

Petri Net Synthesis and Synchronization Using Knitting Technique

DANIEL Y. CHAO

*Department of Management and Information Science
National Chengchi University
Taipei, Taiwan 116, R.O.C.
E-mail: yaw@mis.nccu.edu.tw*

Petri net (PN) synthesis can avoid the state exploration problem, which is of exponential complexity, by guaranteeing the correctness in the Petri net while incrementally expanding the net. The conventional Petri net synthesis approaches, in general, suffer the drawback of only being able to synthesize a few classes of nets. However, the knitting technique, originally proposed by Chao [1-4], can synthesize Petri nets beyond asymmetric-choice nets. In addition, one major advantage of the knitting technique is that the resultant Petri net is guaranteed to be live, bounded, and reversible — the well-behaved properties. Therefore, the cumbersome reanalysis and modification procedures that are inevitable for the conventional Petri net synthesis methods can be avoided. Most current synthesis techniques cannot handle systems with shared resources. Zhou et al. [5] presented the conditions for a PN containing Sequential Mutual Exclusion (SME) to be live, bounded, and deadlock-free. The major motivation of this work is to generalize Zhou et al.'s pioneering work [5] and to extend the knitting technique to construction of classes of PNs that involve synchronization and shared resources according to the synthesis rules. In addition, the knitting technique developed prior to this work concentrated on the structural relationships among the pseudo-processes only and was not related to the marking. This paper is the first work to consider marking in the PN synthesis with the knitting technique.

Keywords: Petri net, knitting technique, synchronization, deadlock, SME, liveness, boundedness, shared resource

1. INTRODUCTION

The Petri net (PN) behavior depends not only on the graphical structure, but also on the initial marking of the PN. Therefore, it cannot be determined by means of static analysis, such as dependency analysis, only; rather, it can be obtained through reachability analysis. The size of a reachability graph is determined not only by the structure of the net, but also by the initial marking. In general, the larger the initial marking (i.e., the more tokens are involved), the larger the reachability graph. It has been shown that the complexity of the reachability analysis of PNs is exponential [6].

Petri net synthesis can construct large nets without the requirement of reachability analysis. Any synthesis technique should obey the following three principles:

Received December 2, 1996; accepted September 26, 1997.
Communicated by Shing-Tsaan Huang.

- (1) Preservation: The synthesized net should preserve all well-behaved properties.
- (2) Simplicity: To reduce the complexity of synthesis, the rules must be as simple as possible.
- (3) Generality: The rules should be as powerful as possible to generate as many classes of nets as possible.

The synthesis rules developed in the knitting technique are based on the above principles. New paths are added according to two guidelines, which do not disturb the reachable markings of the old net, and which ensure that the new path is live, bounded, and reversible. A temporal matrix (T-Matrix) is used in the knitting technique to record the structural relationships (concurrent, exclusive, sequential, cyclic etc.) among processes and to detect rule violations when a new generation is created from one node to another. There are four types of path generations; namely TT, PP, TP, and PT; depending on whether the generation is from a transition or a place to a transition or a place. One of the following three actions is taken, depending on the structural relationship between the two nodes and the type of generation: (1) forbidden, (2) permitted, or (3) permitted; however, more new generations are needed. In the previous version of the knitting technique, TT (PP)-generations were permitted between two sequential or concurrent (exclusive) processes while TP (PT) generations were forbidden since they could create unbounded (nonlive) PNs.

An algorithm was developed [2, 7] accordingly 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 [1] among multiple parties and has been applied to synthesis of communication protocols and automated manufacturing systems [7]. In [8], we extended these rules to synthesize General Petri Nets (GPNs) whose arcs have multiple weights. This technique can synthesize nets more generally than can *free-choice* (FC)^{*1}, *extended free-choice* (EFC)^{*2} or *asymmetric-choice nets* (AC)^{*3} [MUR 89a]. FC is a subset of EFC, which, in turn, is a subset of AC.

Other contributions in this direction have been as follows. Esparza & Silva [9] proposed two rules to synthesize live and bounded free-choice PNs. Their TT- and PP-handles [10] are similar to the TT- and PP-paths in [1]. Similar to [1] but rephrased in another way, they also showed that circuits without TP- and PT-handles have good structural properties. However, they did not explicitly apply them to synthesis using the notion of a *structural relationship*. In [9], Rule RF1 refined a macroplace by means of a state machine, and RF2 added a marking structurally implicit place (MSIP) to an FC net. RF1 in [9] corresponds to the PP rule in [1] and add paths between places. RF2 in [9] corresponds to the TT rule in [1], adds paths between transitions and increases 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 [1], however, requires no checking of the generated paths; rather, it uses the T-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 [9]. In addition, the T-Matrix can record self-loops and find the maximum concurrency with linear time complexity. Although [10] added the RF3 rule to synthesize EFC nets, [10] was unable to synthesize ACs which can be synthesized easily using the knitting technique.

*1. $\forall p \in P, |\bullet p| \leq 1$ or $\bullet(\bullet p) = \{p\}$.

*2. $\forall p_1, p_2 \in P, \bullet p_1 \cap \bullet p_2 \neq \emptyset \Rightarrow \bullet p_1 = \bullet p_2$.

*3. $\forall p_1, p_2 \in P, \bullet p_1 \cap \bullet p_2 \neq \emptyset \Rightarrow \bullet p_1 \subseteq \bullet p_2$ or $\bullet p_1 \supseteq \bullet p_2$.

Datta & Ghosh [11-12] also proposed two rules that are similar to the TT and PP rules [1]. Instead of using the T-Matrix, they used 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 [13] proposed a synthesis procedure to compose two modules with two TT paths. An algorithm to find P-invariants 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 given in section 3 of this work.

The knitting technique is a rule-based interactive approach. It expands the 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) provision of knowledge of which construction is building which class of nets. It, therefore, opens up a novel avenue to PN analysis.

Rather than refining transitions, this technique generates new paths to a PN, N_1 , by producing a larger PN, N_2 based on **two guidelines**. (1) The new generations are obtained in such a fashion that all the reachable markings in N_1 remain unaffected in N_2 ; hence, all the transitions and places in N_1 stay live and bounded, respectively. (2) N_2 is kept live and bounded by making the new paths live and bounded. This notion is novel compared with other approaches and can synthesize more general PNs than others can.

Using the knitting technique, designers start with a basic process, which is modeled by a set of closed-circuits including sequentially-connected places and transitions with a home-place (defined in section 2) 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 specifications. 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 a final well-behaved PN. Another advantage of this knitting technique is that it is easily adapted to computer implementation to make the synthesis of PNs a user-friendly process [2, 7].

The knitting technique, however, cannot synthesize certain classes of nets. For instance, there is no order of firing among the members of a set of transitions that are exclusive to each other. Sometimes, these transitions must execute one by one. In addition, if the synthesized net is initially safe, it stays safe for any reachable marking. It is marking monotonic; that is, it will not evolve into a deadlock due to addition of tokens. Certain classes of nets will evolve in such a way as to become unsafe and marking nonmonotonic, and will allow any positive synchronic distance.

The above limitation springs from the fact that some generations are forbidden. Allowing forbidden generations will produce new classes of nets. To maintain well-behavedness, new generations must accompany the above forbidden generations. Based on this fact, this paper develops a set of new rules to generate new classes of nets. First, we remove the restriction of forbidding the generation of a new TT-path between two exclusive transitions. We add another new TT-path such that the above two exclusive transitions are sequentialized with a synchronic distance of one. These two transitions are both exclusive and sequential to each other – a new temporal relationship, sequentially exclusive,

denoted as “SX” or $\hat{\uparrow}$. Second, because of this relationship, in order to maintain completeness, new rules must be developed to deal with generations between transitions that are “SX” to each other or between one that is “SX” to some transitions and another that is not. Section 4 is devoted to this new rule, which produces new classes of nets with the property of marking nonmonotonicity. Application of this new rule to synthesis of nets with resource sharing will be illustrated.

One important difference between the new rule and the previous ones is that it considers marking rather than the structural relationship. Because of the reachability problem, the previous knitting technique dealt with structures by disregarding marking to avoid exponential complexity of synthesis. As a result, the nets synthesized was limited. By considering localized markings which are structurally constrained, the reachability problem can be avoided. The new rule is constructed accordingly.

Automated manufacturing systems [5] often share resources, such as machines, robots, and material transporters. Because of the nature of resource sharing, processes in a system may enter a deadlock state while waiting for the completion and, hence, the release of the resource, of another process. There are two forms of resource sharing: Parallel mutual exclusion (PME) and sequential mutual exclusion (SME) as proposed by Zhou et al. [5]. A PME, as shown in Fig. 1, models a resource shared by independent processes, and an SME, as shown in Fig. 2, models sequentialization (synchronization) of PMEs. In a PME, any independent process may monopolize the use of a resource, thus creating an unfair situation. The SME eliminates this unfairness problem by serializing or synchronizing the independent processes.

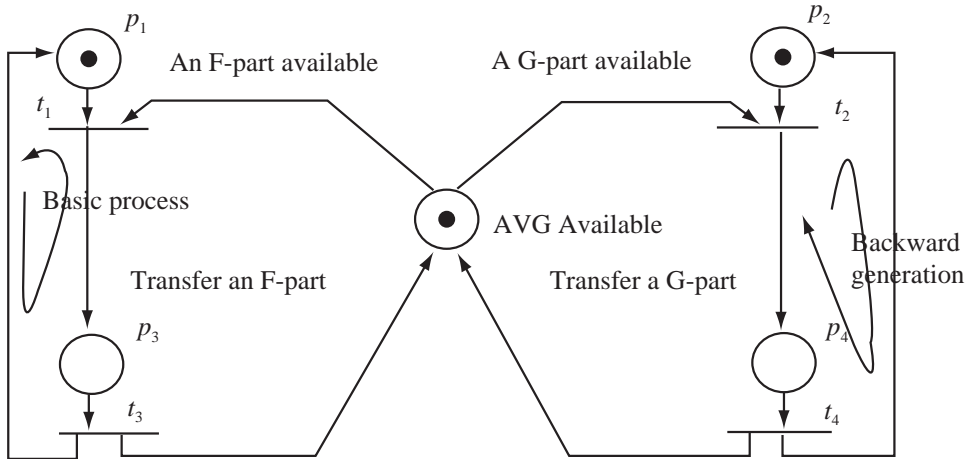


Fig. 1. An example of PME (after [5]).

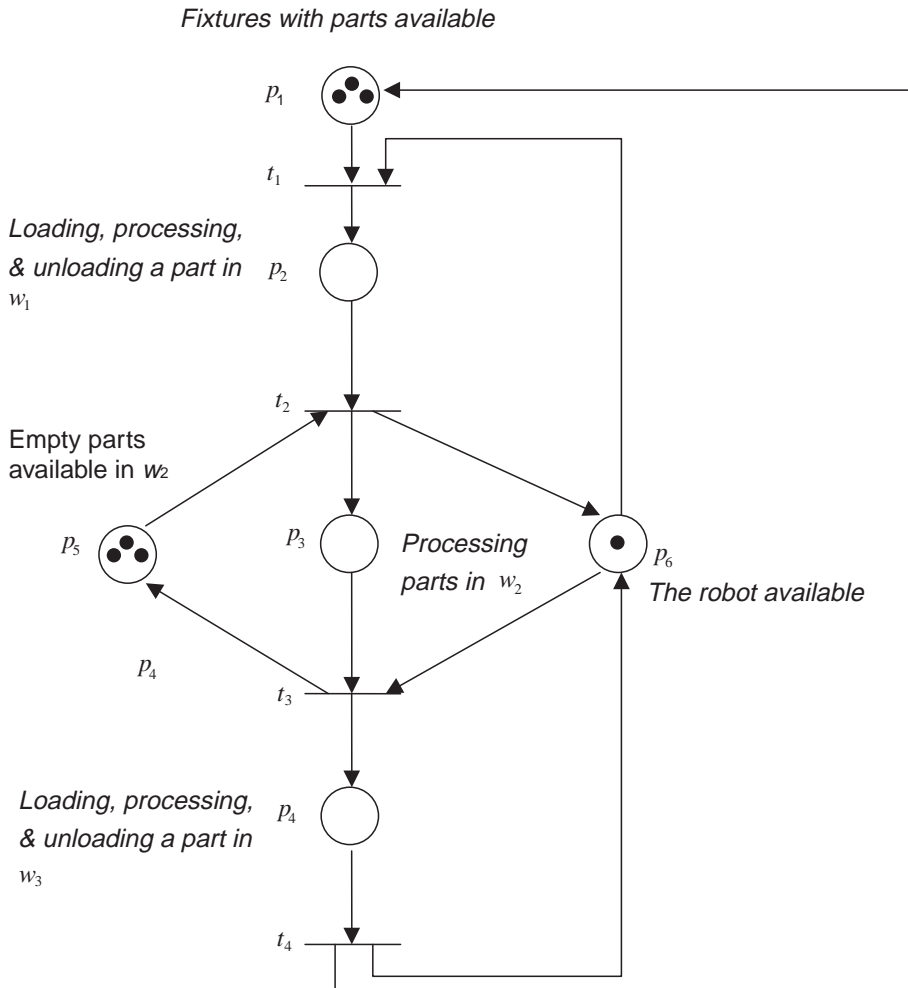


Fig. 2. An example of sequential mutual exclusion (after [5]).

The PN models of SMEs have deadlocks due to either an inappropriate structure of the net or incorrect allocation of shared resources. Zhou et al. [5] discovered that a Petri net with SME could become nonlive and nonreversible when inappropriate tokens were distributed in the net. In their pioneering work, they derived the sufficient conditions under which a net containing SMEs was live, bounded, and reversible. They also pointed out that an algorithm is needed to detect sequential and mutual exclusions automatically and to verify the conditions for boundedness, liveness and reversibility. In this work, we present an alternative approach to study SME and PME. Further, we generate the sufficient and necessary deadlock-free Conditions (DFCs) for SMEs [5]. Most synthesis approaches, including top-down/bottom-up [10, 14-15], peer entity generation [16], and the knitting technique [1-4], do not deal with processes sharing resources. We were thus, motivated, in this work, to extend the knitting technique so that it can also handle Petri net with shared resources.

This paper is organized as follows. The preliminaries, which includes the terminology, the synthesis rules for pure and interactive generation, and some examples of how to apply these rules, are presented in section 2. The synchronization rule is discussed in section 3. Section 4 presents the synthesis-synchronization rules between an SME and another SME or LEX. The use of these rules is illustrated in section 5. Section 6 concludes this paper.

2. PRELIMINARIES

We refer the reader to [17] for various PNs terminology. A brief review is presented below for better understanding of this paper. A marked PN is *reversible* if and only if $M_0 \in R(N, M) \forall M \in R(N, M_0)$. Most concurrent systems are designed to achieve cyclic behavior, so their Petri net models should be reversible, i.e., their initial marking, which stands for the resources available at the beginning of every iteration, should be reachable from any other marking. We call an *iteration* the period from the initial marking to other markings and back to the initial marking for the first time. Note that the corresponding firing sequence may not include all the transitions in PN. This section consists of three parts, namely, terminology, synthesis rules, and examples of Petri net synthesis using the knitting technique.

2.1 Terminology

We let N denote a Petri net, with marking M and initial marking M_0 . The marking at place p is denoted as $m(p)$. We also let $R(N, M)$ denote the reachability set of a Petri net N with a current marking M .

Definition: A marked Petri net N is B -bounded if and only if $m(p) \leq B, \forall p \in P$ (the set of all places) and $M \in R(N, M_0)$, where B is a positive integer. If $B = 1$, N is safe. N is live if and only if $\forall t \in T$ (the set of all transitions), and $\forall M \in R(N, M_0), \exists$ a firing sequence σ of transitions leading to a marking which enables t . N is reversible if and only if $M_0 \in R(N, M), \forall M \in R(N, M_0)$. N is **well-behaved** if N is live, bounded, and reversible. $M_s = M_c(A) - M_d(B)$ is defined as: $\forall p \in A$, if $\neg(p \in B)$, then $m_s(p) = m_c(p)$, else $m_s(p) = m_c(p) - m_d(p)$. The operation ‘+’ is defined similarly.

We will now define *home place*, *pseudo process* and related terms in the following.

Definition (Home Place): When \forall transitions t_α are enabled in the initial marking of a PN, each place $p_h \in \bullet t_\alpha$ is defined as a *home place*.

Definition (Basic Process): A basic process is a cycle in which (1) $\forall n_i, |\bullet n_i| = |n_i \bullet| = 1$, where $i \in [1, z]$ (where $z > 1$ is an integer), and (2) a place p_h is marked with a token.

In other words, in a basic process, $\forall t \in T, |\bullet t| = |t \bullet| = 1$, and $\forall p \in P, |\bullet p| = |p \bullet| = 1$. [$p_1 t_1 p_3 t_3 p_1$] in Fig. 1 is a basic process, and p_1 is a home place. Note that the basic process is the first step in Petri net synthesis using the knitting technique since it is live, bounded, and reversible. Every other synthesis step is an extension of the basic process. We begin the synthesis with a well-behaved Petri net (basic process) and guarantee the preservation of well-behavedness after each synthetic operation without any analysis; hence, the knitting

technique can be used to synthesize large and complex Petri nets. In the knitting technique, the basic structure of the Petri nets is the *pseudo process (PSP)*, which is defined in the following.

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

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. Examples of PSP and VP will be presented after we define the structural relationship between two PSPs. Prior to that, we have the following definitions:

Definition:

If Π_1 and Π_2 in a PN are in the same elementary cycle, then

- $\Pi_1 \leftrightarrow \Pi_2$, i.e., they are sequential (SQ) to each other,
 - if the cycle has tokens,
 - if the DEP from Π_1 to Π_2 does not contain any tokens,
 - then $\Pi_1 \rightarrow \Pi_2$; i.e., Π_1 is sequential earlier (SE) than Π_2 ;
 - else $\Pi_1 \leftarrow \Pi_2$; i.e., Π_1 is sequential later (SL) than Π_2 .
 - A special case of “SE” (“SL”) is “SP” (“SN”) if the $n_j(n_g)$ of Π_1 equals $n_g(n_j)$ of Π_2 .
 - else if the cycle contains no tokens, then Π_1 or Π_2 ; i.e., they are cyclic (CL) to each other.

Definition: Let $DEP_1 = [n_1 n_2 \dots n_k \Pi_1]$ and $DEP_2 = [n_1 n'_2 \dots n'_k \Pi_2]$, where $DEP_1 \cap DEP_2 = \{n_1\}$, Π_1 and Π_2 are two PSPs and there are no DEPs (called bridges) from a node ($\neq n_1$) of DEP_1 to a node ($\neq n_1$) of DEP_2 and vice versa. n_1 is called a **prime start node**. If $n_1 \in P$, then $\Pi_1 \dagger \Pi_2$. $\Pi_1 \parallel \Pi_2$ if none of $\Pi_1 \mid \Pi_2$, $\Pi_1 \leftrightarrow \Pi_2$, and $\Pi_1 \circ \Pi_2$ hold.

Note that $\Pi_1 \parallel \Pi_2$ implies that $n_1 \in T$ or Π_1 and Π_2 are in two separate components of a non-strongly-connected PN. Also, in any synthesized PN using the TT and PP rules, Π_1 cannot be both concurrent and exclusive to Π_2 using the above definition. This is, however, not true for PNs which cannot be synthesized using the TT and PP rules.

In Fig. 1, $[t_1 p_3 t_3]$ is a PSP whose generation point and joint are t_1 and t_3 , respectively. $[p_3 t_3 p_6 t_5 p_8]$ and $[p_3 t_4 p_7 t_6 p_8]$ in Fig. 3 share the same prime $p_s = p_3$ and, hence, are exclusive to each other. $[t_4 p_9 t_5]$ and $[t_4 p_7 t_6 p_8]$ in Fig. 4 share the same $t_s = t_4$ and, hence, are concurrent to each other. In Fig. 4, $[p_3 t_4] \rightarrow [t_3 p_6 t_5]$. PSP $[p_3 t_4]$ in Fig. 4 is a VP.

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 Π_1 , then $\Pi_1 \rightarrow \Pi_2$ implies that Π_1 is executed earlier than Π_2 , $\Pi_1 \dagger \Pi_2$ implies that there is no need to execute Π_2 to complete the iteration, and $\Pi_1 \parallel \Pi_2$ implies that both Π_1 and Π_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* with 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 Petri net 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 Petri net.

A new PSP (Π^w) is generated from a node in Π_g to another node in Π_j . If $\Pi_g = \Pi_j$ (e.g., $[p_3 t_4 p_7 t_6 p_8]$ in Fig. 3), it is a *pure generation (PG)*; otherwise (e.g., $[t_3 p_5 t_4]$ in Fig. 4), it is an **interactive generation (IG)**. Pure-generation processes do not involve interactions; they are pure growth processes, i.e., monogenesis. If both the *generation point* and *joint* of Π^w are places (transitions), then it is a *PP(TT) generation*; the corresponding path is called

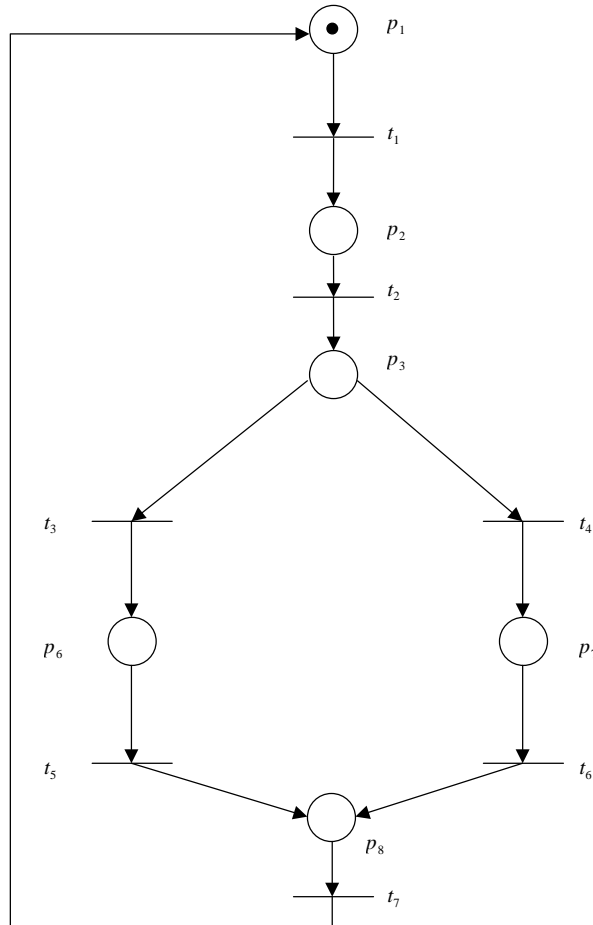


Fig. 3. An example of a Petri net with or without an elementary circle and a path with tokens.

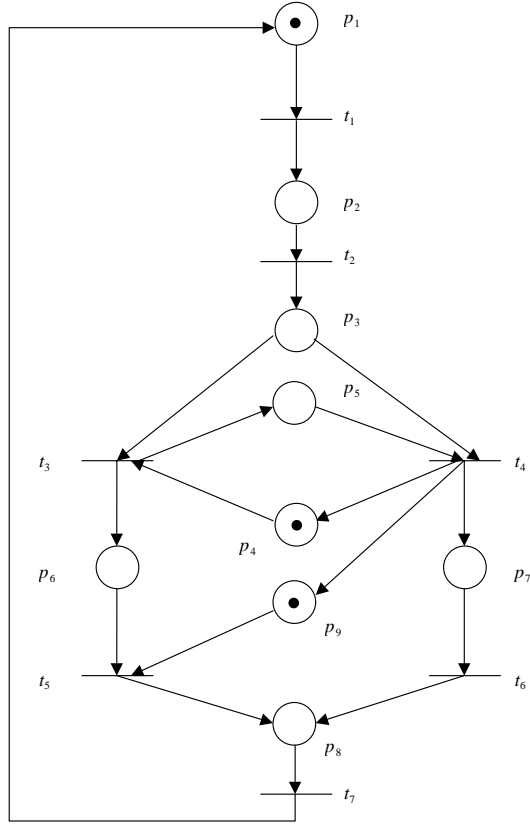


Fig. 4. An example of a Petri net.

a *PP – path* (*TT – path*). The t_g (t_j) of a PP-path is the output (input) transition of its *generation point* (*joint*). See Fig. 3 for $t_g = t_4$ and $t_j = t_6$ of the PP-path $[p_3 t_4 p_7 t_6 p_8]$. The p_g (p_j) of a *TT – path* is the output (input) place of its *generation point* (*joint*). In Fig. 4, p_5 is both the p_g and p_j of TT-path $[t_3 p_5 t_4]$.

We define the local exclusive set and local concurrent set as follows.

Definition (Local Exclusive Set): A local exclusive set (LEX) of Π_i with respect to Π_k, X_{ik} , is the maximal set of all PSPs which are exclusive to each other and are equal to or exclusive to Π_i , but not to Π_k . That is, $X_{ik} = \text{LEX}(\Pi_i, \Pi_k) = \{\Pi_z \mid \Pi_z = \Pi_i \text{ or } \Pi_z \dagger \Pi_i, \neg(\Pi_z / \Pi_k), \forall \Pi_{z1}, \Pi_{z2} \in X_{ik}, \Pi_{z1} \dagger \Pi_{z2}\}$.

Definition (Local Concurrent Set): A local concurrent set (LCN) of Π_i with respect to $\Pi_k, C_{ik} = \text{LCN}(\Pi_i, \Pi_k)$, is the maximal set of all PSPs which are concurrent with each other and are equal to or concurrent with Π_i , but not with Π_k , i.e., $C_{ik} = C(\Pi_i, \Pi_k) = \{\Pi_z \mid \Pi_z = \Pi_i \text{ or } \Pi_z \parallel \Pi_i, \neg(\Pi_z \parallel \Pi_k), \forall \Pi_{z1}, \Pi_{z2} \in C_{ik}, \Pi_{z1} \parallel \Pi_{z2}\}$.

In Fig. 5, $X_{14} = \{\Pi_1, \Pi_3, \Pi_5\}$ and $X_{41} = \{\Pi_2, \Pi_4\}$. In Fig. 6, $C_{14} = \{\Pi_1, \Pi_3\}$ and $C_{41} = \{\Pi_2, \Pi_4\}$. Note that C_{ik} and X_{ik} may not be unique. All the PSPs of $C_{ik}(X_{ik})$ must be involved when we generate a PP(TT)-path between mutually exclusive (concurrent) PSPs; otherwise, deadlock or unboundedness may occur. As will become clear later; the following two tuples are dual to each other: (LEX, TT, ||) and (LCN, PP, †).

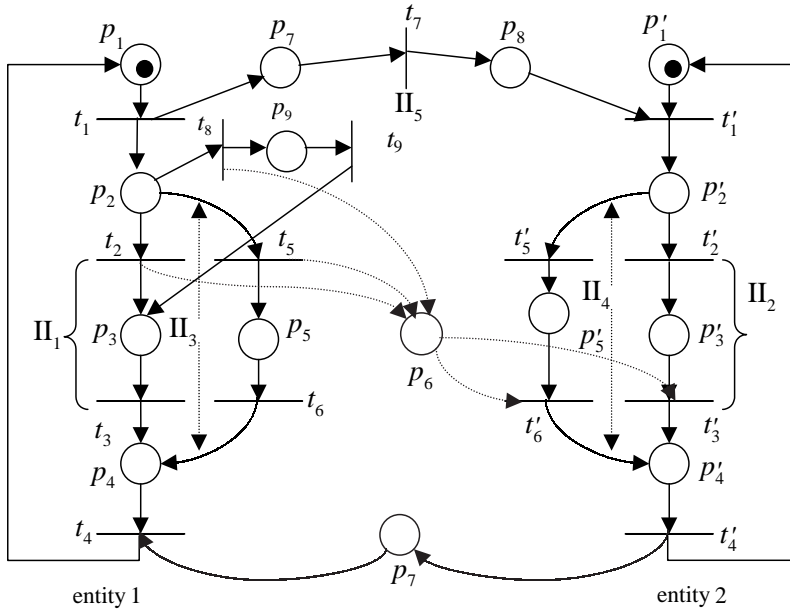


Fig. 5. An example of Rules TT.3 and TT.4.

The definition of the LEX (LCN) between two PSPs can also be applied to the LEX (LCN) between two transitions (places). Thus, $LEX(t_a, t_b)$ [$LCN(p_a, p_b)$] is a set of transitions or places, instead of PSPs. Let $G(J)$ denote the set of all $\Pi_g(\Pi_j)$ involved in a single application of the TT or PP rule. To avoid the unboundedness problem, it is necessary to have a new directed path from each PSP in X_{g_i} to each PSP in X_{j_g} .

Suppose that a Petri net models a set of processes or entities interacting with one another by sending and receiving messages. An entity is an active element (with a token in its home place) such that it can execute some tasks independent of other entities. Initially, each entity has a **home place** (or places) with a *token**4 indicating that it is ready to start. Each entity conducts a set of tasks in serial (which can be considered as transitions), and then returns to the original ready state. Each state can be considered as a place. Executing

*4. The synthesis rules still hold if a home place has multiple tokens. We adopt a token here to simplify the discussion.

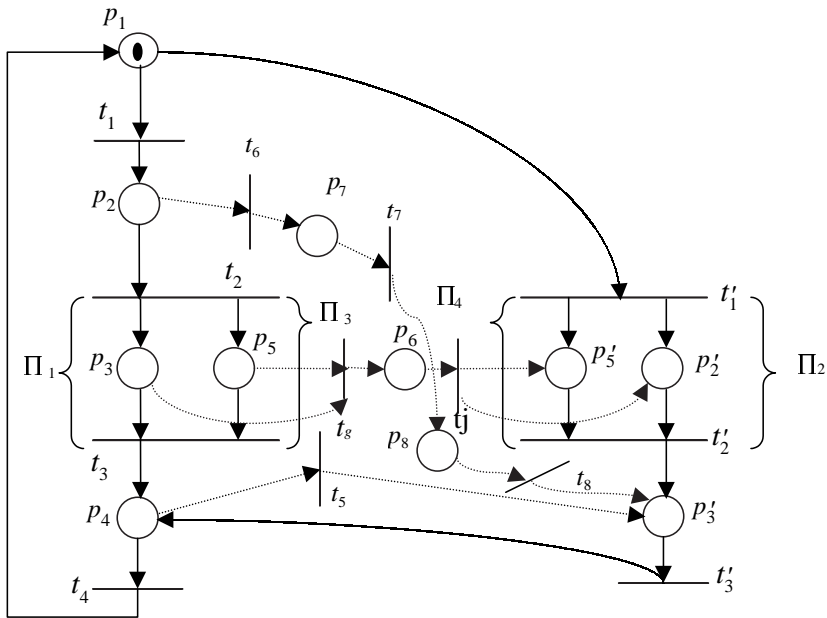


Fig. 6. Interactive PP generations (Rules PP.1 and PP.2).

a task is like firing a transition. At the completion of each task, the token leaves the original state(s) and enters the next state(s) by firing transitions. Thus, the token moves from the *home place* through consecutive places by firing a sequence of transitions. Eventually, the token will return to the *home place*, forming a cycle as shown in Fig. 1. As stated earlier, the synthesis of a PN begins with a basic process. This *basic process* can be expanded through a series of synthetic operations, i.e., path generations.

Definition (Generating PSP and Joint PSP): For a PSP, Π , the PSP containing its $g_{\Pi}(j_{\Pi})$ is called the generating (joint) PSP of Π , denoted as $\Pi_g(\Pi_j)$.

2.2 Rules

The synthesis rules developed for the knitting technique are based on the above principles. The synthesis rules presented below are complete in the sense that depending on the structural relationship between $t_g(p_g)$ and $t_j(p_j)$, different rules are proposed. Some are forbidden (e.g., when $t_g \mid t_j$), and some require the generation of additional PSPs. All t_g , t_j , p_g and p_j in the rules refer to Π^w s, rather than those of Π_g and Π_j .

We present the synthesis rules below with examples given in subsection 2.3. To make understanding the rules easier, the reader may choose to view the examples first.

The *Synthetic TT* and *Synthetic PP* rules are formalized as follows.

TT Rules: For a Π^w from $t_g \in \Pi_g$ to $t_j \in \Pi_j$ generated by the designer,

- (1) TT.1 **If** $t_g \dagger t_j$ and $\bullet t_g \cap \bullet t_j = \phi$ 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) TT.2 **If** $t_g \leftarrow t_j$ or $t_g = t_j$ and without firing t_j , there does not exist a firing sequence σ to fire t_g , **then** insert a token in a place of Π^w ; this place becomes a new home place.
If $\Pi_g = \Pi_j$, **then** display “You may generate another Π ” and return.
- (3) TT.3 (a) TT.3.1 Generate a *TP-path* from a transition t_g of each Π_g in X_{gj} to a place p_k in the Π^w .
(b) TT.3.2 Generate a virtual *PT-path* from a place p_j to a transition t_j of each Π_j in X_{jg} .
- (4) TT.4 **If** there was no path from t_g to t_j prior to this generation or if both are in a circle which was generated using the PP rule, **then** (1) generate a new *TT-path* from t_g' to t_j' such that t_g, t_j, t_g' and t_j' are in a circle, and (2) Go to step 2.

PP Rules: For an Π^w from $p_g \in \Pi_g$ to $p_j \in \Pi_j$ generated by the designer,

- (1) PP.1 **If** $p_g \parallel p_j$, **then** display “Disallowed, generate another Π ” and return.
- (2) PP.2 (a) PP.2.1 Generate a *TP-path* from a transition t_k of Π^w to a place p_j of each Π_j in C_{jg} .
(b) PP.2.2 Generate a virtual *PT-path* from a place p_g of each Π_g in C_{gj} to the transition t_g of Π^w .

Note also that some generations may require the application of more than one rule. For instance, a backward (i.e., $t_g \leftarrow t_j$) IG may require that both Rules TT.2 and TT.3 be applied. It is rather awkward to explicitly express and apply both rules on all such occasions in the above TT rules, which will not be simple if all such rule combinations have to be considered. The old rules TT.3 and TT.4 have been interchanged in the above TT rules for this purpose to avoid explicit expressions of rule combinations. Note also the partial dual relationship between the *TT* and *PP* rules. Replacing transitions with places and vice versa, and reversing each arc in Rule TT.3, we obtain Rule PP.2. In [9], the corresponding net obtained is called its reverse-dual, N_{rd} . These rules have been proved in [8] to satisfy the two guidelines. To save space, we will not duplicate it here. Rather, we will present some examples.

Note that Fig. 6 is not an AC; hence, this approach can synthesize PNs more generally than can the *FC* proposed by Esparza & Silva [9-10] and Datta & Ghosh [11-12].

2.3 Examples of Petri Net Synthesis

This section provides examples of synthesized PNs. To save space, we will present Rules TT.2, TT.3, TT.4, and PP.2. For other rules, the reader is can refer to [8].

Backward Generation (TT.2): Fig. 1 shows a backward generation, $[t_4 p_2 t_2]$. A token is added to p_2 by Rule TT.2.

Interactive TT Generation (TT.3 and TT.4): Fig. 5 shows an interactive *TT-generation* from t_2 of entity 1 to t_6 of entity 2. $X_{gj} = \{\Pi_1, \Pi_3, \Pi_5\}$ and $X_{jg} = \{\Pi_2, \Pi_4\}$. We apply TT.3.1 to generate $[t_5 p_6]$ and $[t_8 p_6]$. As will become clear later, this is done to prevent deadlock.

The Π^w , $[t_2 p_6 t'_6]$, is *unbounded* because if Π_1 fires infinite times to deposit infinite tokens in place p_6 which can never be consumed, then p_6 is thus *unbounded* if we always fire transitions in Π_2 instead of those in Π_4 . The VP from p_6 to t'_3 using *TT.3.2* (Fig. 5) will consume the tokens in p_6 . We will now apply Rule *TT.4* to generate $[t'_4 p_7 t_4]$. Otherwise, p_6 can become unbounded by firing transitions in entity 1 infinite times but not firing any transition in entity 2. Next, we will apply Rule *TT.3* again. Now, transitions in entity 1 cannot fire infinite times without firing any transition in entity 2; thus, they are synchronized and the unboundedness problem disappears. Note that without the TP-paths obtained using Rule *TT.3.1*, the net becomes deadlocked. Another TT path, $[t_1 p_{10} t_7 p_8 t'_1]$, is generated by Rule *TT.3* with $X_{gj} = \Pi_g$ and $X_{jg} = \Pi_j$.

Interactive PP Generation (PP.2): Fig. 6 shows an interactive *PP generation* from p_5 to p'_5 . $C_{gj} = \{\Pi_1, \Pi_3\}$ and $C_{jg} = \{\Pi_2, \Pi_4\}$. Note that if a token in p_5 is diverted to p_6 and, hence, to p'_5 , none of the transitions in PN will be enabled, i.e., the PN will not be *live*. Hence, by Rules *PP.2.1* and *PP.2.2*, respectively, two VPs, $[p_3 t_g]$ and $[t_j p'_2]$, are generated. Another two paths, $[p_2 t_6 p_7 t_7 p_8 t_8 p'_3]$ and $[p_4 t_5 p'_3]$, are generated. Since for these two generations, $C_{gj} = \Pi_g$ and $C_{jg} = \Pi_j$, no other generations are needed, by Rules *PP.2.1* and *PP.2.2*.

3. THE SYNCHRONIZATION RULES

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

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

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

For example, p_3 in Fig. 3 is a decision place. t_3 and t_4 of Fig. 3 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 Petri net (Z, M_0) is defined by $d_{12} = \text{Max}\{\sigma(t_1) - \sigma(t_2), \sigma_i \in L(M), i = 1, 2\}$, where $\sigma(t_i)$, $i = 1, 2$, is the number of times transition t_i appears in σ . $L(M)$ is the set of all firing sequences for the net with the marking. The synchronic distance between Π_1 and Π_2 follows that between t_1 in Π_1 and t_2 in Π_2 .

For example, in the net shown in Fig. 3, $d_{34} = \infty$ because if we have a token in p_1 , there exists a firing sequence $\sigma = t_1 t_2 t_3 t_5 t_7 t_1 t_2 t_3 \dots$ in which $\sigma(t_3) = \infty$ and $\sigma(t_4) = 0$.

Now, a TT-path, $[t_4 p_9 t_5]$, is generated from t_4 in $\Pi_2 = [p_3 t_4 p_7 t_6 p_8]$ to t_5 in $\Pi_1 = [p_3 t_3 p_6 t_5 p_8]$, $\Pi_1 \dagger \Pi_2$, as shown in Fig. 3. This was referred to as “exclusive TT interaction” in [1], which forbids such a generation for the following reason. Each time t_4 fires, it injects a token into p_9 . After an infinite number of such firings, p_9 will become 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. This is done by generating a path with a token, $[t_4 p_4 t_3]$, from t_4 to t_3 , and a TT-path, $[t_3 p_5 t_4]$, from t_3 to t_4 , as shown in Fig. 4. The Petri net after the synchronization generation is bounded, live and reversible. Note that if there is no token in p_9 , the net becomes dead since PSP $[t_3 p_6 t_5]$ executes earlier than PSP $[t_4 p_7 t_6 p_8]$, and the generation from t_4 to t_5 becomes a backward generation. In order to remove this deadlock, a token must be inserted into p_9 according to Rule TT.2. p_9 is a new home place.

Note that t_1 and t_5 fire twice and once, respectively, within one iteration to restore the initial marking. Thus, $\sigma_{51} = 2$. An arbitrary positive synchronic distance of Y can be created by sequentializing Y exclusive transitions. In general, we should avoid a TT-generation between two transitions with synchronic distance greater than 1 in an iteration, which may cause unboundedness. For instance, assume that we generate a new TT-path $[t_1 p_{10} t_5]$, and that p_{10} becomes unbounded after infinite number of iterations. One can avoid this unboundedness by applying Rule TT.3 assuming that t_5 is still exclusive to t_6 . That is, we also generate a VP from p_{10} to t_6 . As a result, Rule TT.3 still applies. Note that we may put more than one token into Π^w without destroying the well-behavedness. In this case, the SX structure relationship does not capture the true behavior among the transitions that are SX to each other. For this reason, we retain the EX for them but enhance it with a synchronic distance identical to the total number of tokens added to Π^w . It takes linear time complexity to update d_{ij} against the exponential one required for analysis of the net.

We will now give another example in which two asynchronous entities can be synchronized. There were two entities in the Petri net shown in Fig. 5 prior to the synthesis. The synchronic distance between any two transitions of these two independent entities, respectively, is infinity. A single TT-path generated between them is insufficient. The two entities, however, can be synchronized. This is illustrated in Fig. 5.

The above examples have some properties in common: (1) a single TT-path generation will cause unboundedness, and (2) both exclusive-and independent-processes can be synchronized.

We will now develop the synchronization rule for exclusive- and independent-processes. For exclusive processes, synchronization paths must be generated between the control transitions of the exclusive PSPs. For independent processes, however, synchronization paths can be generated between any two transitions of the independent processes, respectively.

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 , and there was no path from t_g to t_j prior to the generation or $(t_g \dagger t_j$ and t_j is a control transition), then generate another TT-path from t'_g to t'_j such that these two TT-paths are in a cycle, and if $t_g \dagger t_j$, then both t_j and t'_j must be control transition with the same decision place.

This rule must be checked concurrently along with other TT rules. For instance, after the above cycle (with a token) generation, the following TT-path generation may become a backward generation; then, Rule TT.2 must be applied to insert a token in this TT-path.

4. RULES FOR TT GENERATIONS INVOLVING SMES AND/OR EXS

This section will discuss 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 a LEX. In addition to “CN”, “EX” and “SQ”, t_g may be “SX” to t_j , which corresponds to the generation within an SME. For “SX”, we can apply Rule TT.3 between LEX1 and LEX2 without affecting the well-behavedness. The mutual synchronic distance (MSD) d^m among PSPs in LEX1 or LEX2 is, however, no longer infinite. We denote 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 MSD. In other words, unlike PSPs in a LEX, no process in SME can fire infinitely without firing any other process in the SME. Rule TT.3 cannot be relaxed unconditionally. In other words, not all PSPs in SME1 or SME2 may participate in the generation. With $d^m = \infty$, this may create deadlocks or unbounded places since t_g and t_j may never execute, which is not true for d^m of finite values.

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 will uniformly say that TT-generations are applied from X_{g_j} to X_{j_g} . Therefore, when Rule TT.3 is relaxed, $X_{g_j}(X_{j_g})$, changing its original meaning somewhat, may be SME1 (SME2).

PSPs in both SME1 and SME2 are executed sequentially, and some TT-paths within the generation may be backward. We need to determine whether or not and how many tokens need to be added. These issues will be studied first, followed by presentation of the newly expanded knitting technique: the synthetic-synchronization rules and the proof of the preservation of the well-behavedness of these new rules.

TT-Generations within An SME

In this subsection, we will study the first case, i.e., TT-generations within an SME and application of this case to resource sharing in an FMS. 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 injected into Π^w . The difference is that insertion of a single token may not avoid the deadlock. An example is shown in Fig. 2, where a TT-path is generated backward from t_3 to t_2 . Tokens must be inserted into p_5 such that $M_0(p_1) \leq M_0(p_5)$ in order to avoid deadlocks. For instance, if $M_0(p_1) = 4$ and $M_0(p_5) = 3$, the system comes to a total deadlock after the firing sequence $\sigma = t_1 t_2 t_1 t_2 t_1 t_2 t_1$; the corresponding 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, thus 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_{g_j} . We have the following theorem:

Theorem 1: The necessary and sufficient condition for the SME to be deadlock-free is that the number of tokens n^w to be inserted into a place, p_w , of a TT-generation, Π^w , from t_g of Π_g to $t_j \neq (t_C \text{ of } \Pi_j)$, is

$$n^w \geq \text{MAX}\left(\left[d_{gj} + 1 - M_{\max}(p_D)\right], 0\right). \quad (1)$$

Proof:

Necessary Condition: In an SME, deadlock occurs when the following situations happens: (a) all the $M_{\max}(p_D)$ tokens are trapped in a place of Π_j that is sequentially before t_j , (b) there is no token which can flow into the decision place p_D , and (c) there is no token in Π^w . If deadlock does not exist, then at least one of the above three situations dose not happen. By definition of the synchronic distance, up to d_{gj} tokens can be accumulated in Π_j . If Situation (a) does not happen, no token needs to be added in any place of Π^w , and there must be more than d_{gj} tokens in $M_{\max}(p_D)$ such that after putting d_{gj} tokens into Π_j , Π_g can still be activated at least once, using the remaining tokens in $M_{\max}(p_D)$; thus, not all the $M_{\max}(p_D)$ tokens are trapped in Π_j . Hence $M_{\max}(p_D) \geq d_{gj} + 1$ and $n^w = D$. If Situation (b) does not happen, then no token need to be added into Π^w , and there must be at least one more than d_{gj} tokens in p_D . Hence, $M_{\max}(p_D) \geq d_{gj} + 1$ and $n^w = D$. If (c) does not occur, then since token(s) can flow into Π^w from the execution of Π_g , then either there are at least $(d_{gj} + 1)$ tokens in p_D , (hence, no tokens need to be added to Π^w) or at least $[d_{gj} + 1 - M_{\max}(p_D)]$ token(s) are added to Π^w because if the number of tokens added to Π^w is insufficient, then the tokens untrapped from Π_j will flow through the home place and be trapped again in Π_j . Combining these three situations, one finds that if there is no deadlock in the SME, then Eq. (1) must be satisfied.

Sufficient Condition: There are two cases:

- (1) $M_{\max}(p_D) \geq d_{gj} + 1$: Assume there is no token in p_w , that is, $n^w = 0$. The maximum number of tokens which can be accumulated in Π_j is d_{gj} , by the definition of the synchronic distance. The remaining token(s) in p_D can flow through Π_g up to d_{gj} times without being trapped, putting at least a token into Π^w to release some token(s) from Π_j such that the tokens temporarily accumulated in Π_j can flow out. Because Π_g can execute up to d_{gj} times continuously, it can put up to d_{gj} tokens to Π^w , which is enough to flush out up to d_{gj} tokens in Π_j . Hence, there is no deadlock.
- (2) $M_{\max}(p_D) < d_{gj} + 1$: Given $n^w \geq d_{gj} - M_{\max}(p_D) + 1$, the maximum number of tokens which can be accumulated in Π_j is $M_{\max}(p_D)$. The $d_{gj} - M_{\max}(p_D)$ tokens in Π^w can be used to finish the remaining $d_{gj} - M_{\max}(p_D)$ executions of Π_j . The remaining (at least one) token can start the first execution of Π_g , which can put up to d_{gj} token(s) into Π^w and, thus, can flush out all the tokens accumulated in Π_j . Hence, there is no deadlock.

Using the example in Fig. 2, in which $p_D = p_6$ and $M_{\max}(p_D) = 1$, $d_{gj} = 3$, $n^w = M(p_5) = 3 + 1 - 1 = 3$, it is also easy to see that Theorem 1 is a generalized version of the sufficient deadlock free condition (DFC) in [5].

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

Note that the net shown in Fig. 2 can be synthesized using the knitting technique. It models a manufacturing system composed of two workstations and a shared robot (p_6) for loading/unloading these workstations. Thus, instead of analyzing a limited set of SMEs

[5], we can synthesize well-behaved PME and SMEs. The above net can be interpreted using the terminology of the knitting technique as follows. The single resource at p_6 is utilized by Process 1, $[p_6 t_1 p_2 t_2 p_6]$, and Process 2, $[p_6 t_3 p_4 t_4 p_6]$, which are exclusive to each other without the presence of other nodes and arcs. The net containing only these two processes is a 2-PME. This 2-PME can be sequentialized by adding processes $[t_2 p_3 t_3]$ and $[t_4 p_1 t_1]$. This results in a cycle, $[p_1 t_1 p_2 t_2 p_3 t_3 p_4 t_4 p_1]$, to which we must add a number of tokens to ensure that the net is live. Otherwise, none of the transitions is fireable and, hence, the net is deadlocked. By putting three tokens into p_1 , Process 1 executes before Process 2. Hence, they become sequentially mutually exclusive (SME), not only mutually exclusive, to each other. Note that Process 1 is executed before Process 2. Hence, they are no longer exclusive to each other. We can add tokens to p_3 , instead to p_1 , to maintain the liveness of the net. In this case, Process 2 is executed before Process 1. Now, we can add process $[t_3 p_5 t_2]$ to the net with p_5 holding three tokens, by Eq. (1), to complete the synthesis process.

$M_0(p_1)$ can be viewed as the number of available jobs and $M_0(p_5)$ as the number of empty slots available in W2. The requirement $M_0(p_1) \leq M_0(p_5)$ precludes large number of available jobs being dispatched to the system in a fixed time interval. Physically, workstation W1 has 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 which is needed by W3 to load the part from W2— a deadlock.

Note that prior to sequentialization, an SME is a LEX. We will 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, $t_g(t_j)$ may never fire and, thus, cause deadlock (unboundedness). Hence, Rule TT.3 must be applied. In the sequel, we will assume that all the SMEs under consideration are full and, without loss of generality, that, other than the intra-SME generations, all the TT-generations are from a control transition of a SME/LEX to a control transition in another SME/LEX.

TT-Generations from SME1 to SME2

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 the PSPs in the SME may be involved in the generation. Consider two SMEs, each containing a pair of PSPs with $d^m = 1$ [Fig. 7(a)]. We generate a TT-path from a PSP of the SME on the left to a PSP of the SME on the right. Each SME is a LEX prior to the sequentialization. Figs. 7(b)-(c) show two such cases. The former preserves the logical correctness while the latter causes a deadlock because both Π_1 and Π_3 or both Π_2 and Π_4 must execute to complete one iteration. In Fig. 7(b), PSPs Π_1, Π_3, Π_2 and Π_4 are executed in this order. However, in Fig. 7(c), if there is no token in p_8 , then Π_1 is waiting for the token in p_8 to be generated by the execution of Π_4 , which, in turn, is

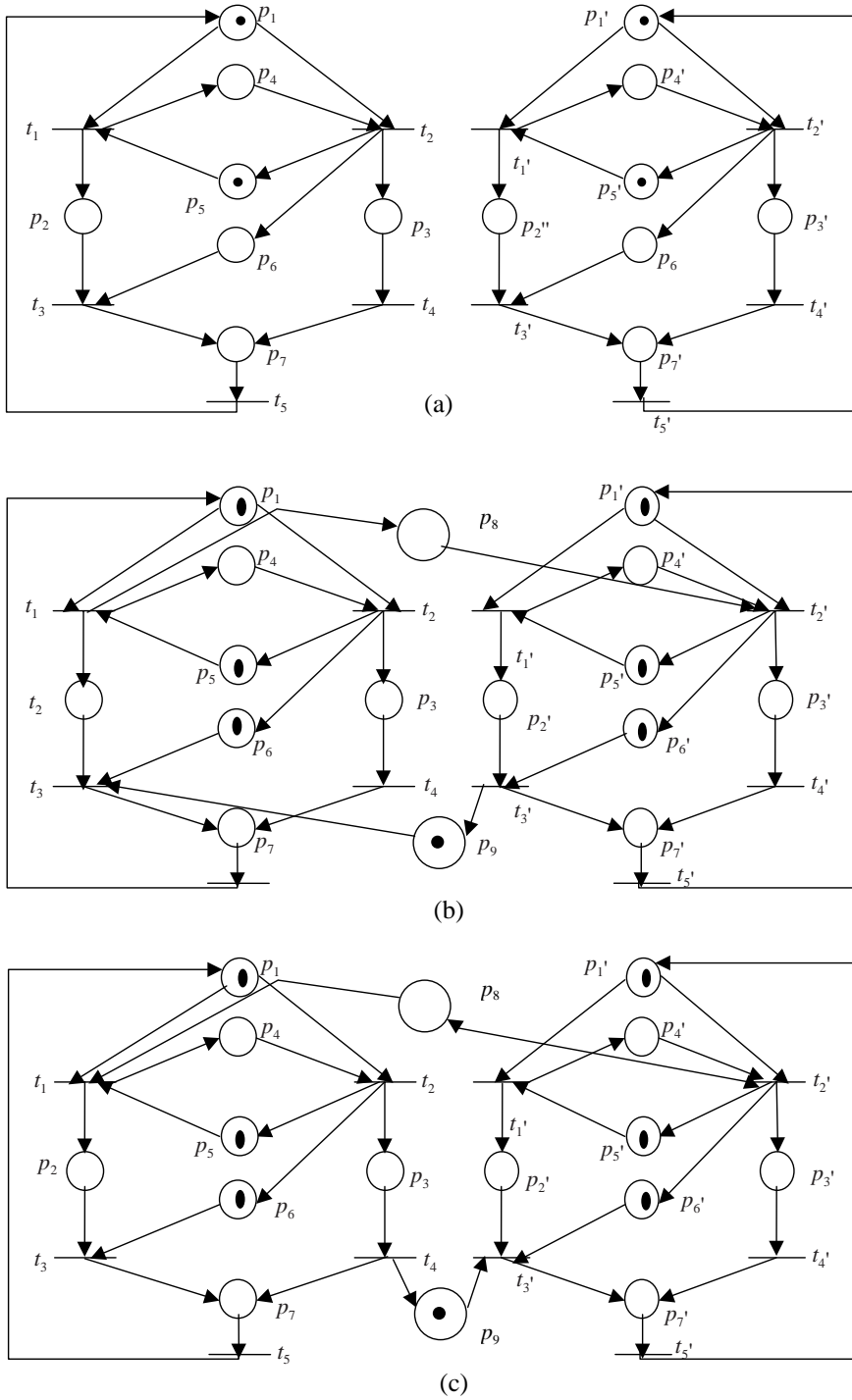


Fig. 7. (a) Two SMEs. (b) Synchronization of two SMEs with liveliness, boundedness and reversibility. (c) Synchronization of two SMEs with deadlock.

waiting for the execution of Π_3 . But Π_3 is waiting for the token in p_9 to be generated by the execution of Π_2 , and this Π_2 is waiting for the execution of Π_1 ; hence, a deadlock. It can be removed by putting one token in p_8 . Note that Rule TT.3 is relaxed in the sense that not all the PSPs in the SME need to be involved in the generation.

If t_g and/or t_j are in partial SMEs, then Rule TT.3 cannot be relaxed. Hence, in the sequel, we will only consider TT-generation between two full SMEs. Similarly, when the MSD among PSPs in a single SME may be greater than one, there is no ordering of executions of PSPs in an SME. The token at p_s of these PSPs can flow to any PSP in the SME, and Rule TT.3 seems not to be relaxed. This, however, is not true as will become clear later. In what follows, we will first show, in Lemma 1, that the ratio of the 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 for enough tokens to be injected into the Π^w to fire t_j , to avoid deadlock, and no persistent accumulation of tokens in the Π^w , to avoid unboundedness. Based on this, we have the following:

Lemma 1: After a TT-generation between two full SMEs, the necessary condition for preserving well-behavedness is $k = \lim_{\sigma_g \rightarrow \infty} \frac{\sigma_g - \sigma_j}{\sigma_g} = 0$. If $k > 0$ (< 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 $\sigma_g \rightarrow \infty$, an infinite number of tokens is accumulated in the Π^w , and the synthesized net is unbounded. Similarly, if $k < 0$, then the Π^w needs to have an infinite number of tokens in the initial marking to enable t_j 's to fire an infinite number of times, implying that the synthesized net has a deadlock. Hence, the necessary condition for preserving well-behavedness is $k = 0$.

We will discuss only the case of MSD $d^m = 1$ among PSPs in an SME; the case of $d^m > 1$ will be left for future research.

When $d^m = 1$, 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 let a and b be the number of PSPs in SME1 and SME2, respectively.

Based on Lemma 1, we have the following theorem.

Theorem 2: 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, Π_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 MSD among PSPs in each SME is one, Π_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 $k = \lim_{\sigma_g \rightarrow \infty} \frac{\sigma_g - \sigma_j}{\sigma_g} = \frac{\tau_j}{\sigma_g} \left(\frac{1}{a} - \frac{r}{b} \right) \neq 0$ for any integer r since neither a nor b is a factor of the other. Because in the above, τ and σ are infinitely large, finite changes of τ will 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., $X_{gj} = \text{SME1}$ and $X_{jg} = \text{SME2}$.

Theorem 3: If $\frac{b}{a}$ is an integer, then applying Rule TT.3 by making $X_{gj} = \Pi_g$ and X_{jg} a set containing arbitrary $\frac{b}{a}$ PSPs of SME2 preserves well-behavedness.

Proof: By making $r = \frac{b}{a}$ in the proof of Theorem 2, 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 Π_j of SME2 and a set containing arbitrary $\frac{a}{b}$ PSPs of SME1 preserves well-behavedness.

In the above discussion, we considered only the case where X_{gj} is only a PSP in SME1. A generation with multi-PSP X_{gj} can be decomposed into the ones of single-PSP X_{gj} 's. Thus, in the remainder of this paper, all the theorems and lemmas will be for generations of single-PSP X_{gj} only.

Note that tokens may have to be inserted into Π^w to avoid deadlock. This happens when a PSP in X_{jg} is about to execute but there is no token in Π^w to support such execution because the PSP in X_{gj} fires too late to supply tokens. The number of tokens to be inserted into Π^w depends on the X_{gj} and X_{jg} selected. The following lemma helps us to develop procedure *Token_Added* below. Let $X_{gj} = \{\Pi_{g_y}\}$ and $X_{jg} = \{\Pi_{j_x}, x = 1, 2, \dots, r\}$. Thus, only one PSP in SME1 is involved in the generation. PSPs in SME1 execute one by one in the sequence of $\Pi_{g_1}, \Pi_{g_2}, \dots, \Pi_{g_a}$. Thus Π_{g_y} executes the y -th time in SME1. j_x is an index indicating the order in which Π_{j_x} executes in SME2.

Lemma 2: If no tokens need to be inserted into Π^w generated from Π_{g_y} to $X_{jg} = \{\Pi_{j_x}, x = 1, 2, \dots, r\}$, then $\forall x, j_x > (x - 1) * a + y$ for the case in which $r = \frac{b}{a}$ is an integer.

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

For the case in which $r = \frac{a}{b}$ is an integer, similar results can be obtained. Let $X_{gj} = \{\Pi_{g_y} \mid y = 1, 2, \dots, r \text{ and } r = \frac{a}{b}\}$ and $X_{jg} = \{\Pi_{j_x}\}$. Again, PSPs in X_{gj} must execute earlier than Π_{j_x} which is just the reverse of the case for $b > a$. That is, $g_y < g'_y$ (note that the sign changes from $>$ to $<$). The remaining discussion will follow that in Lemma 4.

Lemma 3: If no tokens need to be inserted into the Π^w generated from $X_{gj} = \{\Pi_{g_y} \mid y = 1, 2, \dots, r \text{ and } r = \frac{a}{b}\}$ to $X_{jg} = \{\Pi_{j_x}\}$, then $\forall x, g_y < (y - 1) * a + x$ for the case in which $r = \frac{a}{b}$ is an integer.

Procedure *Token_Added* for $b > a$ is presented below. Let $X_{gj} = \{\Pi_{g_y}\}$ and $X_{jg} = \{\Pi_{j_x} \mid x = 1, 2, \dots\}$. 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 ( $X_{gj}, X_{jg}$ ) {
    Token_Added = 0;
    for { $x = 1; x < r + 1; x++$ }
    {
        if ( $j_x < ((x - 1 - \textit{Token\_Added}) * a + y)$ )
            Token_Added++;
    } } .

```

Similarly, procedure *Token_Added* for the case in which $\frac{a}{b}$ is an integer exactly parallels that for the case in which $\frac{b}{a}$ is an integer except that the sign changes from “<” to “>” in the “if” statement. We can prove the correctness of the procedure as follows.

Theorem 4: Procedure *Token_Added* calculates the minimum number of tokens to be added to maintain well-behavedness.

Proof: Consider first the case in which $\frac{b}{a}$ is an integer. The proof for the case in which $\frac{a}{b}$ is an integer is similar. The procedure checks each Π_{j_x} starting from $x = 1$. If it cannot be executed and the synthesized net comes to a deadlock, then the procedure adds a token to Π^w . In the end, the number of tokens added is the minimum. We can then prove that the condition for adding a token at the x -th step is $j_x < [(x - 1 - \textit{Token_Added}) * a + y]$ 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 Π^w prior to the x -th step. At the beginning of the x -th step, we need x tokens to support the executions of all Π_{j_θ} , $\theta = 1, 2, \dots, x$. But Π^w already has “*Token_Added*” tokens; thus, Π_{g_y} must have executed $(x - 1 - \textit{Token_Added})$ times earlier than $\Pi_{j_{x-1}}$. The $(x - \textit{Token_Added})$ -th execution of Π_{g_y} adds a token into Π^w to be consumed by the execution of Π_{j_x} . The former must coordinate with the execution of $\Pi_{j'_x}$ to complete one iteration where $j'_x = (x - 1 - \textit{Token_Added}) * a + y$. As in the proof of Lemma 4, if $j_x > j'_x$, then the above token can be consumed by the execution of Π_{j_x} . Otherwise, a token must be added at the x -th step. The theorem is, thus, proved.

As an example, consider the case where $a = 2$, $b = 4$, $d_1 = d_2 = 1$, and new paths are generated, using Rule TT.3, from Π_{g_2} to Π_{j_1} and Π_{j_2} . Thus, we have $y = 2$, $j_1 = 1$, and $j_2 = 2$. We start from j_1 ($x = 1$). Initially, Π_{g_1} and Π_{j_1} must both execute to avoid deadlock. Π_{j_1} , however, cannot execute because Π_{g_2} has not executed to inject a token into Π^w . Thus, we add a token to Π^w and set the variable *Token_Added* = 1. Next, consider Π_{j_2} ($x = 2$). Now, the token to be injected by executing Π_{g_2} is needed to support the execution of Π_{j_1} rather than Π_{j_2} if *Token_Added* = 0. Thus, we compare $j_2 = 2$ to $(x - 1 - 1) * a + y = 2$ [rather than $(x - 1) * a + y$]; it can be seen that there is no need to add a token. j_x no longer needs to be considered; hence, the total number of tokens to be inserted into Π^w is one.

When more than one PSP in X_{gj} is needed, we can apply the above algorithm to each set of {1 process in SME1, r processes in SME2}.

TT-Generations Between an SME and a LEX

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

Synthesis-Synchronization TT Rule

In this subsection, we will present the extended TT rules to cover the aforementioned three kinds of TT generation as follows.

For a Π^w from $t_g \in \Pi_g$ to $t_j \in \Pi_j$ generated by the designer,

- (1) TT.1 **If** $t_g \mid t_j$ and t_j is a control transition or only if one of them is in a circle which was solely generated using Rule PP.1, **then** signal “forbidden, delete Π^w ” and return **else** continue.
- (2) TT.2 **If** $\Pi_g = \Pi_j$, **then** signal “a pure *TT generation*”; otherwise, signal “an interactive *TT generation*”.
- (3) TT.3 (1) **If** $t_g \leftarrow t_j$ or $t_g = t_j$, and without firing t_j , there does not exist a firing sequence σ to fire t_g , **then** insert some tokens and signal “forming a new circle” **else**
 - (2) **If** $t_g \uparrow t_j$, and $d_{gj} > 1$, **then** insert at least $\text{MAX}([d_{gj} + 1 - M_{\max}(p_D)], 0)$ tokens in a place of Π^w .
If $\Pi_g = \Pi_j$, **then** return, and the designer may start a new generation.
- (4) TT.4 or completeness rule 1
 - (a) TT.4.1 **If** $d_{gj} = 1$, **then** signal “both LEXs are full SMEs”, $X_{gj} = \{\Pi_{g_c} \mid c = 1, 2, \dots, |X_{gj}|\}$, **and** $X_{jg} = \{\Pi_{j_d} \mid d = 1, 2, \dots, |X_{jg}|\}$.
 - (a.0) TT.4.1.0 **If** $\exists X'_g$ **and** X'_j such that $X_{gj} = X'_g$ **and** $X_{jg} = X'_j$, **then** signal “forbidden, delete Π^w ”.
 - (a.1) TT.4.1.1 **If** $\frac{a}{b} = r_1$, where r_1 is a positive integer, **then** **if** $|X_{gj}| \neq r_1$, **then** signal “forbidden, delete Π^w ”;
 - (a.2) TT.4.1.2 **Else** **if** $\frac{b}{a} = r_2$, where r_2 is a positive integer, **then** **if** $|X_{jg}| \neq r_2$, **then** signal “forbidden, delete Π^w ”;
 - (a.3) TT.4.1.3 **Else**/* neither $\frac{a}{b} = r_1$, nor $\frac{b}{a} = r_2$ */ signal “forbidden, delete Π^w ”.
 - (b) TT.4.2 **If** $|X_{gj}| > 1$, **then** generate a *TP – path* from a transition t_g of each Π_g in X_{gj} to a place p_k in the Π^w .
 - (c) TT.4.3 **If** $|X_{jg}| > 1$, **then** generate a virtual *PT – path* from the place p_j to a transition t_j of each Π_j in X_{jg} .
 - (d) TT.4.4 **If** $X_g \neq \text{SME1/LEX1}$, **then** insert $\text{Token_Added}(X_{gj}, X_{jg}) - M(p_j)$ tokens in p_j .
- (5) TT.5 **If** $d_{gj} = \infty$ ^{*5}, **then**

*5. There was no path from t_g to t_j prior to this generation or both are in a circle which was solely generated using Rule PP.1 or $t_g \dagger t_j$.

- (a) TT.5.1 Generate a new *TT-path* from t'_g to t'_j (on the circles containing t_g and t_j , respectively) to synchronize t_g with respect to t_j so that after (if necessary) Step 4, t_g cannot fire infinitely often without t_j firing once. In the case of $t_g \dagger t_j$, make $t'_g = t_j$ and $t'_j = t_g$, and the new *TT-path* may contain output transitions of p'_s .
- (b) TT.5.2 Go to Step 3.

Note that $M(p_j)$ is the number of tokens added to p_j if Rule TT.2 was applied during the synthesis. Because t_g 's and t_j 's are executed sequentially, a t_j may not be able to fire because there are no tokens in p_j resulting from the late firing of t_g . Procedure *Token_Added* calculates the total number of occurrences of such token deficiency which returns the total amount of tokens required when there are no tokens in p_j prior to the application of Rule TT.3. Note that there is only one p_j after one application of Rule TT.3. Multiple p_j 's can be produced by repeatedly applying Rule TT.3. Thus, Rule TT.3, in general, produces nets with all possible interactions between two SMEs.

We will prove the following theorem.

Theorem 5: The Petri net constructed according to the synthesis-synchronization rules preserves the well-behaved properties.

Proof: Rules TT.1, TT.2 and part (1) of TT.3 have been shown to be correct. By Theorem 1, TT-generations within an SME, the least number of tokens to be inserted into Π^w is $MAX([d_{gj} + 1 - M_{max}(p_D)], 0)$, to avoid deadlock. This guarantees that the two synthetic guidelines will be satisfied. Part (2) of Rule TT.3 is, thus, proved to be well-behaved.

The correctness of Rule TT.4.1 is proved by Theorem 3. Rules TT.4.2 and TT.4.3 have the same meaning as do the synthetic Rules TT.3.1 and TT.3.2, whose correctness has been verified. The correctness of Rule TT.4.4 is verified in Theorem 4. Hence, Rule TT.4 has been proved to be correct.

Rules TT.5.1 and TT.5.2 have the same meaning as do the synthetic Rules TT.4.1 and TT.4.2, whose correctness has been verified. We have thus proved that the Petri net constructed according to the synthesis-synchronization rules preserves the well-behaved properties.

5. ILLUSTRATION OF SYNTHESIS-SYNCHRONIZATION RULES

Example: Fig. 8 illustrates the use of the synthesis-synchronization rules discussed in the last section. There are two SMEs in Fig. 8. SME1 includes processes $\Pi_1 (= [t_2 p_3 t_3])$ and $\Pi_2 (= [t_4 p_6 t_5])$, and SME2 contains processes $\Pi_3 (= [t_6 p_7 t_7])$, $\Pi_4 (= [t_8 p_8 t_9])$, $\Pi_5 (= [t_{10} p_9 t_{11}])$, and $\Pi_6 (= [t_{12} p_{10} t_{13}])$. In this example $a = 2$ and $b = 4$; thus, $r_2 = 2$. $X_{gj} = \{\Pi_2\}$ and $X_{jg} = \{\Pi_3, \Pi_4\}$. Hence, the cardinalities $|X_{gj}|$ and $|X_{jg}|$ are 1 and 2, respectively. Rules TT.1 and TT.4.3 are used to generate paths $[t_5 p_{17} t_7]$ and $[p_{17} t_9]$, respectively. Rule TT.5 is not applied since $d_{gj} = 1$, rather than ∞ .

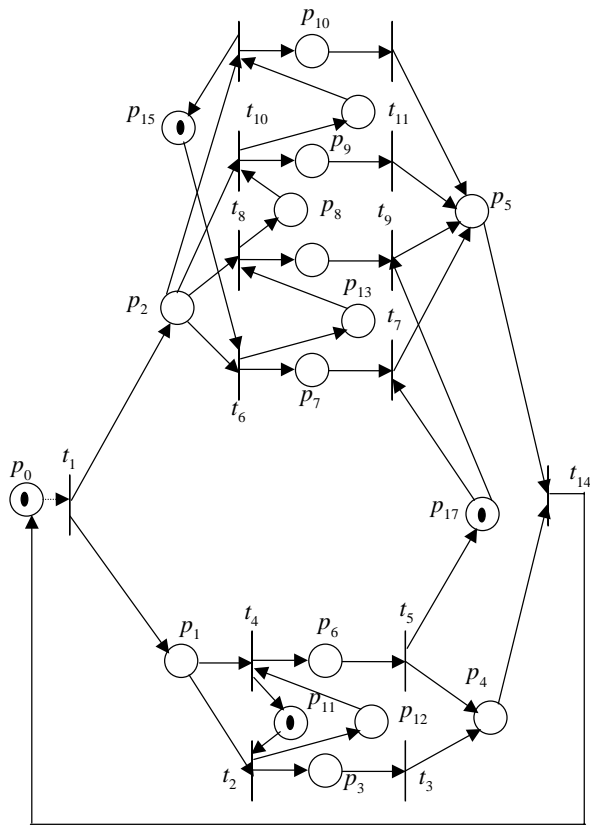


Fig. 8. Example of synthesis-synchronization rules.

6. CONCLUSIONS

The synthesis-synchronization rules of the knitting technique provide sufficient guidelines for a designer to devise new concurrent and synchronized systems incrementally while maintaining logical correctness, namely, unboundedness, liveness, and reversibility. Therefore, the cycle of reanalysis-remodification for logical correctness after the design stage does not need to be carried out. The knitting technique has been extended to synthesis of SMEs and generation of new classes of nets whose properties are summarized below:

- (1) marking nonmonotonic;
- (2) arbitrary finite positive synchronic distances among mutually exclusive PSPs;
- (3) interactions of an SME with another SME or LEX.

This work has included the marking assignment in the synthesis for the purpose of synchronization. The former knitting technique only considered the structural relationships among the PSPs and was independent of marking, except for the very limited case of backward generation.

These new rules render the synthesis-based analysis more powerful as more design errors and their sources can be discovered. Any detected rule violation implies potential ill-design. Similarly, the reverse process of reduction has become more powerful as more paths can be deleted. Any well-behaved PN violating some rules prompts us to discover new rules.

Future work should be directed toward removing other synthetic restrictions, such as TP and PT generations, and other synchronization restrictions, such as classes more general than SMEs. The way to achieve this is to develop the DFC(s) for more general cases. We should also extend the invariant-search technique in [3] to new classes of nets. The CAD tool reported in [18] should also be updated to allow more complicated PNs to be constructed according to the new rules.

REFERENCES

1. Y. Yaw, (now D.Y. Chao), "Analysis and synthesis of distributed systems and protocols," Ph.D. Dissertation, Dept. of EECS, U.C. Berkeley, 1987
2. F.L. Foun, "The algorithm of a synthesis technique for concurrent systems," *IEEE International Workshop on Petri Nets and Performance Models, Tokyo*, 1989, pp. 266-276.
3. D.Y. Chao and D.T. Wang, "Two theoretical and practical aspects of knitting technique - Invariants and a new class of Petri net," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 27, No. 7, 1997, pp. 926-937.
4. D.Y. Chao, "Linear algebra based verification of well-behaved properties and P-invariants of Petri nets synthesized using knitting technique," *MIS Review*, Vol. 5, 1995, pp. 27-48.
5. M.C. Zhou and F. DiCesare, "Parallel and sequential mutual exclusions for Petri net modeling for manufacturing systems with shared resources," *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 4, 1991, pp. 515-527.
6. R. Lipton, "The reachability problem requires exponential space," Report No. 62, Department of Computer Science, Yale University, 1976.
7. D.Y. Chao, "X-Window implementation of an algorithm to synthesize ordinary Petri nets," *Journal of National Cheng-Chi University*, Vol. 73, No. 2, 1996, pp. 451-496.
8. D.Y. Chao and D.T. Wang, "A synthesis technique of general Petri nets," *Journal of Systems Integration*, Vol. 4, No. 1, 1994, pp. 67-102.
9. J. Esparza and M. Silva, "On the analysis and synthesis of free choice systems," *Lecture Notes in Computer Science, Advances in Petri Nets*, 1990, No. 483, Springer-Verlag, pp. 243-286.
10. J. Esparza and M. Silva, "Circuits, handles, bridges, and nets," *Lecture Notes in Computer Science, Advances in Petri Nets*, No. 524, 1991, Springer-Verlag, pp. 210-242.
11. A. Datta and S. Ghosh, "Synthesis of a class of deadlock-free Petri nets," *Journal of ACM*, Vol. 31, No. 3, 1984, pp. 486-506.

12. "Modular synthesis of deadlock-free control structures," *Foundations of Software Technology and Theoretical Computer Science*, No. 241, G. Goos and J. Hartmanis(ed.), Springer-Verlag, 1986, pp. 288-318.
13. Y. Chen, W. T. Tsai and D. Y. Chao, "Dependency analysis – a compositional technique for building large Petri net", *IEEE Transactions on Parallel and Distributed Systems*, Vol 4, No. 4, 1993, pp. 414-426.
14. I. Suzuki and T. Murata, "A method for stepwise refinements and abstractions of Petri nets," *Journal of Computer and System Science*, Vol. 27, No. 7, 1983, pp. 51-76.
15. R. Valette, "Analysis of Petri nets by stepwise refinement," *Journal of Computer and System Sciences*, Vol. 18, No. 1, 1979, pp. 35-46.
16. C.V. Ramamoorthy, S.T. Dong and Y. Usuda, "The implementation of an automated protocol synthesizer (APS) and Its Application to the X.21 Protocol," *IEEE Transactions on Software Engineering*, Vol. 11, No. 9, 1985, pp. 886-908.
17. T. Murata, "Petri nets: properties, analysis and applications," *IEEE Proceedings*, Vol. 77, No. 4, 1989, pp. 541-580.
18. D.Y. Chao and D.T. Wang, "An interactive tool for design, simulation, verification, and synthesis for protocols," *Software-Practice and Experience, an International Journal*, Vol. 24, No. 8, 1994, pp. 747-783.



D.Y. Chao (董毅) received the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 1987. From 1987-1988, he worked at Bell Laboratories. In 1988, he joined the computer and information science department of New Jersey Institute. In 1994, he joined the MIS department of NCCU as an associate professor. In February, 1997, he was promoted to be a full professor. His research interests are in the application of Petri nets to the design and synthesis of communication protocols and CAD implementation of a multi-function Petri net graphic tool. He is now exploring the properties of a new class of Petri nets and implementation of several CAI tools based on Visual C++. He has published 81(including 20 journal) papers in the areas of communication protocols, Petri nets, DQDB, networks, FMS, data flow graphs and neural networks.