X WINDOW IMPLEMENTATAION OF AN ALGORITHM TO SYNTHESIZE ORDINARY PETRI NETS

Chao, D. Y.

趙 玉 *

摘 要

平行系統之證明極爲複雜，合成法用不著驗證，因此可減輕此複雜度之問題，現今大部合成技術，並不討論電腦輔助設計，很少的電腦輔助工具具有整合分析、減少、模擬、查詢及合成的能力，派翠圖可模擬及分析平行系統。

本人前曾發展一組合成律，可逐漸加大派翠圖，同時確保其邏輯正確無誤。

本文指出所合成的派翠圖形成一新的種類，並提出時間矩陣的概念，其可記錄程序間之關係如平行、互斥、及序列。

基於這一觀念，吾人發展出演算法，去決定何時可用此合成法，查覺違反此合成法的情況，及重新計算此一矩陣，此演算法之複雜度是 $O(K^3)$ ， K 是程序的總數。吾人前所發展之 X 視窗工具，以將此演算法具體實踐，此工具可設計、分析，模擬測試及合成通訊協定。

關鍵詞：派翠圖，平行系統，自動製造系統，活性，有限性，死結，合成，程序，區域平行集，區域互斥集，結構關係，時間矩陣，演算法，法則，X 視窗，電腦輔助工具。

Abstract

Vertification of concurrent systems faces complexity problem. Synthesis relieves this by avoiding verification. Most existing synthesis techniques do not deal with CAD tool. Few tools are able to integrate analysis, reduction, simulation, query, and synthesis in one software package as ours. Petri nets are used for modelling and analyzing concurrent systems. A set of synthesis rules were developed *earlier*[1-3] for incrementally generating new processes without incurring logical incorrectness. This paper shows that the synthesized nets form a new class of nets and presents the concept of temporal matrix which records relationship (*concurrent, exclusive, sequential,* ... etc.) among processes. Based on this concept, we develop an algorithm to determinie the applicable rules, detect rule violations, and update the matrix. The complexity for the algorithm is O (K^3) where K is the total number of processes. This algorithm has been incorporated int our X-Window based tool for the design, analysis, simulation, testing, and synthesis for communication protocols and others.

* 作者爲本校資訊管理學系副教授

Key words: Petri net, concurrent system, automated manufacturing system, liveness, boundedness, deadlock, synthesis, processes, local concurrent set, local exclusive set, structural relationship, temporal matrix, algorithm, rules, X-Window, CAD tool.

I. INTRODUCTION

Many *authors*[4-6] have proposed Petri nets (PNs) to model concurrent software. As the system grows in complexity, this modeling process and the reachability analysis become complicated. To address this issue, researchers proposed two directions: *synthesis*[7-29] and *reduction*[30-42]. Synthesis focuses on establishing conditions of augmenting existing PNs to form larger nets so that (1) the system specification can be correctly modelled and (2) the resulting net will retain the *well-behaved* properties such as reversibility, liveness, and boundedness. Reduction, on the other hand, emphasizes on reducing the size of a given PN while preserving properties.

According to *Jeng*[43], two dominant synthesis approaches are bottom-up and top-down. Bottom-up approaches start with decomposition of systems into subsystems, construct subnets for subsystems, and merge these subnets to reach a final PN by sharing *places*[15-16] *transitions*[44], and/or elementary *paths*[14,17-18] or by linking *subnets*[45]. Top-down approaches begin with a first level PN and refiine it to satisfy system specifications until a certain level is reached. The sptewise refinement technique by *Vallette*[30] is a top-down approach where transitions are replaced by well-formed blocks. Suzuki and *Murata*[32] generalizd this technique with the disadvantage of the need to analyze the blocks. Our knitting technique supplements this approach by synthesizing the blocks without analysis.

Based on top-down refinement of activities and bottom-up modeling of shared resources, Zhou *et al*[46-48] have recently formulated a hybrid synthesis approach. Parallel and sequential mutual exclusions are used for resource-sharing modeling. The design of first-level Petri nets is also discussed. However, the validation for mutual exclusions can be a formidable task for large systems.

To automate the synthesis, rules must be developed as in Esparza and *Silva*[20] and Datta and *Ghosh*[34] to synthesize free-choice (*FC*) and extended FC (*EFC*). But they are unable to synthesize assymetric-choice nets (*AC*) and one needs to *analyze*[20] whether the subnet is reducible. Further, they do not have explicit algorithms and the associated complexity. Thus, most techniques do not deal with computer aided design (CAD) explicitly. We devise some simple and yet effective rules to guide synthesis, e.g., for communication *protocol*[1] and automated manufacturing *systems*[3]. It requires no analysis and the synthesized nets constitute a new class of nets that are more generall than *FC*, EFC and *AC*[53]. This new class of nets is referred to as the synchronized-choice *nets*[49] with the special property that any pair of concurrent (exclusive) processes must spring from and join at transitions (places).

The knitting technique expands PN in a structural fashion. While it takes exponential time to determine marking propoerties; it may take polynomial time to determine structural properties. It aims to find the fundamental constructions for building any PNs. There are two advantages: (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, it adds new paths to a PN, N^1, producing a larger N^2 . The generations are performed in such a fashion that all reachable markings in N^1 remain unaffected in N^2; hence all transitions and places in N^1 stay live and bounded, respectively. N^2 is live and bounded by making the new paths (*NP*) live and bounded. This notion is novel compared with other approaches.

Designers start with a basic process modeled by a set of closed-loop sequentially-connected places and transitions with a home-place marked with some tokens representing resources. Then they can add parallel and exclusive processes according to the system specification or semantics or add closed loops for the operations according to the resources required. The knitting technique is so called because expansions are conducted among the nodes in a global way. The synthesis rules guarantee incremental correctness at each generation step. Analysis can be avoided while designers can still build up a PN model for a complicated system. Due to the simplicity of the rules, it is easily adapted to computer implementation for rendering the synthesis of PNs performed in a user-friendly fashion.

There are four types of path generations: **transition-transition (TT), place-place (PP), transition-place (TP), and place-transition (PT)**; namely from a transition or place to a transition or place. One of the following three actions are taken depending on the type of generation: (1) forbidden, (2) permitted, (3) permitted but need more generations. For instance, a *TT* (*PP*) generation is permitted between two sequential or concurrent (exclusive) processes; while *TP* (*PT*) generations are forbidden since they may create unbounded (nonlive) nets. The rules are complete in the sense that all possible generations have been considered.

Recently, we have upgraded the knitting technique by relaxing some forbidden rules. For instance, we allow *TT* generations between exclusive *process*[24] and permit *TP* and *PT* *generations*[25] to generate more complicated classes of nets.

Thus, we can continuously enhance the rules by allowing new generations formerly forbidden. **Ideally, if all possible generations are allowed, we then could synthesize all classes of nets.**

A Temporal Matrix (T-Matrix) is proposed in this paper to record the relationships (*concurrent, exclusive, sequential,* ... etc.) among the modeled processes. Tracking all the process and the relationships among processes in large PNs is a difficult task and thus makes desirable the automatic tracking of rule applicability upon generation and the automatic updating of the T-matrix.

This paper develops such an algorithm and its X-Windows implementation. In addition, the T-Matrix can record self-loops and find maximum concurrency with linear time complexity which helps for processor assignments.

The knitting rules are useful for analysis and *reduction*[21-26]. For a given PN, we construct its T-Matrix and check mutual relationships among *PSP*s against the rules. Any violation spots potential ill-designs. The reverse process of removing *PSP*s — reduction, according to the 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, besides the possibility of continuous enhancement, that while reducing, it can discover wrong designs and suggest how to fix the problem based on the knitting rules. Other enhancements are as follows. We

- extend the rules for synthesizing General Petri Nets (*GPN*[21]s);
- find invariants of synthesized *nets*[26];
- show that the synthesized nets is a new class of *nets*[49];
- apply the knitting technique to reduction and incorporate it into the tool; and
- add more semantics in addition to "concurrency", "exclusiveness", "iteration", "message exchange", and "process switching" by allowing the designer to assign statements in "C" language to each *transition*[22]. They can be compiled for simulation and allow automatic source code generation. This enhances the semenatic power of the tool.

In short, this work overcomes some drawbacks of most existing synthesis approaches; i.e., they do not

- deal with the algorithm and CAD tool using graphical user interface for synthesis explicitly;
- show how to continuously update their synthesis techniques;
- indicate how to extend the synthesis for analysis;
- show temporal relationships among processes after synthesis;
- find the maximum concurrency of the synthesized net; and
- they synthesize very limited classes of nets;

Most important, none of the current synthesis approach employs the notions of relationships of *concurrent, exclusive, sequential,* ... etc. and path generations which mark the most distinct features of the knitting technique. **Notations** are provided in **Appendix XIV**.

II. PRELIMINARIES

(A). Terminology

A Petri *net*[50-51] is a directed bipartite graph consisting of two types of nodes: *places* (represented by circles) and *transitions* (repressented by bars). Places represent conditions or the presence of raw materials and transitions represent events. Each transition has a certain number of input and output places indicating the preconditions and postconditions of the event. The holding of the condition or a raw material in a place is indicated by a *token* (represented by a dot) in the place. The system status is represented by the holding of a pattern of tokens in places which is called a *marking*.

Definition 1: Let

P=[p1, p2, ..., pa], T=[t1, t2, ..., tb], with $P \cup T = \phi$ and $P \cap T \neq \phi$;

I: PxT $\rightarrow$ {0, 1};
O: TxP $\rightarrow$ {0, 1};
M_0: P $\rightarrow$ {0, 1, 2, ...},
then N=(P, T, I, O, M_0) is an ordinary marked Petri net.

In this definition, pi $(1 \leq i \leq a)$ is called a place, ti $(1 \leq i \leq b)$ a transition, I an input function defining the set of directed arcs from P to T, O an output function defining the set of directed arcs from T to P, and M_0 an initial marking whose ith component represents the number of tokens in place pi. Note the functional values of both I and O are restricted to 0 and 1. If other positive ineger values are allowed, then the N is a General PN.

Definition 2: The firing rules are:

- A transition t ϵ T is enabled if and only if the marking at p, m(p)>0, $\forall$ p ϵ P such that I(p,t)=1;

- An enabled transition t fires at marking M' (with components $m(p)$, yielding the new marking M (with components $m'(p)$,

$$m(p) = m'(p) + O(p,t) - I(p,t), \forall\ p \in P.$$

The marking M is said to be reachable from M'. Given N and its initial marking M_0, the reachability set R(N,M_0) (abbreviated as R) is the set of all markings reachable from M_0.

Definition 3: A marked Petri net N is B-bounded if and only if $m(p) \leq B$, $\forall\ p \in P$ and $M \in R(N,M_0)$ where B is a positive ineger. If B=1, N is safe. N is live if and only if $\forall\ t \in T$, and $\forall\ M \in R(N,M_0)$, $\exists$ a firing sequence σ of transitions to lead 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. The synchronic distance between t1 and t2 in N is defined as d_{12} = Max $\{\sigma(t_1) - \sigma(t_2), \sigma \in L(N,M)\}$, where $\sigma(t)$ is the number of times t appears in σ and L(N,M) is the set of all firing sequences from M. $d(W,V)$ is the maximum number of firings of transitions in set W without any transition firing in set V. □

The synchronic distance is a measure of dependence between *transitions*[52].

Definition 4: A node x in N=(P,T,I,O,M_0) is either a $p \in P$ or a $t \in T$. The post-set of node x is $x\bullet=\{y|\exists$ an arc $(x, y)\}$ and its pre-set $\bullet x=\{y|\exists$ an arc $(y, x)\}$. An directed elementary path (DEP) in N is a sequence of nodes: $[n_1n_2...n_k]$, such that $n_i \in \bullet n_{i+1}$ $1 \leq i < k$ if k>1, and $n_i=n_i$ implies that i=j, $\forall 1 \leq i, j \leq k$. An elementary cycle in N is $[n_1n_2...n_k]$, k>1 such that $n_i=n_j$, $1 \leq i \leq j < k$, implies that i=1 and j=k.

Definition 5: An A-path is a DEP whose places initially have no tokens.

Definition 6: A basic process is defined as elementary cycle $[n_1n_2...n_k]$ in a PN where n_1 is a place holding tokens.

Definition 7: Let $t_a \in T$ be enabled in M_0, $\forall p_h \in \bullet t_a$, p_h is a *home place.* The set of home places is denoted as H = $\{p_h|p_h$ is a home place$\}$.

Definition 8: A *pseudoprocess (PSP)* in a PN is a DEP $[n_1 n_2 \ldots n_k]$, $k \geq 2$ such that $|\bullet n| = |n \bullet| = 1$, $2 \leq i \leq k-1$, or

k=2. The *PSP* is termed a **virtual PSP (VP)**.

$n_g(n_j) = n_1(n_k)$ is defined as the **generation point (joint)**. $\dot{n}_g(\dot{n}_j) = n_2(n_{k-1})$ is defined as the **next generation point (next joint)** □

Fig. 1 shows a basic process with p1 as a home place. In Fig. 2(3), [t1 p2 t2 p3 t3] ([p1 t1 p2]) is a *PSP* whose n_g = t1 (p1) and n_j = t3 (p2) respectively. Note the VP in Fig. 3 (Π_1) contains only n_g and n_j.

n_g, n_j, $\dot{n}_j$ are also used in new-path generations.

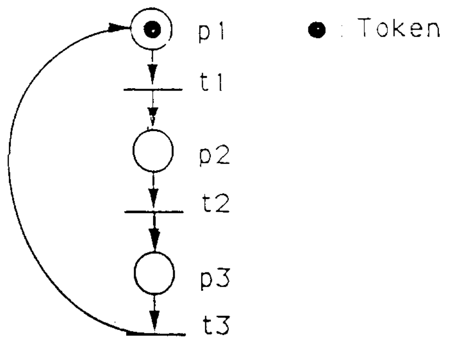


Fig. 1. A basic process.

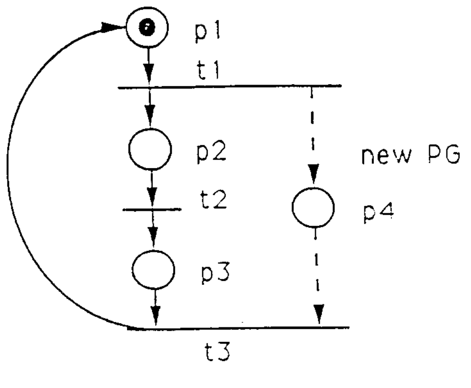


Fig. 2. Create a Pure-Generation (dashed line) by using TT rule.

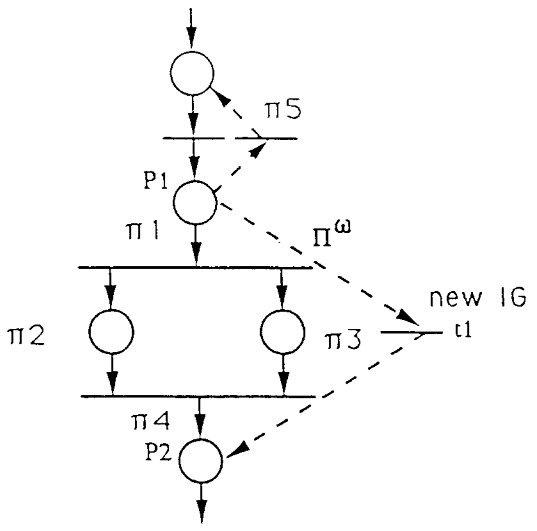


Fig. 3. New Interaction-Generation (IG) connecting Π1 and Π4.

Definition 9: Let *NP* be a set of new paths generated from a set of nodes called generation points (n_g) in N^1 to another set of nodes called joints (n_j) in N^1 to produce N^2. $\dot{n}_g$ ($\dot{n}_j$) is a node $n_k \in NP$ and $n_k \in n_g\bullet$ ($\bullet n_j$). □

Definition 10: 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 is an A-path, 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 ΠP_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 o Π_2; i.e., they are cyclic (CL) to each other.

Definition 11: Let DEP1$=[n_{ps}n_2 \ldots n_k\Pi_1]$ and DEP2$=[n_{ps}n_{2'} \ldots n_{k'}\Pi_2]$, where DEP1 $\cap$ DEP2$=\{n_{ps}\}$, Π_1 and Π_2 are two *PSP*s, and there are no DEPs (called bridges) from a node ($\neq n_{ps}$) DEP1 to a node ($\neq n_{pss}$) of DEP2 and vice versa. n_{PS} is called a **prime start node** n_{ps}. If $n_{ps} \in$ P, then $\Pi_2 | \Pi_2$. $\Pi_1 || \Pi_2$ if none of $\Pi_1 | \Pi_2$, $\Pi_1 \leftrightarrow \Pi_2$, and Π_1 o Π_2 hold.

Note $\Pi_1 || \Pi_2$ implies $n_{ps} \in T$ 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 that cannot be synthesized using the TT and PP rules.

The above definitions of *sequential earlier* and *later* seem to be also *marking related*. However, in the synthesized net, any cycle contains *at most* a place marked with tokens; thus the *marked place* can be considered as a *structure*. In Fig. 3, the new *PSP* Π^w is Π_1 and $\Pi_2 | \Pi^w$. In Fig. 4(a) $\Pi_1 || \Pi_4$. Π_3 and Π_6 are on an elementary cycle containing Π_3, Π_5, and Π_6; hence Π_3 o Π_6 in Fig. 4(a).

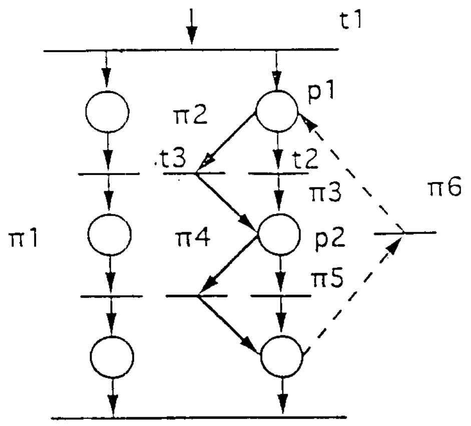


Fig. 4(a). Generate a new IG using PP rule.

	1	2	3	4	5	6
1		CN	CN	CN	CN	CN
2	CN		EX	CL SE	CL SE	CL
3	CN	EX		CL SE	CL SE	CL
4	CN	CL SL	CL SL		EX	CL
5	CN	CL SL	CL SL	EX		CL
6	CN	CL	CL	CL	CL	

CN: CoNcurrent
EX: EXclusive
SE: Sequential Earlier
SL: Sequential Later
CL: Cyclic

Fig. 4(b). An example of the matrix to record to the relationships between PSPs.

The structural relationship between any two *PSP*s is recorded in a matrix. An entry A_{ik} corresponding to row i and column k represents the relationship between Π_i and Π_k. For example, if Π_i is *concurrent* to Π_k, then $A_{ik} = A_{ki} = CN$, where CN stands for *concurrent*. Such a matrix is termed a **Temporal-Matrix** (T-Matrix) because the structural relationship between two *PSP*s resembles their temporal relationship as explained below; see also Lemma 1.

It is easy to see that if we execute a PN starting from the initial marking involving Π_1, then $\Pi_1 \rightarrow \Pi_2$ implies that Π_1 is executed earlier than Π_2; $\Pi_1 | \Pi_2$ implies that there is no need to execute Π_2 to proceed; and $\Pi_1 \| \Pi_2$ implies that both of them need to be executed to proceed. Intuitively, $\Pi_1 \leftrightarrow \Pi_2$, if they are subject to an intra-iteration precedence relationship; $\Pi_1 \| \Pi_2$, if they can proceed in parallel. $\Pi_1 | \Pi_2$ if it can return to its initial marking with only one of them being executed.

Complementing the prime start node is the prime end node defined as follows:

Definition 12: Let DEP1 = $[\Pi_1 n_k \ldots n_2 n_{pe}]$ and DEP2 = $[\Pi_2 n_{k'} \ldots n_{2'} n_{pe}]$, where DEP1 $\cap$ DEP2 = $\{n_{pe}\}$, Π_1 and Π_2 are two *PSP*s, and there are no DEPs (called bridges) from a node $(\neq n_{pe})$ DEP1 to a node $(\neq n_{pe})$ of DEP2 and vice versa. n_{pe} is called a **prime end node** n_{pe}.

A new class of nets called **Synchronized-Choice** (SC) nets is defined to be the set of nets satisfying the SC condition: the n_{ps} and n_{pe} of any pair of *PSP*s must be both transitions or both places. That is if $\Pi_1 \|(|) \Pi_2$, then both n_{ps} and n_{pe} are transitions (places). Our synthesized nets belong to this class of nets.

Definition 13: If $\exists\ \Pi$, $n_1 \in \Pi$ and $n_2 \in \Pi$, then $n_1 \leftrightarrow n_2$. If $n_1 \in \Pi_1$ and $n_2 \in \Pi_2$, $\Pi_1 \neq \Pi_2$; the *structural relationship* between n_1 and n_2 is A_{12}. □

Independent, uncoordinated action may cause a N to be unbounded and nonlive. A single generation may thus necessitate additional generations to maintain coordination and the associated searching of additional n_g and n_j as in Rules TT.4 and PP.2. This searching motivates the following two definitions.

Definition 14: A local exclusive set (LEX) of Π_i with respect to Π_k, X_{ik}, is the maximal set of all *PSP*s which are exclusive to each other and are equal to or exclusive to Π_i, but not to Π_k. That is, X_{ik} = LEX(Π_i, Π_k) = $\{\Pi_z | \Pi_z = \Pi_i$ or $\Pi_z | \Pi_i$, $\neg(\Pi_z | \Pi_k)$, $\forall \Pi_{z1}, \Pi_{z2} \in X_{ik}$, $\Pi_{z1} | \Pi_{z2}\}$. □

Definition 15: A local concurrent set (LCN) of Π_i with respect to Π_k, C_{ik}, is the maximal set of all *PSP*s which are concurrent to each other and are equal to or concurrent to Π_i, but not to Π_k, i.e., $C_{ik} = LCN(\Pi_i,\Pi_k) = \{\Pi_z | \Pi_z = \Pi_i$ or $\Pi_z \| \Pi_i, \neg (\Pi_z \| \Pi_k), \forall \Pi_{z1}, \Pi_{z2} \in C_{ik}, \Pi_{z1} \| \Pi_{z2}\}$. □

Examples of LEX and LCN appear in Figs. 5 and 6 and Rules TT.4 and PP.2 employ them respectively.

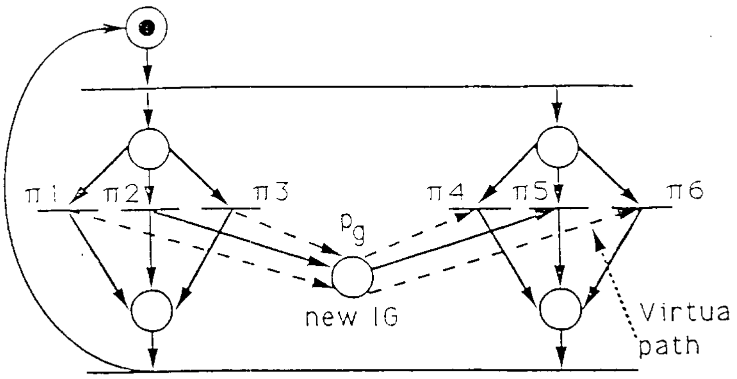
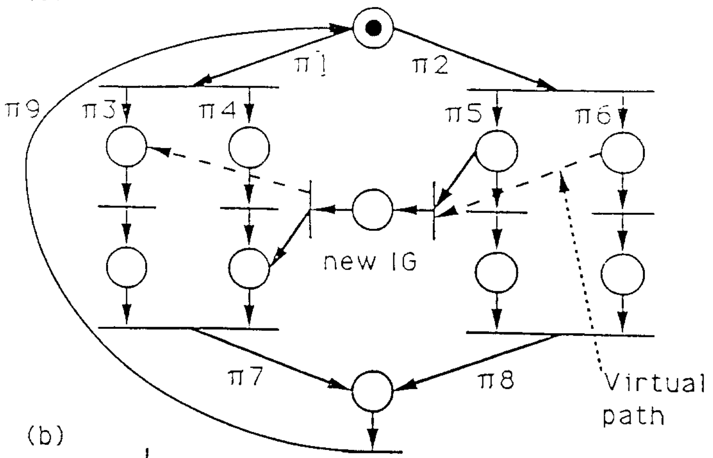


Fig. 5(a). An Example of the application of Completeness rule 1.

	1	2	3	4	5	6
1		EX	EX	CN	CN	CN
2	EX		EX	CN	CN	CN
3	EX	EX		CN	CN	CN
4	CN	CN	CN		EX	EX
5	CN	CN	CN	EX		EX
6	CN	CN	CN	EX	EX	

Fig. 5(b). Original T-Matrix (excluding the new IG).
LEX(Π2, Π5) = {Π1, Π2, Π3}
LEX (Π5, Π2) = {Π4, Π5, Π6}

(a)



(b)

	1	2	3	4	5	6	7	8	9
1		EX	SP	SP	EX	EX	SE	EX	SE
2	EX		EX	EX	SP	SP	EX	SE	SE
3	SN	EX		CN	EX	EX	SP	EX	SE
4	SN	EX	CN		EX	EX	SP	EX	SE
5	EX	SN	EX	EX		CN	EX	SP	SE
6	EX	SN	EX	EX	CN		EX	SP	SE
7	SL	SL	SN	SN	EX	EX		EX	SP
8	EX	SL	EX	EX	SN	SN	EX		SP
9	SL	SL	SL	SL	SL	SL	SN	SN	

Fig. 6(a) An example of the completeness rule 2 involving interactions between two exclusive PSP using PP rule.
LCN(Π4,Π5) = {Π3,Π4}, LCN(Π5,Π6) = {Π5,Π6}
6(b) The T-Matrix of Fig. 6(a).

III. THE SYNTHESIS RULES

We first present possible types of generations followed by the definitions of the rules.

In order for the rules to be complete, all possible generations must be considered. The type of generations depend on n_g and n_j in two factors: (1) whether they are transitions or places and (2) their structural relationship. For factor (1), there are four types: **TT, PP, TP and PT generations** defined as follows:

Definition 16: The n_g and n_j of a TT (PP, TP, and PT) generation are transition (place, transition, place) and place (place, place, and transition) respectively. □

For factor (2), n_g and n_j can be in the same or different *PSP*s. For the latter case, there are five possibilities: (1) sequential earlier (SE), (2) sequential later (SL), (3) concurrent (CN), (4) exclusive (EX), and (5) cyclic (CL). The former case is formalized as follows:

Definition 17: Let Π_g (Π_j) denotes the *PSP* which contains the generation point (joint). Pure Generation (*PG*) generates paths within a single *PSP* (Fig. 2); i.e., $\Pi_g = \Pi_j$. Interactive Generation (*IG* generates paths between two *PSP*s (Fig. 4); i.e., $\Pi_g \neq \Pi_j$. □

For the latter case, (1) and (2) are defined as:

Definition 18: If $n_g \rightarrow n_j$ prior to the generation, then it is a forward generation; otherwise it is a backward generation. □

In Fig. 2, [t1 p4 t3] is a forward generation and Π6 in Fig. 4(a) is a backward generation. A backward *TT* generation needs the addition of tokens to the *NP* to avoid the resulting N to be not live. This makes Rule TT.2.

The idea of constructing the rules is simple. An eligible generation upon N^1 whenever the *NP* gets a token. This alters the marking behavior of N^1 and the net is unbounded. A *PT* generation robs a token from N^1 and causes N^1 to be nonlive; hence changing its firing behavior. To avoid this problem, add more *generations*[24,25] or forbid such generations. This paper does not deal with *TP* and *PT* generations.

The following definitions for TT and PP rules have consiered all possible structural relationships between n_g and n_j. Some generations are forbidden; some require the addition of tokens and others require further generations.

Definition: TT Rule:

For an *NP* from $t_g \in \Pi_g$ to $t_j \in \Pi_j$ generated by the designer,

(0) TT.0 If $t_g | t_j$ or only one of them is in a cycle, which was solely generated using Rule PP.1, then signal "forbidden, delete the Π^{w}" and return.

(1) TT.1 If $\Pi_g = \Pi_j$, signal "a pure *TT generation*"; otherwise signal "an interactive *TT generation*".

(2) If $t_g \leftarrow t_j$, signal "forming a new cycle".
If, without firing t_j, there does not exist a firing sequence σ to fire t_g, then insert a token in a place of *NP*.
If $\Pi_g = \Pi_j$, return and the designer may start a new generation.

(3) TT.3 If there was no path from t_g to t_j prior to this generation or each of t_g and t_j is in a cycle which was solely generated using Rule PP.1, then

(a) TT.3.1.

Apply Rule TT.4.

(b) T.3.2.

Generate a new *TT-path* from t_g' to t_j' (on the cycles 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.

(4) TT.4

(a) TT.4.1

Generate a *TP-path* from a transition t_g of each Π_g in X_{gj} to a place p_k in the NP.

(b) TT.4.2

Generate a *virtual PSP*, a *PT-path*, from the place $\hat{n}_j$ to a transition t_j of each Π_j in X_{jg}.

Definition: PP Rule:

For an *NP* from $p_g \in \Pi_g$ to $p_j \in \Pi^J$ generated by the designer,

(0) PP.0 If $p_g \| p_j$, then signal "forbidden, delete the Π^{w}" and return.

(1) PP.1 If $\Pi_g = \Pi_j$, signal "a pure *PP generation*"; otherwise signal "an interactive *PP generation*".

(2) PP.2.

(a) PP.2.1 Generate a *TP-path* from a transition t_k of *NP* to a place p_j of each Π_j in C_{jg}.

(b) PP.2.2 Generate a *virual PSP*, a *PT-path*, from a place p_g of each Π_g in C_{gj} to the transition $\mathring{n}_g$ of *NP*.

Note some generation may require the application of more than one rule. For instance, a backward IG may need both Rules TT.2 and TT.4. 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.4, we obtain Rule PP.2. The rules are summarized in Table 1.

Table 1. Summary of the Rules

Conditions of Generations	PP Rule	TT Rule
A. cycles created		TT.2
B. no cycles created		
B.1 only one of n_g and n_j in a cycle from PP.1		TT.0
B.2 Exclusive	PP.2	TT.0
B.3 Concurrent	PP.0	
B.3.1 neither n_g nor n_j in a cycle from PP.1		TT.4
B.3.2 both n_g and n_j in a cycle from PP.1		TT.3
B.4 Sequential or Cyclic	PP.1 (PG)	TT.1 (PG)
	PP.2 (IG)	TT.4 (IG)

Section VI demonstrates the application of these rules to an automated manufacturing system. Note that during each application of a rule, more than one new *PSP* may be generated; we call such a generation a *macro generation*. Otherwise, it is a *singular generation*.

In the remainder of this paper, all nets mentioned refer to nets synthesized using the rules presented in the above.

In the sequel, we explain the rules starting with the rules that forbid generations.

Forbidden *TT* and *PP* generations come from *synchronic* and *concurrency* mismatches respectively which cause uncoordinated actions. For a *TT* generation, a single parameter, the synchronic distance between t_g and t_j, can unify all cases of the structural relationships between n_g and n_j.

Definition 19: Synchronic Mismatch: A *TT* generation causes a synchronic mismatch if $d^1_{gj} \neq d^w_{gj}$, where d^1_{gj} (d^w_{gj}) denotes the synchronic distance between t_g and t_j in N^1 (*NP*).

Definition 20: Concurrency Mismatch: A *PP* generation causes a concurrency mismatch if $(p_g \| p_j)$ and $\exists$ a $p \in P$, $p \| p_g$, and $\neg (p \| p_j)$.

There are three possible synchronic distances: 1, ∞ and the integers in between. A *TT* generation should not alter the synchronic distance between t_g and t_j. But a *TT-path* implies a synchronic distance of 1 between t_g and t_j. Thus, if the synchronic distance between g_g and t_j is one prior to the generation, the *TT* generation will not alter the reachable markings of N^1. If t_g (t_j) can fire infinitely often without firing t_j (t_g), then the *NP* will get unbounded tokens (become nonlive due to deficiency of tokens).

In terms of structural relationship, $d_{gj} = \infty$ for the following cases: (1) $t_g | t_j$, (2) t_g and/or t_j is in a cycle generated earlier using the *PP rule*, (3) $t_g \leftrightarrow t_j$ or $t_g \| t_j$, $\exists$ t_x such that $t_g | t_x$ and/or $t_j | t_x$, and (4) t_g and t_j are in two separate PNs. For cases (1) and part of (2) where only one of t_g and t_j is in a cycle, the generation is chosen to be forbidden to make Rule TT.0. The rest cases need additional generations to make Rules TT.3 and TT.4.

Rule PP.0 forbids concurrent mismatch since $\Pi_g \| \Pi_j$, they need to cooperate to complete the task. But the *NP* diverts the token away to break the coordination and induces a deadlock; hence Rule PP.0 forbids a *PP* generation between concurrent *PSP*.

Rules TT.4 and PP.2 model the messages exchange between two parties and the context switching between two processes respectively. To satisfy the *completeness requirement*[1] in communication protocols, each *PSP* in X_{gj} (X_{jg}), if executed, must send (receive) a token by firing the relevant t_g (t_j).

In the context switching, all local contexts in C_{gj} must switch to those in C_{jg}. See Figs. 5 and 6 for the application of these two rules.

The correctness of these rules is established in the sequel. We first show the correctness for a synchtesized net where each home place holds exactly one token. Let N^a denote such a net, N^b the net after adding tokens to N^a, and N^{1a} a N^1 and also a N^a. We then show that the synthesized net is **marking monotonic**; that is it remains well-behaved by adding tokens to places in the synthesized net.

Rules TT.1, TT.2 and PP.1 do not require further generations. They are simple and have been dealt with in many literatures usinig the concept of reductions and expansions.

Theorem 1: Any PN resulting from a singular application of Rules TT.1 and PP.1 respectively to a N^{1a} synthesized from a basic process is well-behaved.

Proof: See Appendix I. □

Theorem 2: Any PN resulting from a singular application of rules TT.2 to a N^{1a} synthesized from a basic process is well-behaved.

Proof: See Appendix II. □

Lemma 1: for a N^2, any pair of *PSP*s cannot be both concurrent and exclusive to each other.

Proof: See Appendix III. □

This lemma leads to the equivalence of structural and temporal relationship, which is marking or time, rather than structurally related. Two transitions (places) are temporally concurrent to each other if they can fire (have tokens) simultaneously in a safe PN. Without this lemma, p_1 and p_2 may not (may) be able to hold tokens simultaneously even if $p_1 \| (|)\ p_2$ since there may be two DEPs from a p_{ps} (t_{ps}) to p_1 and p_2 respectively.

Lemma 2: For a N^2, let $M^2_\alpha \in R^2$, where all Πs in C_{12} (C_{21}) holds exactly one token.

Proof: See Appendix IV. □

This lemma ensures that the application of Rule PP.2 will not alter the reachable markings of N^1. The following lemma also comes from the equivalence of temporal and structural relationship.

Lemma 3: Using Rule TT.4, if one t_j in X_{jg} gets a token in each of its input places except the one from $\mathring{n}_j$, then any subsequent firing sequence, if long enough, will inject a token to $\mathring{n}_j$ and fires the above t_j.

Proof: See Appdneix V. □

This lemma along with Lemma 4 ensures no tokens get accumulated in the *NP* indefinitely. Note if $t_j \leftarrow t_g$, then the *NP* is marked by Rule TT.2. t_j can fire without the token injected by firing a t_g. But subsequent firings will always lead to the firing of a t_g to restore M_0^w.

The above firing sequence, denoted as σ^w, including both t_g and t_j and $\forall$ t ϵ σ^w, t ϵ T^w, is called a *complete firing sequence* of the *NP*. Lemma 4 shows that after a σ^w, the marking of N^{1a} in N^{2a}, $M^{1/2}$ will be a reachable one in N^{1a}. The correctness of this lemma is based on the following:

Obersation I: $\forall$ t ϵ T^w, if $t \neq n_g$, $t \neq n_j$ and $t \neq \mathring{n}_g$,

(1) $|\bullet t| = 1$.

(2) $\forall$ pair of t_1 and t_2 ϵ T^w, $\neg$ $(t_1 \| t_2)$ using Rule TT.4, and $\neg$ $(t_1 | t_2)$ using Rule PP.2.

(3) Any TP (PT) generation occurs between exclusive (concurrent) *PSP*s.

(4) Any pair of *PSP*s Π_1 and Π_2 joining at a node n_k in the *NP*, then $\Pi_1 \| (|)$ Π_2, if n_k ϵ T^w (P^w).

Proof: See Appendix VI. □

The observation derives from the fact that there are only *TP* and *PT* generations beyond the first generation using Rules TT.4 and PP.2. From (2) of Observation I, any pair of *PSP*s Π_1 and Π_2 using Rule TT.4 (PP.2) are either $\Pi_1 \rightarrow \Pi_2$ or $\Pi_1 | (\|)$ Π_2. Thus, a complete firing path tnrough the *NP* is a single DEP using Rule TT.4 and includes all transitions in the *NP* using Rule PP.2.

Lemma 4: For the N^2 after applying Rule TT.4 (PP.2), after injecting a token (tokens) into the *NP* by firing only one t_g (the $\mathring{n}_g$), the subsequent firing sequence will lead to (M_α^{1a}, M_0^w) $[\sigma^w \rightarrow (M_\beta^{1a}, M_0^w)$, where both M_α^{1a} and M_β^{1a} ϵ R^{1a}.

Proof: See Appendix VII. □

Lemma 5: $\forall\ \sigma$, such that $M_0^2\,[\sigma M_0^2$, either $\sigma^w \subseteq \sigma$ or $\forall$ t ϵ σ, $\neg$ (t ϵ T^w).

Proof: See Appendix VIII. □

Theorem 3: Any PNs resulting from the singular applications of Rules TT.4 and PP.2 to a N^{1a} synthesized from a basic process are well-behaved.

Proof: See Appendix IX. □

Theorem 4: Any PN resulting from a singular application of Rule TT.3 to a N^{1a} synthesized from a basic process is well-behaved.

Proof: See Appendix X. □

Theorem 5: Any PNs resulting from the applications of the TT and PP rules to a basic process are bounded, live, reversible and marking monotonic.

Proof: See Appendix XI. □

The *bounded* property can also be proven by deriving the P-*invariants*[26] for the synthesized nets.

Theorem 6: The class of synthesized PN belongs to the class of synchronized-choice (SC) nets.

Proof: See Appendix XII. □

Note a SC may not be synthesized; this happens when the SC is not well-behaved. For instance, when the *PT*-path (TP-path) generated using Rule TT.4.2 (PP.2.2) is not a virtual one, then N^2 is a SC but with deadlocks.

IV. TEMPORAL MATRICES FOR PETRI NETS

The rules should be implemented as an interactive and visual-aid tool. The designer can view the PN model being designed on the screen and use a mouse to input paths according to the design specifications. Each time a path is generated, the system should check the new path generation against the rules. If some rule is violated, a warning should be issued to the designer to request that either this new path be deleted or else add more *PSP*s. In order to automate this process, a mechanism is needed to:

- Record the relationship between any two *PSP*s, and
- Determine the applicable rule upon a path generation. Generate additional paths and entries if necessary. If no rules are applicable, delete this path.

We use matrix to perform the first function. Instead of recording connectivity between places and transitions, the matrix in our algorithm records structural relationship between *PSP*s. Before a path is generated, the T-Matrix needs to be consulted to verify that no rule is violated. Whenever a new path is created, a *PSP* may be separated into several new *PSP*s, and relationships between some *PSP*s are changed. Therefore, the number of total *PSP*s needs to be updated, so do the matrix entries.

Example: Fig. 4(a) shows a PN model with its matrix shown in Fig. 4(b). Originally there are 5 *PSP*s in solid paths Π_1 to Π_5; Π_6 in a dashed path is to be added by IG. Π_1 is *concurrent* to all other *PSP*s. Therefore, all entries in the first column an the first row are *CN*. Π_2 and Π_3 are *exclusive* to each other, so are Π_4 and Π_5. Therefore,

$$A_{23} = A_{32} = A_{45} = A_{54} = EX.$$

Notice Π_6 returns to form cycles; therefore some entries need to be updated. Prior to Π_6 generation, Π_3 is structurally *sequentially earlier* (*SE*) to both Π_4 and Π_5. After the addition, Π_6, Π_4 and Π_5 become *SE* to Π_3, implying the formation of cycles.

We have described how the T-Matrix records the relationships. The next section presents the algorithm and the associated time complexity for updating the T-Matrix.

V. THE ALGORITHM AND ITS COMPLEXITY

The algorithm consists of two steps:

- Create new entries in the T-Matrix for each *NP* (new path).
- Determine the applicable rule; generate additional new paths if necessary and update the T-Matrix accordingly. If no rule is applicable or some rules are violated, delete the path.

For example, if a *PP-path* is generated between p_g and p_j and $p_g \| p_j$, then Rule PP.0 dictates that this path be deleted. It is easy to find the structural relationship between the new *PSP* Π^w and the *PSP*s directly involved with the generation, but not obvious as to the structural relationship between *PSP*s far away from (no direct connection with) the Π^w. In the following, we will develop techniques to enter new entries for PG and IG respectively.

A. Determination of Entries for Pure-Generation

After the PG of a Π_l from Π_g, Π_g is split into three new *PSP*s $\Pi_{g1} \rightarrow \Pi_{g2} \rightarrow \Pi_{g3}$. The entires related to Π_g should be deleted. The values for the new entries are:

$$A'_{ik} = A_{gk}, \qquad A'_{ki} = A_{kg}$$

where Π_k is any *PSP* other than Π_g, Π_l, Π_{g1}, Π_{g2} and Π_{g3} and Π_i is one of the newly created *PSP*s. Note Π_{g1} and/or Π_{g2} may be empty. The rest new entries are the relationships among the newly created *PSP*s:

$$A'_{g2g1} = A'_{g3g1} = A'_{g3g2} = SL,$$

$$A'_{g1g2} = A'_{g2g3} = A'_{g1g3} = SE$$

and

$$A'_{lg2} = CN \qquad \text{for } PG \text{ using } TT \text{ rule}$$

$$A'_{lg2} = EX \qquad \text{for } PG \text{ using } PP \text{ rule}$$

Therefore, no modifications for existing entries are necessary for PG. There are four new *PSP*s. Each new entry takes constant time to enter, and the corresponding time complexity is $O(n)$, where n is the number of *PSP*s before the generation. We add a PG in Fig. 7 using the *PP rule* with the T-Matrix and PN shown in Fig. 8. There are four new entries: Π_3, Π_4, Π_5 and Π_6 respectively.

B. Determination of Entries for Interaction-Generation

In the following, we consider only the case where $\Pi_g | \Pi_j$ or $\Pi_g \| \Pi_j$; the cases where $\Pi_g \rightarrow \Pi_j$ or $\Pi_j \rightarrow \Pi_g$ can be treated similarly.

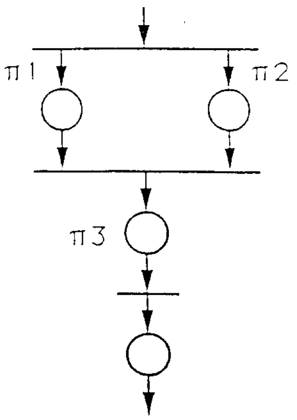


Fig. 7(a). Original Petri Nets with 3 PSPs.

	1	2	3
1		CN	SP
2	CN		SP
3	SN	SN	

SP: Sequential Previous

SN: Sequential Next

Fig. 7(b). Corresponding T-Matrix of Fig. 7(a) prior to the pure-generation.

For an IG of Π_l from Π_g to Π_j, Π_g (Π_j) is split into $\Pi_{g1} \rightarrow \Pi_{g2}$ ($\Pi_{j1} \rightarrow \Pi_{j2}$). Note that $\Pi_{g1} \rightarrow \Pi_l \rightarrow \Pi_{j2}$. The structural relationship between the new *PSP* Π_l and existing *PSP*s must be determined. Also some structural relationships among existing *PSP*s are changed. For instance, independent of the structural relationship between Π_g and Π_j prior to the generation, after the generation, $\Pi_{g1} \rightarrow \Pi_l \rightarrow \Pi_{j2}$. It is easy to determine the structural relationships among *PSP*s directly involved in the generation, but rather difficult to determine between *PSP*s far away from the generation and *PSP*s directly involved in the generation.

The new matrix (A') entries are determined by the following lemma.

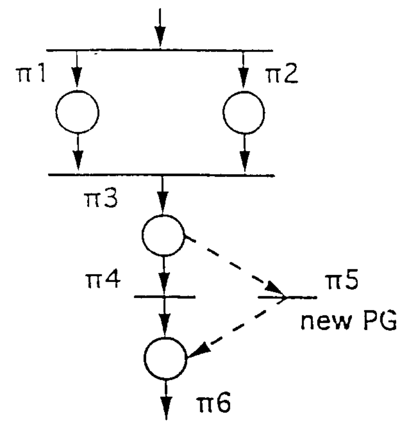


Fig. 8(a). After adding a pure-generation, there are 6 PSPs in the Petri Net.

	1	2	3	4	5	6
1		CN	SP	SE	SE	SE
2	CN		SP	SE	SE	SE
3	SN	SN		SP	SP	SE
4	SL	SL	SN		EX	SP
5	SL	SL	SN	EX		SP
6	SL	SL	SL	SN	SN	

SP: Sequential Previous

SN: Sequential Next

Fig. 8(b). Updated T-Matrix.

Lemma 6: For an IG of Π_l from Π_g to Π_j, then update A to A' as follows:

(1) $A'_{ik}=SE$ *and* $A'_{ki}=SL$, where $ik=g_1g_2$, j_1j_2, g_1l, lj_2, lq, g_1j_2, rl, rj_2, and rq.

(2) $A'_{mz}=A_{mz}$ and $A'_{zm}=A_{zm}$, where $mz \neq rq$, qr $(m,\ z\epsilon\{r,q,u,v\})$, and

(3) $A'_{tz}=A_{sz}$ and $A'_{zs}=A_{zt}$, where $st=gg_1$, gg_2, jj_1, jj_2, $z\epsilon\{r,q,u,v\}$, and $tz \neq j_2r$, g_1q.

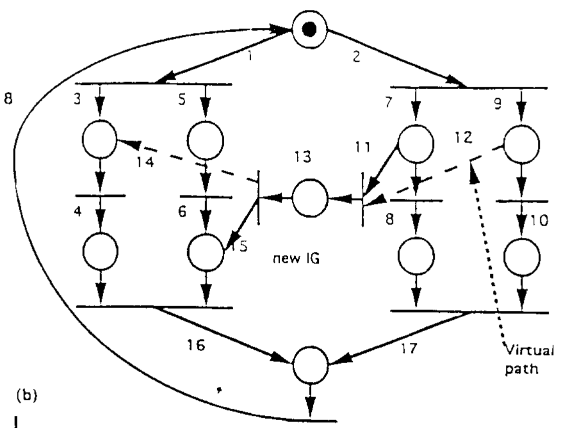
where Π_r, Π_q, Π_u, and Π_v be old *PSP*s not belonging to C_{gj} or X_{gj} such that $\Pi_r \rightarrow \Pi_g$, $\Pi_j \rightarrow \Pi_q$, $\neg\ (\Pi_u \rightarrow \Pi_g)$ (Π_u not *sequentially earlier* to Π_g), $\neg\ (\Pi_u \rightarrow \Pi_j)$, ($\Pi_u$ not *sequentially earlier* to Π_g), $\neg\ (\Pi_u \rightarrow \Pi_g)$, and $\neg\ (\Pi_v \rightarrow \Pi_j)$.

Proof: See Appendix XIII.

The corresponding time complexity is $O(n^2)$ mainly incurred by the operation $A'_{rq} = SE$. To complete the step of a single application of a *TT* or *PP rule*, additional *TP*- or *PT-paths* may have to be generated from/to the *NP*s. The matrix update after each such generation can be performed in a similar fashion to that in Lemma 6 with the same time complexity. Since there are $O(n)$ such *TP*- or *PT-paths*, the total time complexity of completing one single application of Rule TT.4 or PP.2 is $O(n^3)$.

Fig. 9 shows that the PN after the IG and the updated T-Matrix (refer to Fig. 6). Any alphabetic number in Fig. 9(a) stands for a *PSP*. Π_3 in Fig. 6

(a)



(b)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1		EX	SP	SE	SP	SE	EX	EX	EX	EX	EX	EX	EX	EX	EX	EX	EX	SE
2	EX		EX	SE	EX	SE	SP	SE	SP	SE	SE	SE	SE	SE	SE	SE	SE	SE
3	SN	EX		SP	CN	CN	EX	EX	EX	EX	EX	EX	EX	EX	EX	SE	EX	SE
4	SL	SL	SN		CN	CN	SL	EX	SL	EX	SL	SL	SN	SN	CN	SP	EX	SE
5	SN	EX	CN	CN		SP	EX	EX	EX	EX	EX	EX	EX	EX	EX	SL	EX	SE
6	SL	SL	CN	CN	SN		SL	EX	SL	EX	SL	SL	SL	CN	SN	SP	EX	SE
7	EX	SN	EX	SE	EX	SE		SP	CN	CN	SP	CN	SE	SE	SE	SE	SE	SE
8	EX	SL	EX	EX	EX	EX	SN		CN	CN	EX	CN	EX	EX	EX	EX	SP	SE
9	EX	SN	EX	SE	EX	SL	CN	CN		SP	CN	SP	SE	SE	SE	SE	SE	SE
10	EX	SL	EX	EX	EX	EX	CN	CN	SN		EX	EX	EX	EX	EX	EX	SP	SE
11	EX	SL	EX	SE	EX	SE	SN	EX	CN	EX		CN	SP	SE	SE	SE	EX	SE
12	EX	SL	EX	SE	EX	SE	SL	EX	SN	EX	CN		SP	SE	SE	SE	EX	SE
13	EX	SL	EX	SP	EX	CN	SL	EX	SL	EX	SN	SN		SP	SP	SE	EX	SE
14	EX	SL	EX	CN	EX	SP	SL	EX	SL	EX	SL	SL	SN		CN	SE	EX	SE
15	EX	SL	EX	CN	EX	SP	SL	EX	SL	EX	SL	SL	SN	CN		SE	EX	SE
16	SL	SL	SL	SN	SL	SN	SL	EX	SL	EX	SL	SL	SL	SL	SL		EX	SP
17	EX	SL	EX	EX	EX	EX	SL	SN	SL	SN	EX	EX	EX	EX	EX	EX		SP
18	SL	SL	SL	SL	SL	SL	SL	SL	SL	SL	SL	SL	SL	SL	SL	SN	SN	

Fig. 9. (a) After the Interaction-generation (IG), there are 18 PSPs in Petri nets. Each number stands for a PSP.
(b) The new T-Matrix of Fig. 9(a).

has been separated into two *PSP*s, Π_3 and Π_4 respectively. Note this PN does not belong to the class of assymetric-choice nets. But note that every pair of exclusive (concurrent) *PSP*s have places (transitions) as their n_{ps} and n_{pe}; i.e., the PN belongs to the class of synchronized-choice nets. The synthesized PN in Fig. 5(a) also belongs to the class of synchronized-choice nets.

C. Complexity of the Algorithm

The complexity for the algorithm consists of the following components: The complexity for

(1) determining which rule is applicable and checking the rule violation, and

(2) updating matrix entries,

For factor 1, it takes $O(1)$ time to consult the T-Matrix to find A_{gj}. For factor 2, the complexity to update entries has been shown to be $O(n)$ for PG and $O(n^2)$ for each *PSP* generation during an IG.

Thus each *PSP* generation takes $O(n^2)$ time complexity. Let n_i be the number of *PSP*s prior to the ith generation. The time complexity for the ith generation is $O(n_i^2)$. After a PG, one *PSP* is deleted, while at most four new *PSP*s are created. Thus after the PG, the number of *PSP*s increases at most from n_i to n_i+3.

Four an IG of an *NP* between two *PSP*, two *PSP*s are deleted and at most five new *PSP*s are created. Hence after the IG of a single *PSP*, there are at most n_i+3 *PSP*s. The time complexity for the next generation is bounded by $O([n_i+3]^2)$. Let K be the total number of *PSP*s at the end of a design. The total time complexity for the design is

$$\sum_{n_i=1}^{n_i=K-4} (n_i+3)^2 \leq \sum_{k=0}^{K} k^2 = O(K^3)$$

Thus the total time complexity for the whole design process is $O(K^3)$.

VI. X-WINDOW IMPLEMENTATION

We have implemented the above algorithm by incorporating it into a Multi-Function Petri Net Graphics (MFPNG) tool based on X-Window (Motif version). This software *package*[22,53] is a user friendly CAD tool for designing, verifying, simulating, querying (including cycle time, invariants, incidence matrix, inputs/outputs of nodes, ... etc), reducing and synthesizing PN. The structure of this tool is shown in Fig. 10.

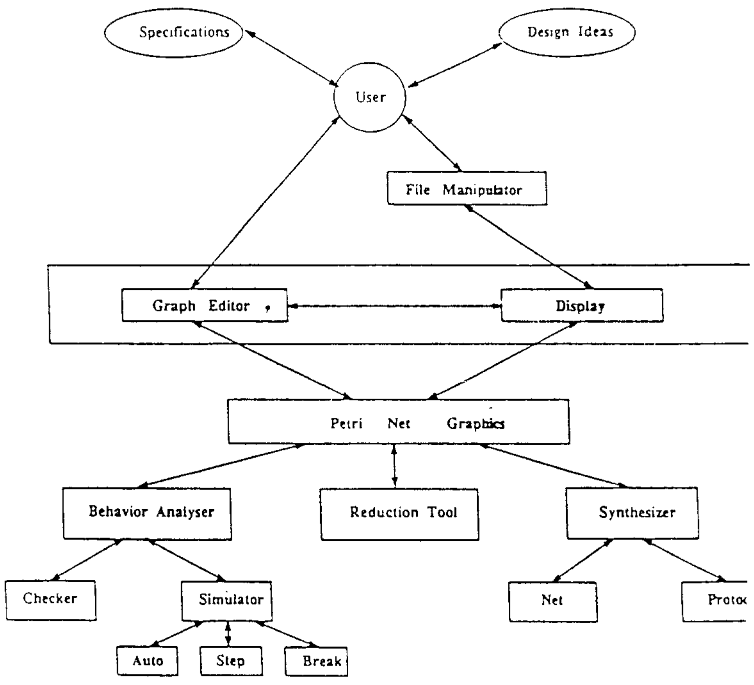


Fig. 10. Structure of the CAD tool for designing, analysis, and synthesis of protocols.

This tool can draw, erase, copy, move, pan, and zoom in/out objects such as places, transitions, states, rectangular and elliptical objects (filled and nonfilled) and texts (with a variety of fonts). The user can construct a PN either using the "DRAW" button to draw transitions, places and arcs, or using the "FILE" button to input a PN file. Buttons "Move", "Pan", and "Zoom" allow large PNs to be drawn. Clicking the "Pan" button centers the graphics at a chosen location. In addition, the vertical (horizontal) scrollbar allows the graphics to be moved up and down (left and right). After the PN is drawn or displayed on the screen, graphical interconnections among places and transitions are translated into internal representations for further manipulations.

A user can initiate the synthesis by clicking the "Petri" button of the "Synthesis" menu. Afterwards, no further clicking is necessary for further generations except for clicking the "T-Matrix" button to display the T-Matrix in the bottom window (called text_w) after a new generation. An example of this process to the synthesis of an automated manufacturing system (fig. 11) is shown in Fig. 12. Shown in the bottom is a message window text_w. After each generation step, it displays the kind of generation and signals any rule violations or whether tokens must be added to places in the *NP*s.

A new window is popped up upon an interaction generation, inside which displays an arrow linking two sets of nodes inviting the designer to pick one node in the left-hand (right-hand) set as n_g (n_j). Each time after a node is picked, the tool will call a filtering procedure to eliminate nodes in the set that are sequential to the node just picked and redisplay the window. This process continues until both sides are empty and no more *NP* needs to be generated for this specific *IG*.

In the displayed T-Matrix, each structural relationship is expressed by a single letter (e.g. *E*) different from the two letters (*SE*) presented earlier. The correspondence is shown in Table 2. In addition, A_{ik} = $'Y'$ if A_{ik} = $'X'$ and Π_i is in a cycle which has a place with more than one input transition; A_{ik} = $'Z'$ if A_{ik} = $'U'$ and Π_i is in a cycle which has a place with more than one input transition. We now apply the tool to the synthesis of an automated manufacturing system (Fig. 11).

Description of a Manufacturing System

This automated manufacturing system consists of the following major components (Table 3): two entries, two exists, five machines, two robots, and two Automatic Guided Vehicles (AGV), and related conveyors. It can produce two types of products. An unlimited source of raw materials is assumed. Once machines, robots, or AGVs start any operation, they cannot be interrupted until the work is complete. We now build up a well-behaved PN model.

Modeling And Synthesis Process

First, synthesize the production protion from Entry 1 to Exit 1, followed by that from Entry 2 to Exit 2. For the first portion, pick a basic process by identifying a sequence of operations of loading by R1, machining by M1, unloading via AGV1, loading by R2, and machining by M3 modeled as shown by the solid cycle

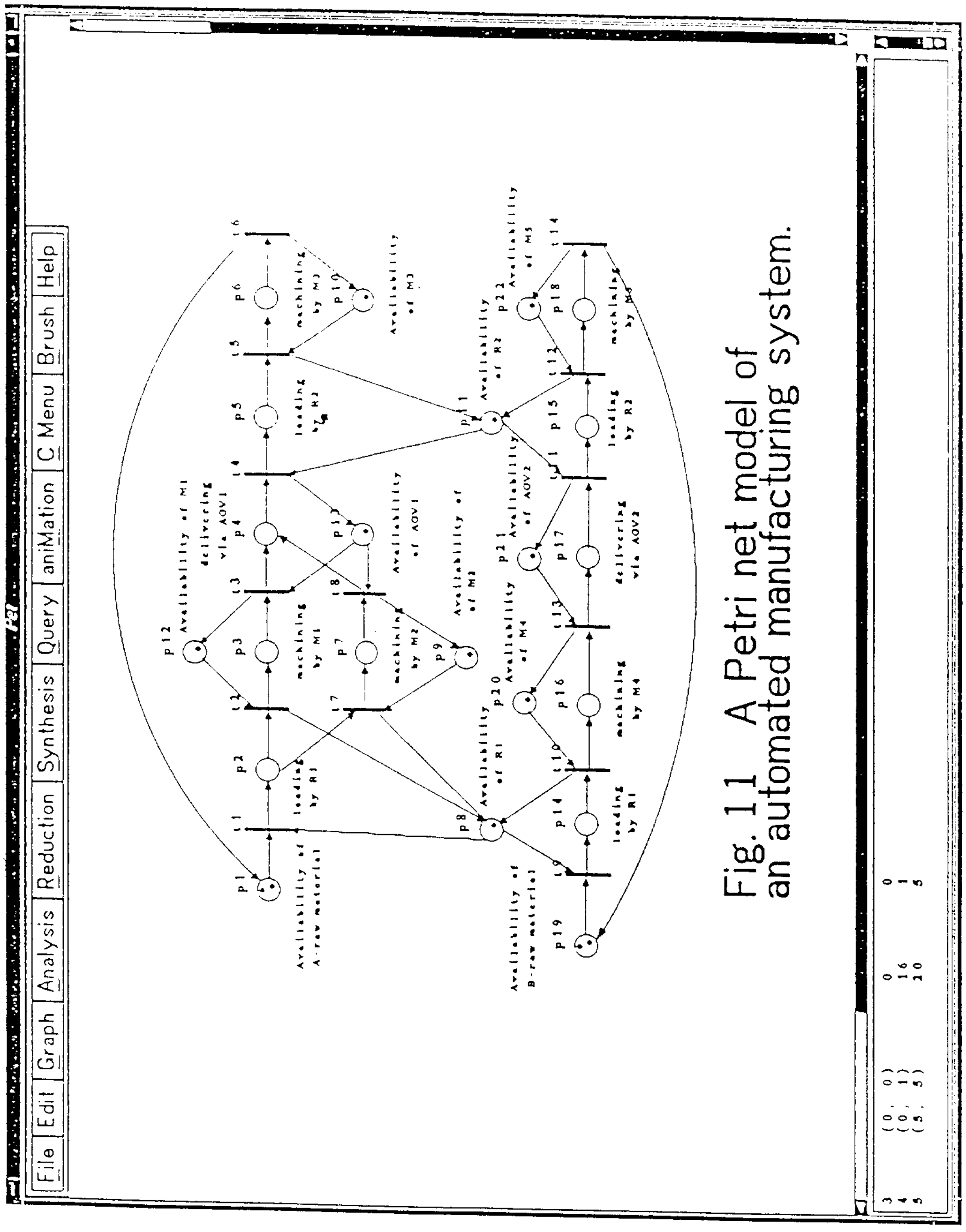


Fig. 11. A Petri net model of an automated manufacturing system.

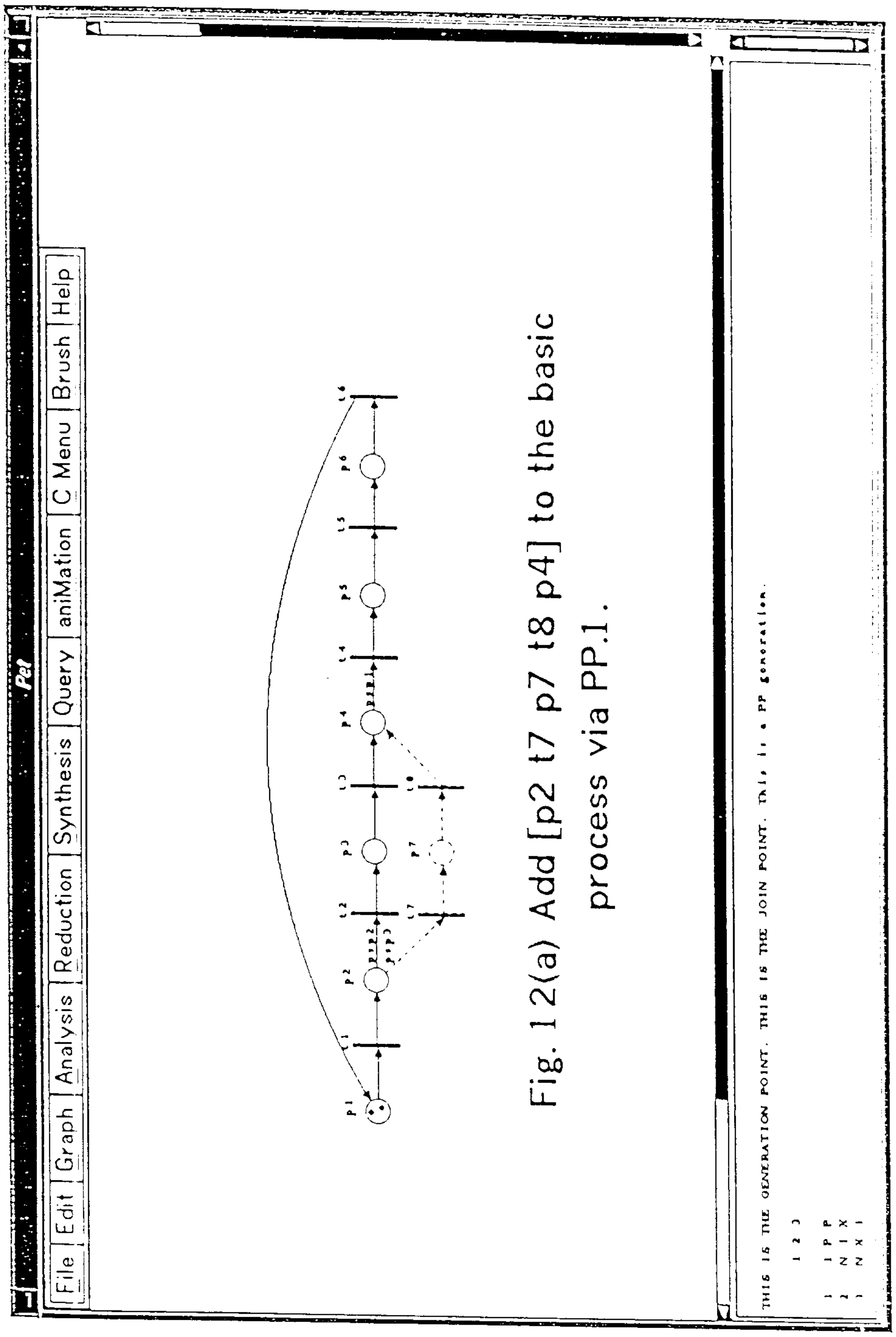
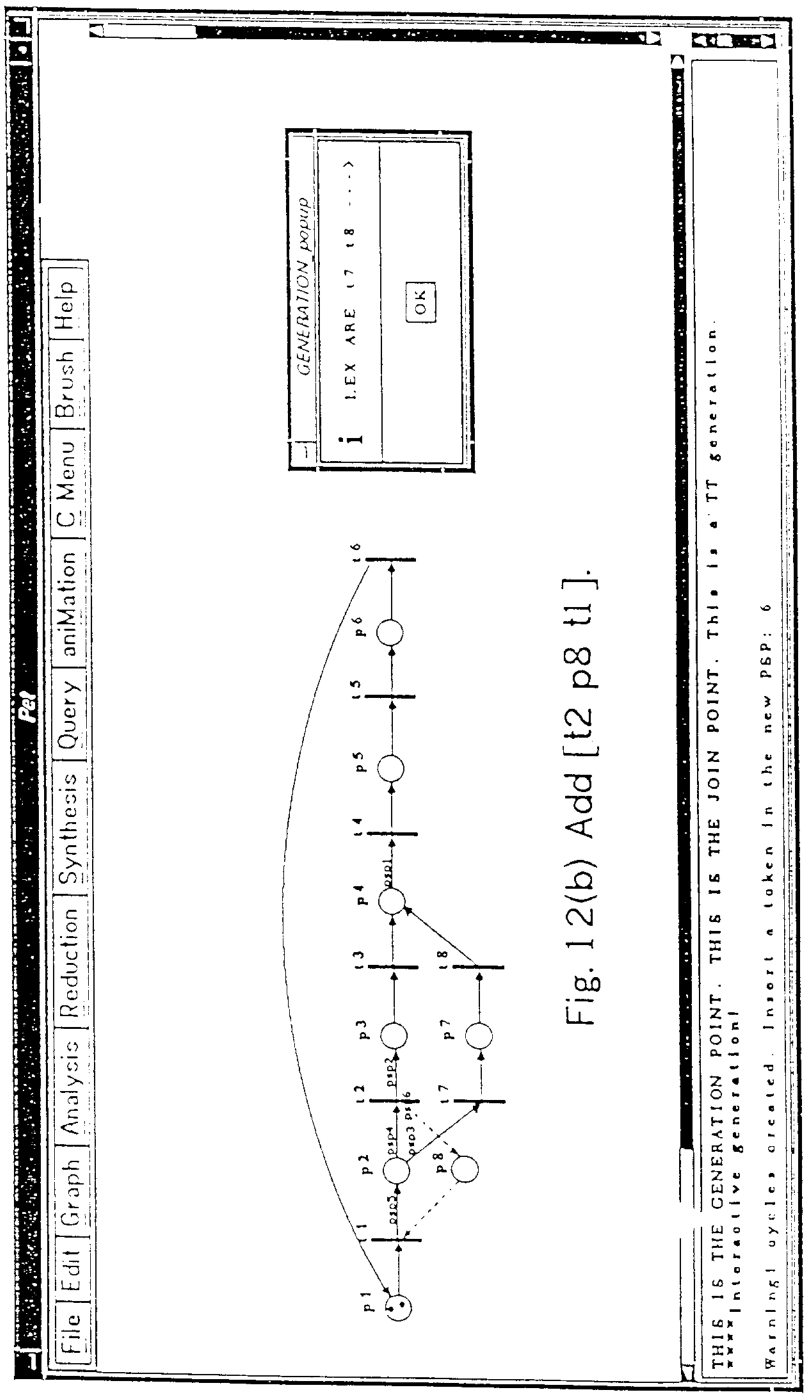


Fig. 12(a) Add [p2 t7 p7 t8 p4] to the basic process via PP.1.

THIS IS THE GENERATION POINT. THIS IS THE JOIN POINT. This is a PP generation.

```
   1 2 3
1  1 P P
2  N 1 X
3  N X 1
```

Fig. 12(a). Add [p2 t7 p7 t8 p4] to the basic process via PP.1.



GENERATION_popup

1.EX ARE (t7 t8 ...)

OK

THIS IS THE GENERATION POINT. THIS IS THE JOIN POINT. This is a TT generation.
**** Interactive generation!
Warning! cycles created. Insert a token in the new PSP: 6

Fig. 12(b). Add [t2 p8 t1].

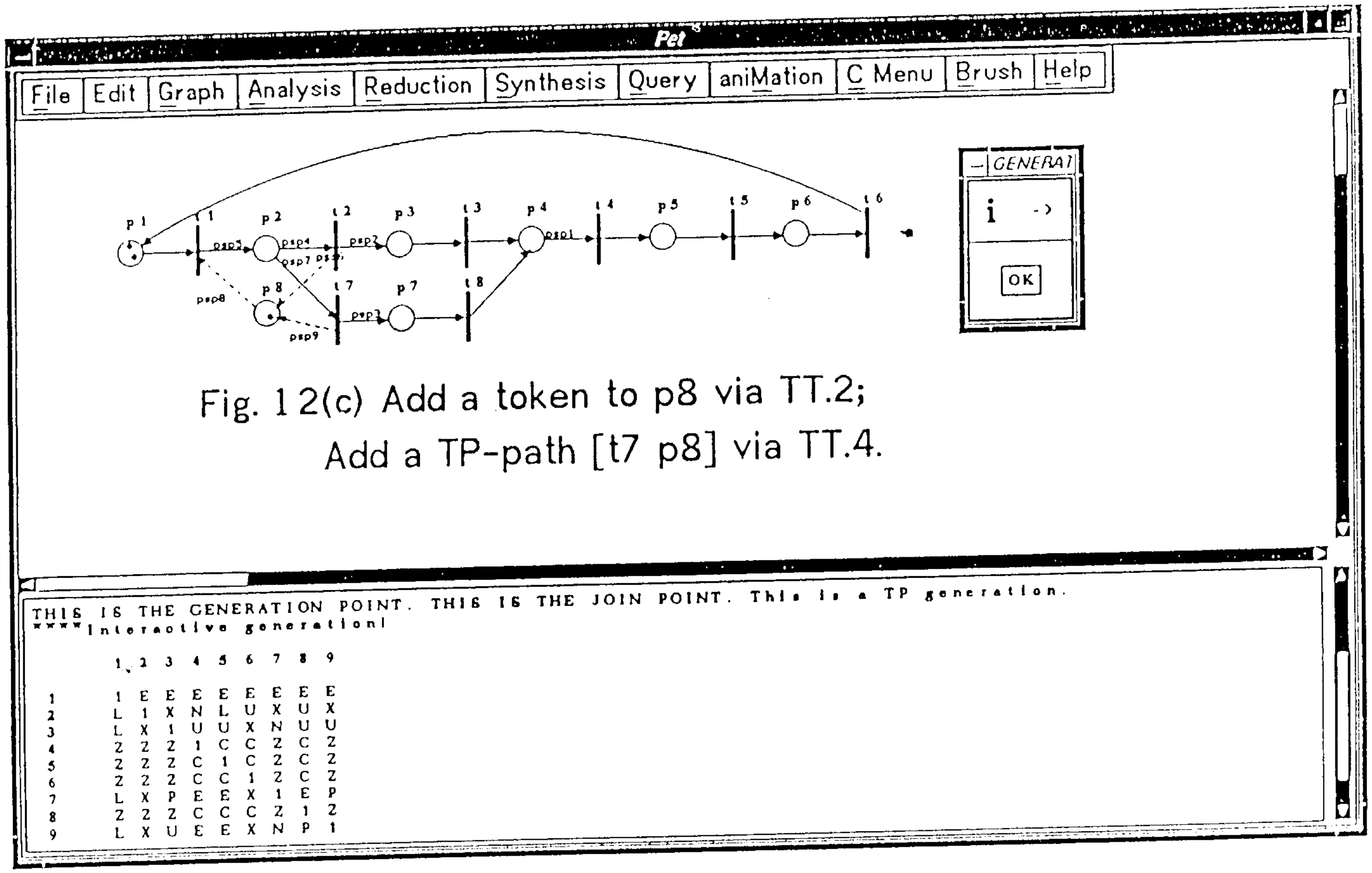


Fig. 12(c) Add a token to p8 via TT.2;
Add a TP-path [t7 p8] via TT.4.

```
THIS IS THE GENERATION POINT. THIS IS THE JOIN POINT. This is a TP generation.
****Interactive generation!

        1  2  3  4  5  6  7  8  9

1       1  E  E  E  E  E  E  E  E
2       L  1  X  N  L  U  X  U  X
3       L  X  1  U  U  X  N  U  U
4       Z  Z  Z  1  C  C  Z  C  Z
5       Z  Z  Z  C  1  C  Z  C  Z
6       Z  Z  Z  C  C  1  Z  C  Z
7       L  X  P  E  E  X  1  E  P
8       Z  Z  Z  C  C  C  Z  1  Z
9       L  X  U  E  E  X  N  P  1
```

Fig. 12(c). Add a token t p8 via TT.2; Add a TP-path [t7 p8] via TT.4.

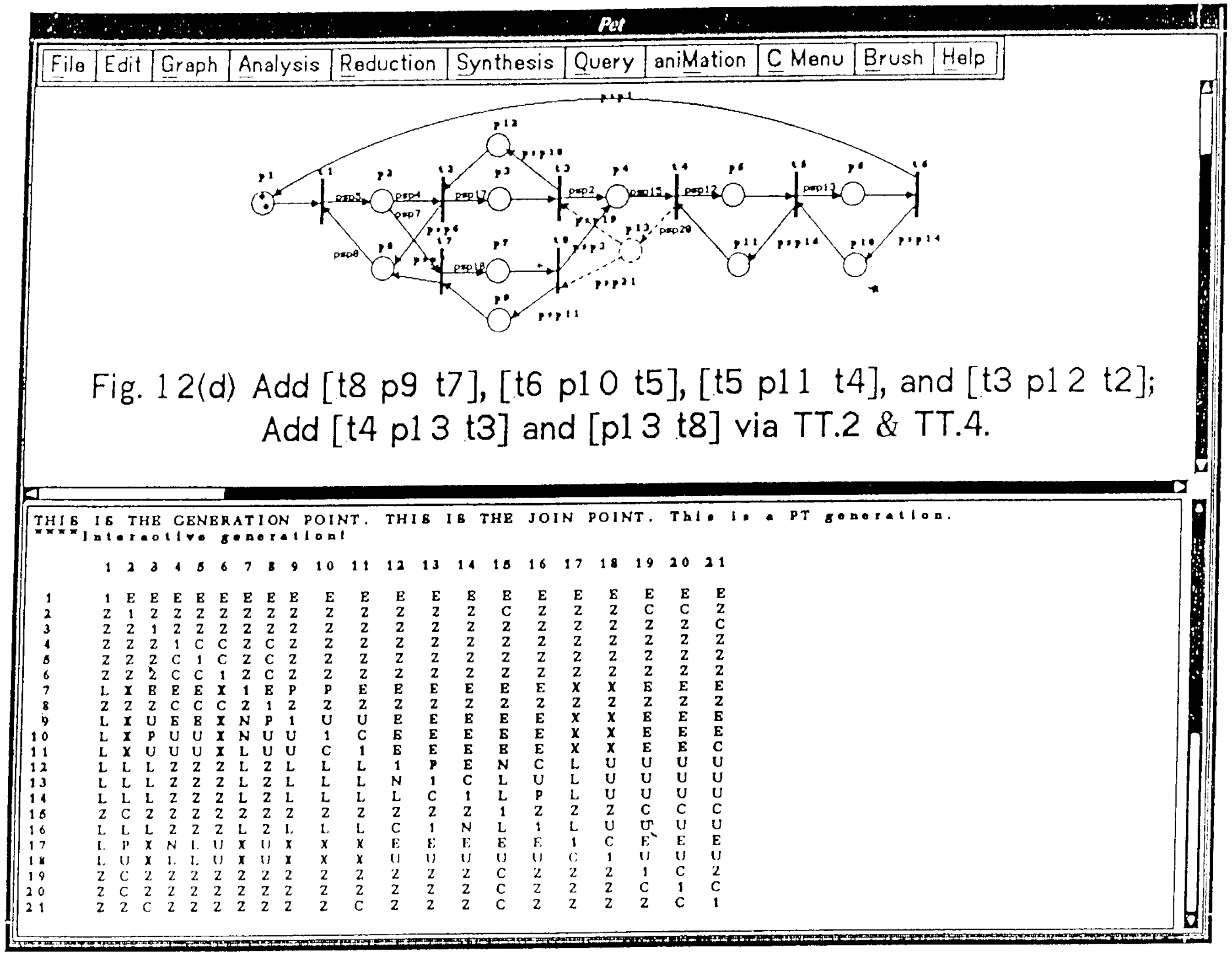


Fig. 12(d) Add [t8 p9 t7], [t6 p10 t5], [t5 p11 t4], and [t3 p12 t2];
Add [t4 p13 t3] and [p13 t8] via TT.2 & TT.4.

```
THIS IS THE GENERATION POINT. THIS IS THE JOIN POINT. This is a PT generation.
****Interactive generation!

        1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21

  1     1  E  E  E  E  E  E  E  E  E  E  E  E  E  E  E  E  E  E  E  E
  2     Z  1  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  C  Z  Z  Z  C  C  Z
  3     Z  Z  1  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  C
  4     Z  Z  Z  1  C  C  Z  C  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z
  5     Z  Z  Z  C  1  C  Z  C  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z
  6     Z  Z  Z  C  C  1  Z  C  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z
  7     L  X  E  E  E  X  1  E  P  P  E  E  E  E  E  E  X  X  E  E  E
  8     Z  Z  Z  C  C  C  Z  1  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z
  9     L  X  U  E  E  X  N  P  1  U  U  E  E  E  E  E  X  X  E  E  E
 10     L  X  P  U  U  X  N  U  U  1  C  E  E  E  E  E  X  X  E  E  E
 11     L  X  U  U  U  X  L  U  U  C  1  E  E  E  E  E  X  X  E  E  C
 12     L  L  L  Z  Z  Z  L  Z  L  L  L  1  P  E  N  C  L  U  U  U  U
 13     L  L  L  Z  Z  Z  L  Z  L  L  L  N  1  C  L  U  L  U  U  U  U
 14     L  L  L  Z  Z  Z  L  Z  L  L  L  L  C  1  L  P  L  U  U  U  U
 15     Z  C  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  1  Z  Z  Z  C  C  C
 16     L  L  L  Z  Z  Z  L  Z  L  L  L  C  1  N  L  1  L  U  U  U  U
 17     L  P  X  N  L  U  X  U  X  X  X  E  E  E  E  E  1  C  E  E  E
 18     L  U  X  L  L  U  X  U  X  X  X  U  U  U  U  U  C  1  U  U  U
 19     Z  C  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  C  Z  Z  Z  1  C  Z
 20     Z  C  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  C  Z  Z  Z  C  1  C
 21     Z  Z  C  Z  Z  Z  Z  Z  Z  Z  C  Z  Z  Z  C  Z  Z  Z  Z  C  1
```

Fig. 12(d). Add [t8 p9 t7], [t6 p10 t5], [t5 p11 t4], and [t3 p12 t2];
Add [t4 p13 t3] and [p13 t8] via TT.2 & TT.4.

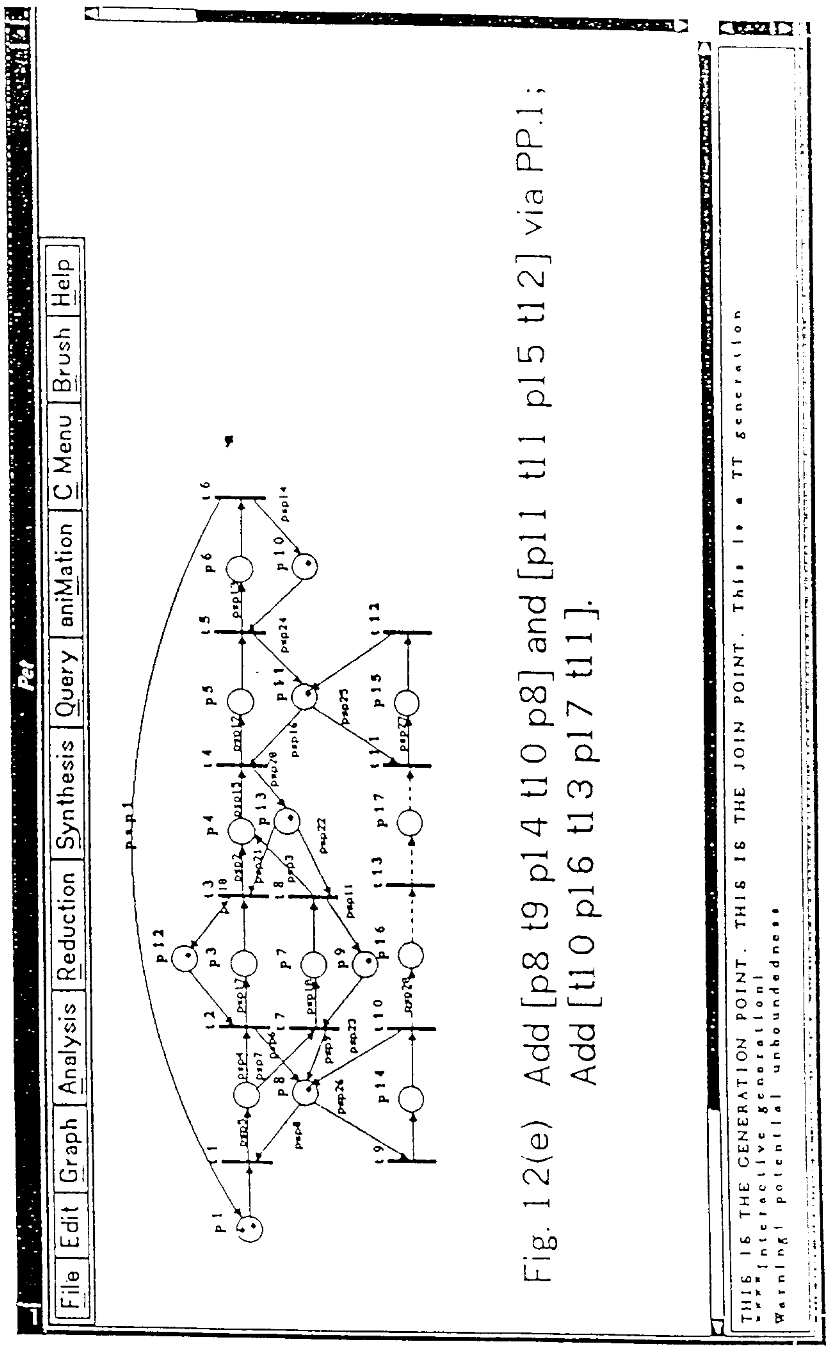


Fig. 12(e) Add [p8 t9 p14 t10 p8] and [p11 t11 p15 t12] via PP.1;
Add [t10 p16 t13 p17 t11].

```
THIS IS THE GENERATION POINT. THIS IS THE JOIN POINT. This is a TT generation
***Interactive generation!
Warning! potential unboundedness
```

Fig. 12(e). Add [p8 t9 p14 t10 p8] and [p11 t11 p15 t12] via PP.1;
Add [t10 p16 t13 p17 t11].

[pl tl p2 t2 p3 t3 p4 t4 p5 t5 p6 t6 pl] in Fig. 12(a). In this basic process, pl models the availability of A-raw materials.

Upon this basic process, generate a new *PSP* [*p*2 *t*7 *p*7 *t*8 *p*4] as the dashed line in Fig. 12(b) using Rule PP.1 since n_g = p2 and n_j=*p*4; both are from the same basic process. The new *PSP* models another alternative to produce A-paths by Machine 2. Now there are three unique *PSP*s, i.e., Π_1 (or pspl) = [*p*4 *t*4 *p*5 *t*5 *p*6 *t*6 *p*l], Π_2 (or psp2) = [*p*2 *t*2 *p*3 *t*3 *p*4], and Π_3 (or psp3) = [*p*2 *t*7 *p*7 *t*8 *p*4] . The tool also displays the T-matrix after the *PP* generation the bottom window. It is easy to see that $\Pi_2 | \Pi_3$ and Π_1 is sequential to both Π_2 and Π_3.

Now since the operation in p3 requires Robot 1, generate a backward *TT-path* from t2 (in Π_2) to tl (in Π_1), i.e., [t2 p8 tl]. And add a token in place *p*8, standing for Robot 1, using Rule TT.2. Furthermore, since the local exclusive set LEX(Π_2, Π_1) = {Π_2, Π_3}, generate a *TP-path* [t8 p8] via Rule TT.4.1 from a transition in Π_3 to p8. Since LEX(Π_1, Π_2) = {Π_1}, nothing more needs to be added via Rule TT.4.2.

Note there are two transitions t7 and t8 in Π_3, we can pick either one as the t_g of the new *TP-path*. The tool conveys this information to the designer by popping up a new window, inside which displays an arrow linking two sets of transitions: {t7, t8} and — (an empty set). Picking t7, resulting in the dotted lines in Fig 12(c). The tool then pops up another window linking two empty sets signaling the completion of this step of synthesis. The set of home places now is H = {pl, p8}.

Machines 1, 2, 3, and R2 are required to start their corresponding operations. Thus using Rules TT.2 and TT.4, generate the new paths: [t3 p12 t2], [t8 p9 t7], [t5 pl1 t4], and [t6 p13 t6] (Fig. 12(d)) where p12, p9, pl1 and pl0 model the availability of Ml, M2, R2 and M3 respectively.

Consider the AGV1, first generate a *TT-path* between t4 (in Π = [p4 t4]) and t3 (in Π' = [t3 p4]), i.e., [t4 p13 t3]. Then add a token to p13 using Rule TT.2. Since LEX([p4 t4], [t3 p4]) = ϕ, add nothing according Rule TT.4. LEX([t2 p3], [p3 t3]) can be any one of {[t7 p3]}, {[t6 p6 t7]} and {[pl t6]}. Using either {[t7 p3]} or {[t6 p6 t7]}, add the *virtual PT-path* from p12 to t7 as the dotted line in Fig. 12(d). This implies that once Machine 2 completes its processing, AGV1 delivers the part. Theoretically, one may use {[pl t6]} and then a *PT-path* [p12 t6] would be added. H = {pl, p8-p13}.

Having synthesized the partial system, i.e., the production portion from Entry 1 to Exit 1, now synthesize the rest portion, from Entry 2 to Exit 2 to involve Machines 4 and 5, AGV2 and Robots 1 and 2. For [*t*7 *p*8 *t*l] and [*t*5 *p*12 *t*4], applying Rule PP.1 leads to the two cycles [*p*8 *t*9 *p*14 *t*10 *p*8] and [*p*12 *t*11 *p*15 *t*12 *p*12]. The meanings of the added places are shown in Fig. 12(e).

Consider two new *PSP*s: Π_4 = [*p*8 *t*9 *p*14 *t*10 *p*8] and Π_5 = [*p*12 *t*11 *p*12 *t*12 *p*12] and choose *t*10 from Π_4 and *t*11 from Π_5 to generate a new *PSP*, psp=[*t*10 *p*16 *t*13 *p*17 *t*11]. Note each of the transitions *t*10 and *t*11 is in a cycle which were generated using Rule PP.1. Hence, now apply Rule TT.3 to pick up *t*9 from Π_4 and *t*12 from Π_5 and generate psp' = [*t*12 *p*18 *t*14 *p*19 *t*9]. It is easily verified that psp, psp', Π_4 and Π_5 constitute a cycle. Furthermore, insert the tokens in *p*19 to represent the availability of raw material from Entry 1 to fulfill Rule TT.2. H={*p*1, *p*8-*p*13, *p*19}.

Now apply Rule TT.2 to obtain the paths: [t13 p20 t10], [t11 p21 t13], and [t14 p22 t12], which model the availability of Machines 4, 5, and AGV2. H={*p*1, *p*8-*p*13, *p*19-*p*22}. This completes the modeling process and the final net is depicted in Fig. 11 as a bounded, live, and reversible net if $\forall p \in$ H, M_0(p)>0 and $\forall p \in$ PN-H, M_0(p)=0; i.e., the places except those in H in the net have no tokens.

VII. CONCLUSION

The tool based on the synthesis rules and the algorithm helps designers to construct large PNs interactively and to synthesize an automated manufacturing system in a user-friendly fashion. We have also implemented a reduction algorithm based on the rules; the code is very simple containing less than 100 lines.

None of the existing tools integrate drawing, file manipulation, analysis, simulation, animation, reduction, synthesis and property query in one software package. Further, because PNs model discrete-event systems, the tool finds applications in communication protocols, flexible manufacturing systems, (extended) finite state machines, expert systems, interactive paralel *debuggers*[22], digital signal processing (*DSP*[22,54-55]), ... etc.

We have enhanced the tool to include models not only PNs but also state diagrams and data flow graphs (DFGs) with few code changes. Thus a designer can choose the model that he is familiar with. For instance, DSP professionals do not know PNs well. They can, however, draw DFGs and obtain iteration bounds, critical loops, rate-optimal scheduling and others by justing clocking a *button*[22,54-55].

The fact that some generations are prohibited however, limits the classes for the synthesized nets. Recently we have discovered new rules to free forbidden generations. For instance, Rule TT.0 forbids a *TT-path* generation (e.g., *PSP* [t3 p4 t4] in Fig. 13(a)) between exclusive transitions. The new rule allows it, but needs additional generations (*PSP* [t4 p5 t3]) to synchronize the two exclusive transitions (t3 and t4).

Also, adding new paths allows *TP* and *PT* generations. A new *PT* generation ([p4 t3]) must accompany each *TP* generation ([t1 p2]) in Fig. 14) and vice versa. And each new *PT* generation must synchronize with existing *PT-paths* having the same n_g. These rules deal with ordinary PNs having unit weights between places and transitions. We have extended these rules to synchesize *GPN*s with multiple *weights*[21].

As a result, these new rules can synthesize more classes of nets. Future work should enhance the algorithm and incorporate these new rules into the tool.

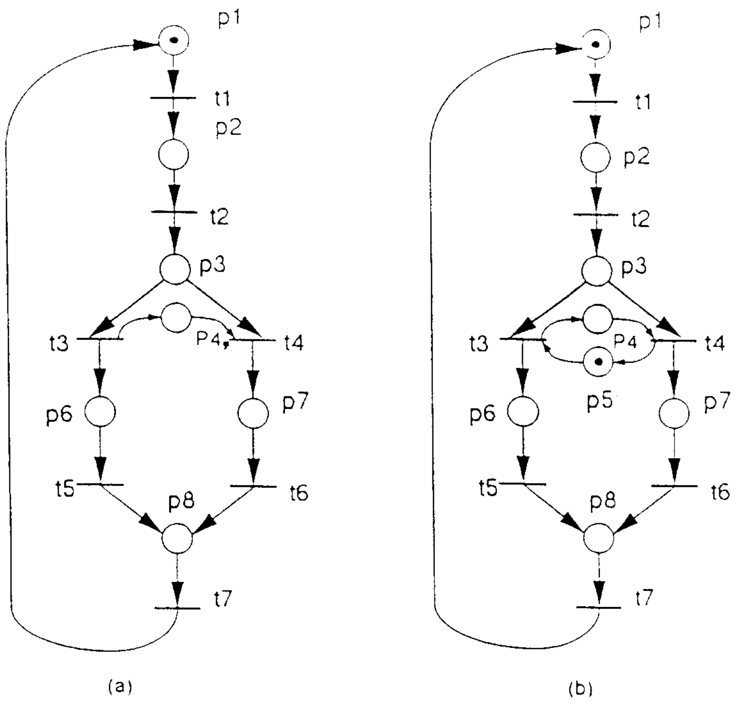


Fig. 13. An example of Rule TT.0. (a) [t3 p4 t4] is generated. The net deadlocks or is unbounded.
(b) The addition of [t4 p5 t3] renders the net live and bounded.

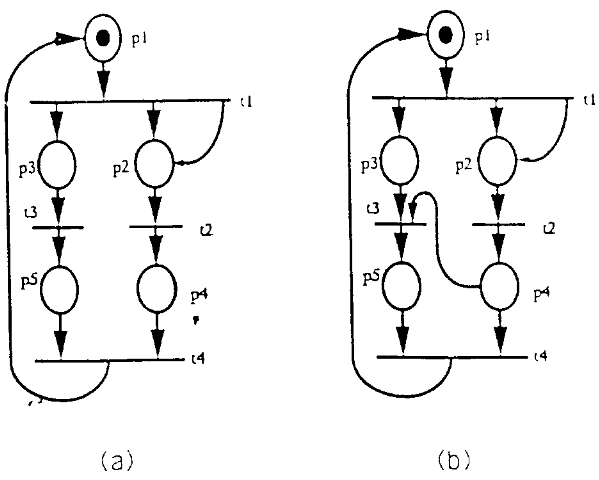


Fig. 14 An example of TP-path ([t1 p2] generation causing the net unbounded.
(b) The generation of [p4 t3] turns the net bounded again.

REFERENCES

1. Yaw, Y., "Analysis and Synthsis of Distributed Systems and Protocols," Ph.D. Dissertation, Dept. of EECS, U.C. Berkeley, 1987.
2. _____ and F. L. Foun, "The algorithm of a synthesis technique for concurrent systems," *1989 IEEE Int. Workshop on Petri Nets and Performance Models*, pp. 266-276, Tokyo.
3. D.Y. Chao, M.C. Zhou, and D.T. Wang, "Extending knittin technique to Petri net synthesis of automated manufacturing systems," *The Computer Journal*, Oxford University Press, Vol. 37, No. 1-2, pp. 1-10, 1994.
4. Ramamoorthy, C.V., and So, H., "Software Requirements and Specifications: Status and Perspectives," *IEEE Tutorial: Software Methodology*, 1978.
5. Yau, S.S., and Caglayan, M.U., "Distributed Software System Design Representation Using Modified Petri Nets," *IEEE Trans. on Software Engineering*, Vol. SE-9, No. 6, Nov. 1983, pp. 733-745.
6. Murata, T., "Modeling and Analysis of Concurrent Systems," in *Handbook of Software Engineering*, edited by C. Vick and C.V. Ramamoorthy Van Nostrand Reinhold, 1984, pp. 39-63.
7. _____, "Circuit theoretic analysis and synthesis of marked graphs," *IEEE Trans. Circ. and Sys.*, CAS-27, 1977, pp. 400-405.
8. _____, "Synthesis of decision-free concurrent systems for prescribed resources and performance," *IEEE Trans. Software Engineering*, SE-6, NO. 6, 1977, pp. 400-405.
9. Ramamoorthy, C.V., Dong, S.T., and Usuda, Y., "The Implementation of an

Automated Protocol Synthesizer (APS) and Its Application to the X.21 Protocol," *IEEE Trans. on Software Engineering*, No. 9, September 1985, pp. 886-908.
10. Ramamoorthy, C.V., Y. Yaw, W.T. Tsai, R. Aggarwal, and J. Song, "Synthesis of Two-Party Error-Recoverable Protocols," *Computer Communication Review* (USA), Vol. 16, No. 3, Aug. 1986, pp. 227-235.
11. ____, Y. Yaw, W.T. Tsai, R. Aggarwal, and J. Song, "Synthesis and Performance Evaluation of Two-Party Error-Recoverable Protocols, *COMSAC Symp.* Oct. 1986, pp. 214-220.
12. Yaw, Y., Ramamoorthy, C.V., and Tsai, W.T., "A Synthesis Technique for Designing Concurrent Systems," Second Parallel Processing Symposium, April 1988, pp. 143-166.
13. ____, Ramamoorthy, C.V. and Tsai, W.T., "Synthesis Rules for Cyclic Interactions Among Processes in Concurrent Systems" *COMSAC Symp.* Oct. 1988, pp. 496-504.
14. Krogh, B.H. and C.L. Beck. "Synthesis of place/transition nets for simulation and control of manufacturing Systems," *Proc. 4th IFAC/IFORS Symp. Large Scale Systems,* Zurich, 1986.
15. Agerwala, T. and Y. Choed-Amphai. "A synthesis rule for concurrent systems," *Proc. of Design Automation Conference,* 1978, pp. 305-311.
16. Narahari, Y. and N. Viswanadha (1985). "A Petri net approach to the modeling and analysis of flexible manufacturing systems," *Annals of Operations Research,* 3, pp. 449-472.
27. Koh I. and F. DiCesare. "Transformation methods for generalized petri nets and their applications in flexible manufacturing systems," *IEEE Trans System, Man, and Cybernetics,* 1991. Vol. 21(6), pp. 963-973.
18. Valvanis, K.S., "On the hierarchical analysis and simulation of flexible manufacturing systems with extended Petri nets," *IEEE Trans. on System, Man and Cybernetics,* SMC-20, 1, pp. 94-100.
19. Esparza, J. and M. Silva, "Circuits, handles, bridges, and nets," *LNCS, Advances in Petri Nets 1991,* Springer-Verlag, pp. 210-242.
20. ____ and ____, "On the analysis and synthesis of free choice systems," *LNCS, Advances in Petri Nets 1991,* Springer-Verlag, pp. 243-286.
21. D.Y. Chao and D.T. Wang, "A synthesis technique of General Petri Nets," *J. Systems Integration,* Vol. 4, No. 1, 1994, pp. 67-102.
22. ____ and ____, "An Interactive Tool for Design, Simulation, Verification, and Synthesis for Protocols," *Software-Practice and Experience, an International Journal,* Vol. 24, No. 8, august 1994, pp. 747-783.
23. ____ and ____, "Petri Net Synthesis and Synchronization Using Knitting Technique," *1994 IEEE Int'l Conf. SMC,* San Antonio, TX, October 2-5, pp. 652-657.
24. ____ and ____, "Knitting Technique and Structural Matrix for Deadlock Analysis and Synthesis of Petri Nets with Sequential Exclusion", *1994 IEEE Int'l Conf. SMC,* San Antonio, TX, October 2-5, pp. 1334-1339.
25. ____ and ____, "Knitting Technique with TP-PT Generations for Petri Net Synthesis," (Invited) *Proc. 1995 IEEE Int'l Conf SMC,* Vancouver, Canada, October 22-25, 1995. pp. 1454-1459.
26. ____, "Linear Algebra Based Verification of Well-Behaved Properties and P-Invariants of Petri Nets Synthesized Using Knitting Technique," MIS Review, Vol. 5, December 1995, pp. 27-48.

27. Y. Chen, W.T. Tsai, and D.Y. Chao, "Dependency Analysis — A Compositional Technique for Building Large *IEEE Trans. on Parallel and Distributed Systems*, Vol 4, No. 4, pp. 414-426, 1993.
28. Wang, D.T., and Chao, D.Y., "Enhanced Knitting Technique to Petri Net synthesis," *Proc. 1994 IEEE Int'l Conf SMC*, San Antonio, Texas, October 2-5, 1994, pp. 658-663.
29. ____ and ____, "New Knitting Technique for Large Petri Net Synthesis with Automatic Preservation of Liveness, Boundedness and Reversibility," (Invited) *Proc. 1995 IEEE Int'l Conf SMC*, Vancouver, Canada, October 22-25, 1995. pp. 1460-1465.
30. R. Valette, "Analysis of Petri Nets by Stepwise Refinement," *Journal of Computer and System Sciences* 18, 1979, pp. 35-46.
31. T. Murata, and J.Y. Koh, "Reduction and expansion of live and safe marked-graphs," *IEEE Trans. Circuits Syst.*, Vol. Cas-27, Jan. 1980, pp. 68-70.
32. Suzuki. I., and Murata, T., "A method for stepwise refinements and abstractions of Petri Nets," *J. of Comp. and Syst. Sci.* 27, 1983, pp. 51-76.
33. 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.
34. Berthelot, G., "Preuve de non blocage de programmes paralleles par reduction de reseaux de Petri," in *Proc. First Conf. Parallel and Distributed Processing*, Toulouse, France, 1979, pp. 251-259.
35. Hyung. Lee-Kwang and Favrel, J., "Analysis of Petri nets by hierarchical reduction and partition," in *IASTED Modelling and Simulation*. Zurich, Switzerland: Acta Press, 1982, pp. 363-366.
36. ____, Lee-Kwang and Favrel, J., "Hierarchical reduction method for analysis and decomposition of Petri nets," *IEEE Trans. Syst., Man, Cybern.*, Vol. SMC-15, Mar. 1985, pp. 272-280.
37. ____, Lee-Kwang, "Generalized Petri Net Reduction Method" *IEEE Trans. Syst., Man, Cybern.*, Vol. SMC-17, No. 2, 1987, pp. 297-303.
38. Y.S. Kwong, "On reduction of asynchronous systems." *Theorit. Comput. Sci.*, Vol. 5, 1977, pp. 25-50.
39. R. Johnsonbaugh and T. Murata, "Additional method for reduction and expansion of marked-graphs," *IEEE Trans. Circuits Syst.*, Vol. CAS, Oct. 1981, pp. 1009-1014.
40. Dong, T., "The Modeling, Analysis, and Synthesis of Communication Protocols," Ph.D Dissertation, Computer Science Division, EECS.
41. Ramamoorthy, C.V., Yaw, Y., and Tsai, W.T., "A Petri Net Reduction algorithm for Protocol Analysis," *Computer Communication Review* (USA), Vol. 16, No. 3, Aug. 1 986, pp. 157-166.
42. Lee, K.H. and J. Favrel. "Hierarchical reduction method for analysis and decomposition of Petri nets," *IEEE Trans. Systems, Man, and Cybernetics*, SMC-15, 2, 1985, pp. 272-280.
43. Jeng, M.D. and F. DiCesare, "A Review of Synthesis Techniques for Petri Nets with Applications to Automated Manufacturing Systems," *IEEE Trans. Syst., Man, Cybern.*, Vol. SMC-23, No. 1, 1993, pp. 301-312.
44. ____ and ____, "A modular synthesis techniques for Petri nets," *1992 Japan-USA Sym. on Flexible Automation*, pp. 1163-1170.
45. A. Datta, "Modular synthesis of deadlock-free control structures," *Foundations of Software Technology and Theoretical Computer Science*, Vol. 241, G. Goos and J.

Hartmanis (ed.), Springer-Verlag 1986, pp. 288-318.

46. M.C. Zhou and F. DiCesare, "Parallel and sequential mutual exclusions for Petri net modeling for manufacturing systems with shared resources," *IEEE Trans. on Robotics and Automation,* 7(4), 1991, pp. 515-527.
47. _____, DiCesare, F. and Desrochers, A.A., "A hybrid methodology for Petri net synthesis of manufacturing systems," *IEEE Trans. on Robotics and Automation,* Vol. 8, No. 3, 1992, pp. 350-361.
48. _____, McDermott, K. and Patel, A., "Petri Net Synthesis and Analysis of a Flexible Manufacturing Systems Cell", *IEEE Trans. SMC,* Vol. 23, No. 1, March 1993, pp. 524-531.
49. D.Y. Chao and D.T. Wang, "Synchronized Choice Ordinary Petri Net," (Invited) *Proc. 1995 IEEE Int'l Conf SMC,* Vancouver, Canada, October 22-25, 1995. pp. 1442-1447.
50. Murata, T., "Petri Nets: Properties, Analysis and Applications," *IEEE Proceedings,* Vol. 77, No. 4, April 1989, pp. 541-580.
51. Peterson, J.L., *Petri Net Theory and the Modeling of Systems,* Prentice-Hall, Englewood Cliffs, New Jersey, 1981.
52. Silva, M. "Toward a synchrony theory for P/T nets," in *Concurrency and Nets,* K. Voss, H. J. Genrich, and G. Rozenberg (Eds.), pp. 435-460, Springer-Verlag.
53. Chao, D.Y., and Wang, D.T., "XPN-FMS: A Modeling and simulation software for FMS Using Petri nets and X window," *Int'l J. Flexible Manufacturing systems,* Vol. 7, No. 4, October 1995, pp. 339-360.
54. _____ and _____, "Iteration bounds of single-rate data flow graphs for concurrent processing," *IEEE Trans. Circuits and Systems,* Vol. 40, CAS-1, pp. 629-634, September 1993.
55. _____, "Application of final matrix to data flow graph scheduling using multiprocessors," MIS Review, Vol. 5, December 1995, pp. 65-80.

Appendix I. Proof of Theorem 1

Proof: This lemma can be proved by repetitively applying reduction and expansion [HYU82,85,87, YAW87, RAM86a]. □

Appendix II: Proof of Theorem 2

Proof: This lemma can be proved by repetitively applying reduction [HYU82,85,87, YAW87, RAM86a]. □

Appendix III: Proof of Lemma 1

Proof: Assume contrary, then there are two n_{ps}, one a $t \in T$ and another a $p \in P$. This can only happen when they are *PT* or *PP* generations between concurrent *PSP*s or *TP* or *TT* generations between exclusive *PSP*s. All these four types of generations are forbidden. □

Appendix IV: Proof of Lemma 2

Proof: After each Π in C_{12} fires once, there exists a σ for one of C_{21} to gain a token. By the equivalence of temporal and structural relationship, each Π in C_{21} will gain a token. Hence the lemma. □

Appendix V: Proof of Lemma 3

Proof: This lemma comes from the fact that prior to the generation, $d(X_{gj}, X_{jg}) = 1$, since $X_{gj} \leftrightarrow$ or $\parallel X_{jg}$. □

Appendix VI: Proof of Observation I

Proof (1) $|\bullet t| > 1$ only for transition joints within the *NP*. But this occurs only under Rule PP.2.1 where the joint is a $\mathring{n}_g$. (2) Consider Rule TT.4 first. Any pair of new *PSP*s corresponding to the *TP-path* using Rule TT.4.1 are mutually exclusive since they have the same p_{ps} as the two corresponding Π_gs. The same result applies to Rule TT.4.2 since the n_{ps} is a place p_{j}. The case for Rule PP.2 can be proved in a dual fashion. Cases (3) and (4) can be similarly proved. □

Appendix VII: Proof of Lemma 4

Proof: We first show that the injected tokens will eventually disappear from the *NP* due to the following facts: (1) There are no internal *TP-path* whose $p_j = n_j$ and (2) Injected tokens can flow freely inside the *NP* since every t inside the *NP* has only one input place by Observation I. We then show that upon the above disappearing of the injected tokens from the *NP*, the resultant M^2 contains a $M^{1/2} \in R^{1a}$. This is true for Rule TT.4 since the sub-firing sequence in $N^{1/2}$ leading to the firings of t_g and t_j is exactly the same as that in N^{1a} prior to the generation. For Rule PP.2, the token disappearing is equivalent to the switching of tokens from C_{gj} to C_{jg} which leads to a submarking $M^{1/2} \in R^{1a}$ by Lemma 2. □

Appendix VIII: Proof of Lemma 5

Proof: This lemma comes directly from Lemmas 3 and 4. □

Appendix IX: Proof of Theorem 3

Proof: $\forall \sigma$ that does not include any node in the *NP*, it will fire exactly the same manner as that in N^{1a} and leads to the same reachable marking. Hence this case need not be considered.

Reversible: We need to show that a σ exists such that $(M_{\alpha}^{1a}, M_0^{w})$ $[\sigma > (M_0^{1a}, M_0^{w})$.

There are two cases: (a) $M^w = M^w_0$ and (b) $M^w \neq M^w_0$.

(a): Two cases: if σ (i) does or (ii) not include any t ϵ T^w. For case (ii), the subnet that involves σ are exactly those in N^{1a}. Hence, the σ exists. For case (i), Lemma 5 dictates that a $\sigma^w \subseteq \sigma$. Let $\sigma = \sigma_c \sigma^w \sigma_d$, then (M^{1a}_α, M^w_0) $[\sigma_1 \sigma^w >$ (M^{1a}_β, M^w_0), where $M^{1a}_\beta \, \epsilon \, R^{1a}$. The problem then reduces to case (ii) which has been proved.
(b): $M^w \neq M^w_0$, which implies a partial σ^w. By Lemma 5, all subsequent firing sequences, if long enough, can complete the rest of σ^w to restore M^w_0. The case then is similar to case (a) and can be proved similarly.

Bounded: Assume contrary, and unbounded places are in (a) N^{1a} or (b) the *NP*. (a) is impossible because the generation only at most eliminates (not adds) some reachable markings. (b) is impossible, because by Lemma 5, t_g or $\mathring{n}_g$ would not fire infinitely often relative to t_j or $\mathring{n}_j$.

Live: Assume contrary, then $\exists$ a t ϵ T^2; some of its input places, which are mutually concurrent, can never get tokens. But this is impossible by the equivalence between temporal and structural relationship. □

Appendix X: Proof of Theorem 4

Proof: Lemma 5 does not hold for Rule TT.3.1 since Lemma 3 does not either, which is because t_g may fire infinitely often relative to t_j. Now Rule TT.3.2 prevents such an infinite firings of t_g with respect to t_j and forces $d_{gj} = 1$. Thus Lemma 3, and hence Lemma 5, also hold for Rule TT.3. The rest of proof follows that for Rule TT.4 in Theorem 3. □

Appendix XI: Proof of Theorem 5

Proof: Prove by induction. It is easy to see that a basic process is live, bounded and reversible. Then, assuming the N from the (N-1)th synthesis steps is bounded, live and reversible, the proof needs to show that after each full synthesis step using a certain TT or PP rule, the resulting N^{1a} remains so. The correctness of which is established by Theorems 1-4. Note that we have been assuming that there is only one token in every home place. Now prove the property of marking monotonic by showing that N^b is well-behaved.

Reversible: Let $M^{2a}_\alpha \, \epsilon \, R^{2a}$ in N^{2a}, then

$$M_\alpha^{2a} = M_0^{2a} + Ax^{2a}$$

where A is the incidence matrix and x^{2a} a firing vector. Since N^{2a} is reversible, $\exists$ a x^{2a} such that $Ax^{2a} = 0$ and $M_\alpha^{2a} = M_0^{2a}$. Now add some tokens to some places in N^{2a} to result in N^b. The relevant equation is:

$$M_\alpha^b = M_0^b + Ax^b$$

The above firing vector x^{2a} also makes $M_\alpha^b = M_0^b$ since $Ax^{2a} = 0$. Now choose the least positive integer u such that $x = ux^{2a} - x^b \geq 0$; i.e., every component of x is nonnegative. We have

$$M_\alpha^b + Ax = M_0^b + A(ux^{2a}) = M_0^b$$

Thus N^b is reversible.

Bounded: Since N^{2a} is bounded, $\exists$ an P-invariant $y>0$ such that

$$(M_\alpha^{2a})^T y = (M_0^{2a})^T y + (x^{2a})^T A^T y$$

Since $A^T y = 0$, we have

$$(M_\alpha^{2a})^T y = (M_0^{2a})^T y$$

This equation also applies to N^b so that

$$(M_\alpha^b)^T y = (M_0^b)^T y$$

The above equation will not hold if any component of M_α^b is ∞ since $y>0$. Thus N^b is bounded.

Live: The proof of N^b being live is exactly the same as that in Theorem 3. □

Appendix XII: Proof of Theorem 6

Proof: A basic process is obviously a SC. Then assume N^1 is a SC, we need to prove that N^2 is also a SC. This can be proved by showing that any two *PSP*s joining at a place (transition) in a SC are mutually exclusive (concurrent). For otherwise, there exist $\Pi_1 \| \Pi_2$ and whose n_{pe} is a place. The two *PSP*s, which are incident to n_{pe} and on the two DEPs from Π_1 and Π_2 to n_{pe} respectively, are also mutually concurrent. But they join at a place — a contradiction. The same conclusion applies if $\Pi_1 | \Pi_2$.

This condition applies to N^1 since it is a SC. To prove it for N^2, we need to do it for only the joints inside the *NP*, which is true by (4) of Observation I. □

Appendix XIII: Proof of Lemma 6

Proof: Case (1) is obvious when neither $i = r$ nor $k = q$. For i=r, it is discussed as follows. The presence of Π_l affects the following entries: (a) $A_{rq} = A_{gj}$ and (b) A_{hq}, where $\Pi_g \rightarrow \Pi_h$, $\neg (\Pi_h \rightarrow \Pi_q)$ and $\neg (\Pi_q \rightarrow \Pi_h)$ (note $\Pi_j \rightarrow \Pi_q$).

For case (a), prior to the generation, no path passes through both Π_r and Π_q (otherwise $\Pi_r \rightarrow \Pi_q$); after the generation, there is a path passing through Π_r, Π_l, and Π_q. Hence, $A_{rq} = SE$ and this completes the proof for case 1). For case (b), note the two paths $\Pi_{g2} \ldots \Pi_h$ and $\Pi_l \Pi_{j2} \ldots \Pi_q$ intersect at the generation point of Π_l;. this might change the relationship between Π_h and Π_q. However, it is easy to see that $A_{hq} = A_{gj}$ and from the structure definitions of "|" and "||", we have $A'_{hq} = A_{hq}$. Hence except for $mz = rq$, $A'_{mz} = A_{mz}$ where $m, z \in \{r, q, u, v\}$. Hence case (2) is proved.

Case (3) can be proved similarly to that for cases (1) and (2).

Appendix XIV: Notations

A_{ab}: entries of structure matrix
AC: assymetric-choice nets
CN (||): *concurrent*
CL (o): *cyclic*
C_{ik}: LCN (Π_i, Π_k)
VP: *virtual PSP* which has only the generation point and the joint
EFC: extended free-choice nets
EX (|): *exclusive*
FC: free-choice nets
H: the set of home places
IG: interactive generation
LCN: local concurrent set of *PSP*s
LEX: local exclusive set of *PSP*s
M: marking

$M^2(N^1)$ or $M^{1/2}$: the marking of N^1 in N^2
N: a PN
N^a: a synthesized PN with every home place holding only one token
N^b: a PN after adding tokens to N^a
N^{1a}: a N^1 and N^a
N^{2a}: a N^2 and N^a
$N^{1/2}$: the subnet N^1 in N^2
n: the number of *PSP*s in a Petri net
n_i: the number of *PSP*s before the ith generation
n_g ($\mathring{n}_g$): the (next) generation point
n_j ($\mathring{n}_j$): the (next) joint
n_{ps}: (p_{ps}, t_{ps}): the prime start node (place, transition)
NP: new path
p: place
$\bullet p$ ($p\bullet$): the set of input (output) transitions of place p
p_s (p_{ps}): the (prime) start place for a pair of exclusive *PSP*s.
P: the set of places in net N.
DEP: directed elementary path
PG: pure generation
Π: *PSP*
Π_g: the *PSP* containing n_g
Π_j: the *PSP* containing n_j
Π^w: new *PSP*
PN: Petri net
PP generation: place-place path generation
PP-path: place-place path
PT-path: place-transition path
PSP: pseudo-process
R or $R(N,M_0)$: reachable set of markings of N with initial marking M_0
t: transition
$\bullet t$ ($t\bullet$): the set of input (output) places of transition t
t_s (t_{ps}): the (prime) start transition for a pair of concurrent *PSP*s.
T: the set of transitions in net N
T-Matrix: temporal matrix
TP-path: transition-place path
TT generation: transition-transition path generation
SE or *E* ($\rightarrow$): *sequentially earlier*

SL or *L* ($\leftarrow$): *sequentially later*
SN or *N* ($\leftarrow$): *sequentially next*
SP or *P* ($\leftarrow$): *sequentially previous*
SQ: *sequential*
subscript g: related to generation point
subscript j: related to joint
superscript 1: for the net prior to the generation; i.e., N^1 (T^1, P^1, M^1, R^1) refers to the net (T, P, M, and R) prior to the generation
superscript 1a: for the net prior to the generation and every home place has only one token; i.e., N^{1a}: (T^{1a}, P^{1a}, M^{1a}, R^{1a})
superscript 2: for the net after the generation; i.e., N^2 (T^2, P^2, M^2, R^2) refers to the net (T, P, M, and R) after the generation
superscript w: for the *NP*; i.e., (T^w, P^w, M^w) refers to the (T, P. M) of the *NP*. σ^w denotes a complete firing sequence through the *NP*, which includes both n_g and n_j.
σ: transition firing sequence
$\sigma(t)$: the number of firings of t in σ.
X_{ik}: *LEX* (Π_i, Π_k)
K: the total number of final *PSP*s in the final system