the total number of firings of the r PSPs in SME2. Theorem 4, however, may not be applicable. Further research is needed to modify the procedure \( \text{Token}_\text{Added}() \).

C. TT-Generations Between an SME and an LEX

In this case, it is easy to see that any PSP in the SME must coordinate to all PSPs in the LEX whether the SME is full or partial. Hence Rule TT.3 must be applied without relaxation.

In the next section, we will apply the knitting technique to synthesize PME and SME.

VIII. Application of Extended Knitting Technique to Synthesize PME and SME

The net in Fig. 2(a) can be synthesized as follows. First we construct the basic process \( \Pi_1 = [p_6 t_1 p_2 t_2 p_6] \) with a token and the PP generation of \( \Pi_2 = [p_6 t_3 t_4 p_6] \), forming a PME. Hence these two PSPs are mutually exclusive to each other. \( p_6 \) is the home place. The temporal matrix is shown in Fig. 2(c). They can be sequentialized using the synchronization rule. It is done by generating a path, \([t_4 p_1 t_1] \), with three tokens in \( p_1 \) and a TT-path, \([t_2 t_3 t_3] \), forming an SME as shown in Fig. 2(d). \( p_1 \) is a C-place. The PN after the synchronization generation is bounded, live, and reversible. The execution order of \( \Pi_1 \) and \( \Pi_2 \) now alternates. Now generate a new TT path (a B-path) from \( t_3 \) to \( t_2 \) with tokens in \( p_5 \) which is a B-place; it is a backward TT generation since \( t_2 \) executes earlier than \( t_3 \). At this juncture, we recognize that this new TT path is a B-path since the two PSPs containing \( t_2 \) and \( t_3 \) are 'SX' to each other. Apply Eq. (1) with \( d_g = 3, \text{MAX}[M(p_D)] \).
= 1, we have \(n^* = 3\) tokens in \(p_3\). This net is identical to that in Fig. 2(a).

**More General DFC**

We now demonstrate the usefulness of the generalized DFC in Eq. (1). First consider Fig. 2(b). The deadlock can be removed by putting one more token in \(p_h = p_6\) to fire \(t_2\) and thus to remove the trapped token at \(p_2\). In general, the DFC for this particular net is \(M(p_h) = \text{MAX}(M(p_C) - M(p_B), 0) + 1\), where \(M(p_C) - M(p_B)\) is the number of tokens that could be trapped in a place. One more token is required to fire transitions so as to remove one of the trapped tokens and to make the net deadlock-free. As shown in Fig. 2(a), note that \(p_3\) here is a \(p_C\). \(M(p_C) = d_{S3} = d_{23} = M(p_1) = 3\). \(M(p_B) = M(p_3) = 3\). Hence \(M(p_6) = 1\).

In the DFC by Zhou et al [ZHO 91], each \(B\)-path and each \(C\)-path is a single PSP. The DFC by Eq. (1), however, holds for more complicated PNs; hence is more useful for analysis of arbitrary PNs. Fig. 5 shows a PN expanded by generating new paths \([t_6, p_4, t_1]\) and \([t_5, p_8, t_5]\) by applying Rule TT.2 twice. Note both \(p_h\) and \(p_4\) are home places and 'SE' to \(p_2\). Hence \(\text{MAX}(p_D) = \min(M(p_h), M(p_4)) = 1\). \(d_{63} = 2\) and \(n^* = 2\) at \(p_B\). Repeatedly applying PP rules and other TT rules, we can obtain even more complicated PNs meeting the DFC. In the structure by Zhou et al [ZHO 91], there are exactly three places marked with tokens, therefore it is not useful for analysis since an arbitrary PN may contain a similar structure with more than three places marked. An example is shown in Fig. 6 where the fourth place \(p_{10}\) is also marked. In this case, \(\text{MAX}(p_D) = 2\), \(d_{54} = 3\); therefore \(n^* = 2\). Note this PN cannot be synthesized using the knitting rules; the DFC by Eq. (1), however, still works. Fig. 7 (the two SMEs are sequential to each other) and 8 (concurrent to each other) show other two cases where Eq. (1) holds.

Consider another PN in Fig. 9 which can be neither synthesized nor checked for deadlocks using Eq. (1). \(d_{52} = 4\), \(\text{MAX}(p_D) = 1\), \(n^* = 4\) at \(p_{B1}\) by Eq. (1) larger than the necessary '2'. Further, Calculating \(d_{58}\) and \(\text{MAX}(M(p_D))\) is, in general, a reachability problem. We can, however, disable \(t_7\) and \(t_7\) and fire as far as it can. We now have \(M(p_2) = M(p_D) = 1\) and \(d_{58}^\text{eff} = M(p_{C2}) = 2\). Eq. (1) turns effective again. Similar arguments apply to \(n^* = 3\) at \(p_{B2}\) where \(d_{53}^\text{eff} = 3\).

In order to analyze an arbitrary PN, we need to develop the algorithm for verifying DFC. Such an algorithm can be found in [SUN 92]. However, verifying DFC of arbitrary PNs may be in general rather complex. It is better to reduce the PN to the same simple structure as in [ZHO 91] such that the DFC by Eq. (1) is readily applicable. Using the knitting rules, we can remove some paths while preserving system properties.

In general, to find the DFC, disable the two end transitions of each \(C\)-path so that they can never fire. We fire all enabled transitions until all transitions are disabled. Now define \(M_{\text{eff}}(p_D)\) to be the maximum number of tokens in \(M(p_D)\). Similar
KNITTING TECHNIQUE AND STRUCTURAL MATRIX FOR DEADLOCK ANALYSIS AND SYNTHESIS OF PETRI NETS WITH SEQUENTIAL EXCLUSION / Daniel Y. Chao

Definition applies to $M_{eff}(p_i)$ (or $n^r$) and $M_{eff}(p_i)$ (or $d_i$).

The DFC for the net in Fig. 2(e) still applies here using these $M_{eff}$'s. The DFC for more than two CESs can be derived in a similar way.

IX. CONCLUSION

This paper has 1) explained SME and PME in the context of the knitting technique/structural relationship to automate the detection of PME or SME generations; 2) illustrated the application of S-Matrix to detect SMEs and deadlocks in PMEs; 3) synthesized SMEs based on the knitting technique; 4) generalization of DFC to more complicated SMEs than the example in [ZHO 91]; and 5) discovered new DFCs based on the rules of TT-generations between SMEs.

We currently have a X window interactive system for drawing, analyzing, and synthesizing PNs. [CHA 94b]. Future work would be to implement the algorithms presented in this paper in the current X window PN graphics tool. Also we should strive toward discovering more new DFCs by removing the constraint of forbidden generations, such as the TP and PT generations which also involved shared resources and allowing the forbidden generations would produce new classes of nets. To maintain well-behaved generations, new generations must accompany the above forbidden generations. Further, we should automate the PN analysis via the application of the knitting technique and DFCs by detecting whether rules are violated. We have implemented in the X Window tool the algorithm for constructing the S-Matrix [CHA 94b]. An algorithm must be developed to check a PN against the knitting rules based on the S-Matrix. As new rules and DFCs are discovered, this synthesis-based analysis becomes more powerful as more design errors and their sources can be discovered. Any detected rule violation signals potential ill-designs. Similarly, the reverse process of reduction [CHA 92] becomes more powerful as more paths can be deleted. Any well-behaved net violating some rules prompts us to discover new rules. Also, we should extend the invariant-search technique in [CHA 93e] to the PNs with SMEs and investigate the relationship between the invariant and the DFC. We have performed research on analyzing iteration bounds of data flow graphs or marked graphs [CHA 93b,d,97] and would like to extend the knitting technique for performance synthesis with special emphasis on the effect of SMEs on performance.

REFERENCES


[CHA 93a] [CHA 93a] D. Y. Chao, M. C.


KNITTING TECHNIQUE AND STRUCTURAL MATRIX FOR DEADLOCK ANALYSIS AND SYNTHESIS OF PETRI NETS WITH SEQUENTIAL EXCLUSION

Daniel Y. Chao


Dr. D.Y. Chao (越五) received the Ph.D degree from in electrical engineering and computer science from the University of California, Berkeley in 1987.

From 1987-1988, he worked at Bell Laboratories. Since 1988, he joined the computer and information science department of New Jersey Institute Since 1994, he joined the MIS department of NCCU as an associate professor. Since February, 1997, he has been promoted to a full professor. His research interest was in the application of Petri nets to the design and synthesis of communication protocols and the CAD implementation of a multifunction Petri net graphic tool. He is now working on the exploration of property of a new class of Petri nets and implementation of several CAI tools based on Visual C++. He has published 72 (including 15 journal) papers in the area of communication protocols, Petri nets, DQDB, networks, FMS, data flow graphs and neural networks.
Fig. 1. An example of PME (after [ZHO 91 C])
Fig. 2. (a). An example of sequential mutual exclusion (after [ZHO 91C])
Fig. 2.(b). An example of a deadlock situation.

Fig. 2.(c). The S-Matrix for the two basic processes of the Petri net in Fig. 2(a).
Fig. 2.(d). An example of a Petri net synthesis of Fig. 2(a).

Fig. 2.(e). The new path [T3 P5 T2] is a TT-generation within the SME.
Fig. 3. (a). An example of a Petri net. Fig. 3. (b). An example of a Petri net w/o the generation of VP (PSP8-[P3T4] in Fig. 3(a))
Fig. 3.(c). An example of a Petri net w/o the elementary cycle and the path w/o tokens based on Fig. 3(a).

Fig. 3.(d). An example of no "exclusive TT interaction" rule.
Fig. 3.(e). An example of a deadlock situation in a Petri net with no tokens in P9 based on Fig. 3(a).

Fig. 3.(f). An example of the S-Matrix to record the relationship between PSPs of Fig. 3(a).
Fig. 4. Examples of the Use of Knitting Techniques.
Fig. 5. A more complicated PN meeting the DFC.

Fig. 6. An example where the DFC still holds for a PN that cannot be synthesized.
Fig. 8. An example of two SMEs concurrent to each other.
Fig. 9. A more complicated case for DFC with two C-paths and two B-paths.

Fig. 10. A Conceptual Petri Net in Paths.