Petri-Net-Based Deductive Reasoning Strategy for Fault Identification in Batch Processes

Yi-Feng Wang and Chuei-Tin Chang*

Department of Chemical Engineering, National Cheng Kung University, Tainan, Taiwan 70101, Republic of China

In implementing any hazard analysis method, there is always a need to reason deductively to identify all possible fault origins that could lead to an undesirable consequence. Because of the complex time-variant cause-and-effect relations between events and states in batch chemical processes, a rigorous risk assessment study is labor- and time-consuming, and its results are often error-prone. The aim of this paper is thus to develop deductive reasoning algorithms on the basis of Petri nets for automating such cause-finding procedures. The effectiveness and correctness of this approach is successfully demonstrated with a number of practical examples.

1. Introduction

To ensure operation safety, hazard analysis is one of the basic tasks that must be performed while designing or revamping any chemical process. A variety of techniques have already been proposed in the literature, including hazard and operability (HAZOP) studies, fault-tree analysis (FTA), and failure mode and effect analysis (FMEA). In applying any of these methods, there is always a need to identify all possible causes of an undesirable consequence with a *deductive* reasoning approach. Traditionally, the task of characterizing the corresponding fault propagation scenarios is performed manually on an ad hoc basis. For a complex chemical process, the demand for time and effort is often overwhelming. To alleviate this practical problem, many attempts have been made to automate the cause-finding process on the basis of various qualitative models.

The research on automatic hazard analysis has advanced significantly in the past 2 decades. Many efficient tools have been employed for the development of fault-tree synthesis algorithms, e.g., digraphs,[1,2] decision tables,[3] and mini-fault trees,[4,5] among others. Several generic expert systems have also been constructed to produce comprehensive HAZOP reports.[6−8] The prerequisite of fault identification in using these methods is basically a qualitative system model. It can be observed from the literature that digraphs are, by far, the most popular choice.[9−12] Although the digraph-based approach has been demonstrated to be useful, it is effective mostly in applications concerning *continuous* processes. This is because digraphs are inherently unsuitable for describing dynamic causal relationships among times, discrete events, equipment states, and system configurations in *batch* or *semibatch* processes.

In the recent literature, Petri nets (PNs) have often been used as a modeling tool to circumvent the above drawbacks.[13,14] Wang et al.[15] and Wang and Chang[16] have developed a step-by-step procedure for constructing the appropriate Petri nets for the modeling of normal and abnormal batch operations. Systematic simulation techniques have also been proposed to enumerate all critical fault propagation scenarios. In implementing such an approach, it is necessary to first acquire a comprehensive list of failure modes associated with each component in the system. Next, all failure scenarios must be generated with repeated simulation runs. The possible causes of the undesirable consequence under investigation can then be identified from the simulation results. However, as the system complexity increases, the number of cases requiring simulation becomes extremely large. Thus, this fault identification procedure tends to be tedious and ineffective if it is applied to realistic systems.

To relieve the work load, a novel PN-based deductive reasoning method is developed in this study. Essentially, this method is implemented on the basis of backward Petri nets. To illustrate the proposed deductive reasoning strategy, the rest of this paper is organized as follows: Because an accurate (forward) system model must be constructed to describe the fault propagation behaviors first, the model-building approach is briefly outlined in section 2. In section 3, a simple procedure is presented to transform the system model into the corresponding backward Petri nets. The time-stamped fault-tree synthesis rules are then provided in section 4 for the representation of deductive reasoning processes. A modified version of the conventional state equation is developed in section 5 to facilitate computer implementation of the fault-tree construction algorithm. Section 6 is concerned with a special failure mode that can be described with the stalled transitions in Petri-net models. The above PN-based reasoning techniques are all incorporated into the systematic fault identification procedure presented in section 7. Finally, this procedure is applied to an air-drying process in section 8 to demonstrate the effectiveness of the proposed strategy.

2. System Model

A formal mathematical description of an *ordinary* Petri net can be found in Peterson.[17] As originally designed, it comprises only three types of elements, namely, discrete places, discrete transitions, and normal arcs. A discrete place is marked by a circle, and a discrete transition is marked by a bar. A normal arc is represented by a directed solid line. It connects either a place to a transition or vice versa.

* To whom correspondence should be addressed. Tel.: 886-6-275-7575 ext. 62663. Fax: 886-6-234-4496. E-mail: ctchang@mail.ncku.edu.tw.

10.1021/ie034026l CCC: $27.50
Published on Web 04/27/2004

To facilitate proper representation of the sequential operations in batch processes, several special extensions are also employed in the present study. Following is a list of these additional transitions and arcs:[18]

(i) Timed Transitions. The introduction of deterministic time labels into Petri nets was first attempted by Ramchandani.[19] In his applications, a time label was placed at each transition, denoting the facts that transitions are often used to represent actions and actions take time to complete. On the other hand, if a transition is without a time label, it is usually treated as an untimed transition. This added feature is especially suitable for characterizing various operating steps in practical batch processes.

(ii) Weighted Arcs. A distinct positive integer k can be used to label each arc. A k-weighted arc is interpreted as a set of k parallel normal arcs. These weighted arcs are used in this work mainly to allow more flexibility in system modeling. Notice also that the labels for unity weights are often omitted from the Petri nets for the sake of simplicity.

(iii) Inhibitor Arcs. An inhibitor arc is usually represented by a directed solid line pointing to a transition with a small circle at its end. This type of arc can be used in executing zero tests or in modeling the failure mechanisms that inhibit certain normal events in operation.

(iv) Static Test Arcs. A static test arc is marked by a directed dash line pointing to a transition. In general, such arcs are used to replace self-looping structures in Petri nets. In other words, a static test arc is equivalent to two equally weighted arcs in opposite directions.

The execution of a Petri net is controlled according to the *token distribution* in the net. Specifically, the following enabling and firing rules should be employed to govern the token flows:

Enabling Rule. A transition t_i is said to be *enabled* if the number of tokens in each input place p_j is greater than or equal to the weight of the weighted arc (or static test arc) connecting p_j to t_i, i.e., $M(p_j) \geq W(p_j,t_i)$, $\forall p_j \in P_i$, or less than the weight of the inhibitor arc connecting p_j to t_i, i.e., $M(p_j) < W(p_j,t_i)$, $\forall p_j \in P_i$, where $M(p_j)$ denotes the token number in place p_j, $W(p_j,t_i)$ denotes the weight associated with the arc from place p_j to transition t_i, and P_i is the set of all input places connecting to transition t_i.

Firing Rule. A transition t_i is *firable* at time ST if and only if it is enabled continuously during the time interval $[ST - \Delta\theta(t_i), ST]$, where $\Delta\theta(t_i)$ denotes the time delay associated with transition t_i.

If an input place is connected to a transition with a weighted arc, a portion of the place's tokens must be removed after the transition is fired . The number of removed tokens should equal to the weight on the corresponding place-to-transition arc. Moreover, additional tokens must be deposited in each output place at the same time, and the number of deposited tokens should equal the weight of the corresponding transition-to-place arc. Finally, if an input place is connected to a transition with an inhibitor arc or a static test arc, then its tokens should *not* be removed after firing.

In constructing a Petri-net-based system model, a hierarchical approach can be followed to describe the operating steps in batch processes. Basically, each item in the piping and instrumentation diagram P&ID is described with a component model. The normal behavior of each component should first be represented with the Petri-net elements described above. Moreover, to facilitate hazard analysis, it is also necessary to incorporate additional elements in each component model to depict its failure mechanisms. All completed component models can then be classified into a hierarchy of four different levels (see Table 1). In building the system model, these component models should be assembled one-by-one from the top level to the bottom level according to a given P&ID. It should also be noted that this Petri-net construction approach is in fact more suitable for smaller systems with moderately complex recipes. The net size can grow rapidly as the number of pieces of equipment in a process increases. This is due to the state-space explosion caused by the need to describe not only the process configurations but also the operating steps in an industrial-size system model. Consequently, the tasks of building the component models and then synthesizing the system model can be very time-consuming in practical applications.

Table 1. Hierarchy in PN-Based System Models for Sequential Operations

level	component models
1	timer, operator, PLC
2	valve, pump, compressor
3	process unit
4	sensor

Table 2. Arc Transformation Rules

original arc type	original arc weight	reversed arc type	reversed arc weight
normal	W	normal	W
static test	W	normal	W
inhibitor	W	normal	$W - 1$

Although the number of units in a realistic system might be large, it is often possible to group the units into relatively few equipment types, e.g., controllers, valves, pumps, separators, reactors, heat exchangers, etc. Thus, the model-building effort can be somewhat reduced by prefabricating a set of standard component models of the often-used units in all four levels of the hierarchy presented in Table 1. The system model can simply be built by retrieving the needed components from a database and then assembling them according to the given P&ID. The detailed description of this systematic model construction procedure is omitted from this paper since it has already been published elsewhere.[16]

3. Backward Petri Nets

The *backward* Petri net (BPN) is a useful vehicle for tracking the deductive reasoning steps in hazard analysis. It can be easily obtained from the system model according to the following conversion procedure:

1. Review the forward Petri net (FPN), i.e., the system model. If the weights of its inhibitor arcs are W_1, W_2, ..., W_I, then make $\prod_{i=1}^{I} W_i$ copies of the FPN. In each copy, the weights of the inhibitor arcs should be a unique combination of integers w_1, w_2, ..., w_I such that $1 \leq w_1 \leq W_1$, ..., $1 \leq w_I \leq W_I$.

2. Reverse the directions of all input and output arcs in the above PN copies.

3. Apply the arc transformation rules specified in Table 2 to determine the type and weight of each reversed arc.

Notice from Table 2 that, although static test arcs and inhibitor arcs are allowed in a forward Petri net, all of them should be transformed into normal or weighted

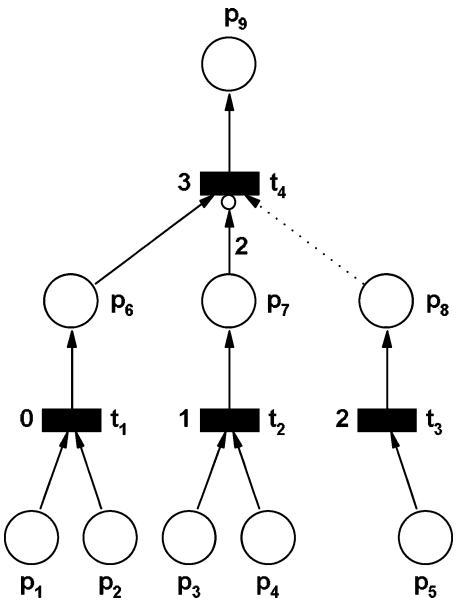

Figure 1. Forward Petri net.

arcs in the corresponding backward nets. Let us now consider the simple FPN in Figure 1 as an example. The delay times of transitions t_1, t_2, t_3, and t_4 are 0, 1, 2, and 3 units, respectively. Notice that, other than the nromal arcs, only two different types of input arcs are employed in this example, i.e., an inhibitor arc and a static test arc. By applying the proposed transformation procedure, the backward Petri nets in Figure 2a and b can be generated. These nets can be viewed as the road maps for reasoning deductively from a given consequence p_9 to the root causes represented by the places p_1-p_5.

4. Time-Stamped Fault Trees

A fault tree is a graphical system model. In essence, such a graph can be regarded as an accurate representation of the deductive reasoning process in fault identification. This is true because all logical interrelationships of the basic and intermediate events leading to the given top event can be clearly depicted in a tree.

Two basic types of logic gates, i.e., OR gates and AND gates, are used in a typical fault tree. Any other gate can be replaced by a proper combination of these two. In this study, a fault tree is developed layer-by-layer on the basis of the backward Petri net. The general two-layer fault-tree structure associated with an input place and its outputs in a BPN model can be found in Figure 3. The output of this fault-tree structure in the upper layer is the event represented by placing a token (or tokens) in the input place. On the other hand, the lower-layer events in the general structure can be simulated by firing the enabled transitions and then inserting tokens in the output places.

The key step in developing a particular two-layer fault-tree structure involves the identification of the actual number of AND gates connected to the OR gate and also of the input events connected to each AND gate. In this paper, two simple construction rules are employed for these purposes:

OR-Gate Rule. Identify a place that enables one or more transitions in the backward Petri net. Connect an OR gate to the output event associated with this place.

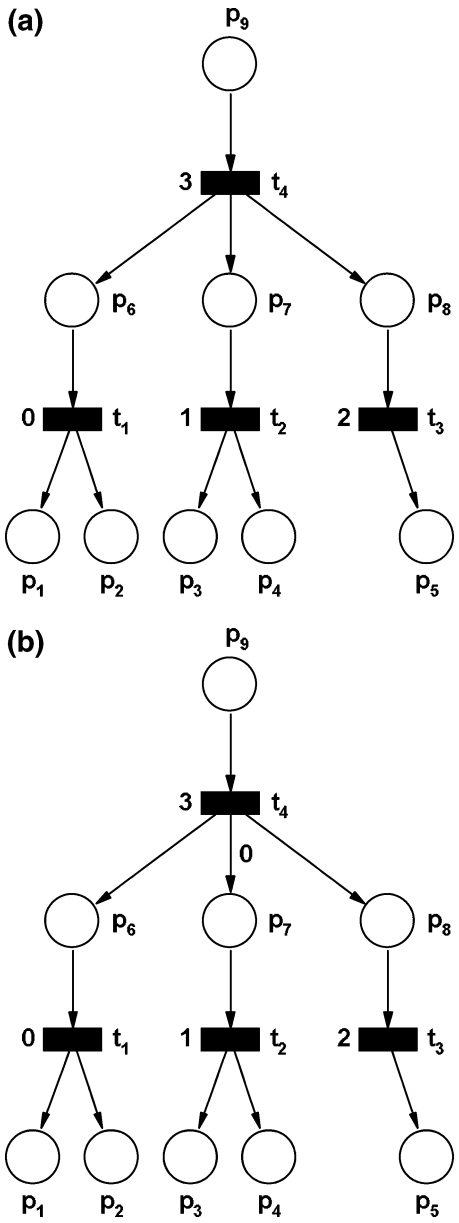

Figure 2. Backward Petri net obtained from Figure 1 for (a) $w_1 = 2$ and (b) $w_1 = 1$.

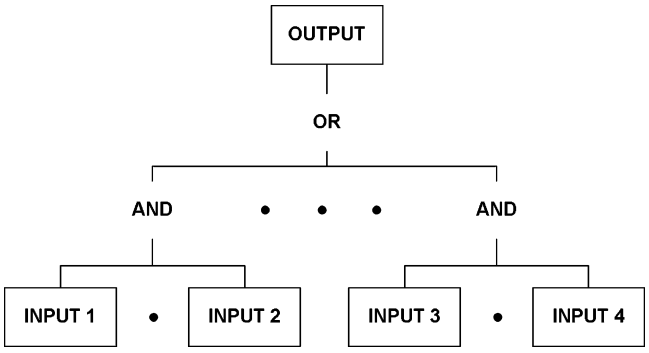

Figure 3. General fault-tree structure.

The number of AND gates connected to the OR gate should be the same as the number of transitions enabled by the identified place.

AND-Gate Rule. Establish a one-to-one correspondence between the enabled transitions and the AND gates obtained with the OR-gate rule. Fire these enabled transitions simultaneously. The input events of each

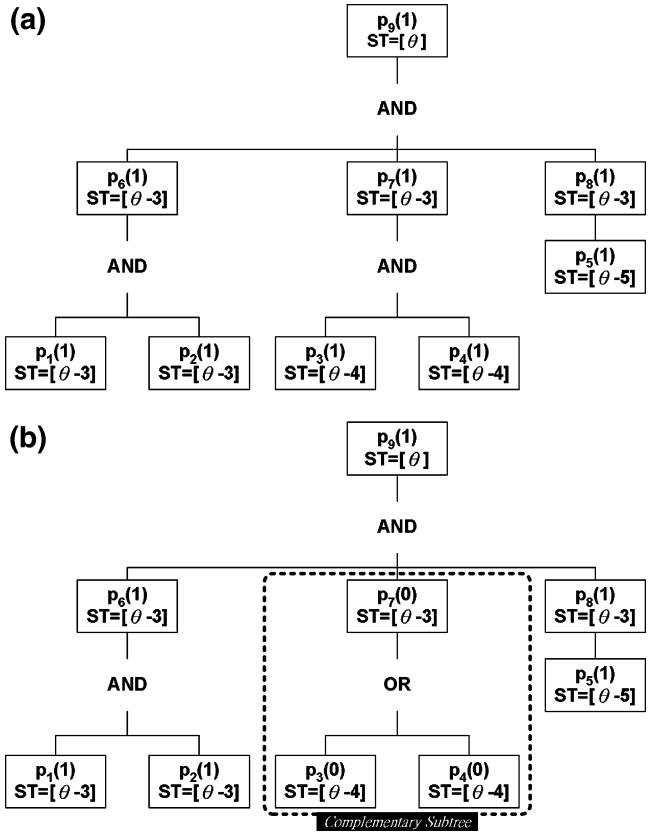

Figure 4. Time-stamped fault trees generated from (a) Figure 2a and (b) Figure 2b.

AND gate can be characterized by the token numbers in the output places of the corresponding transition.

In a conventional fault-tree representation, the causal relations among top, intermediate, and basic events are not expressed explicitly as functions of time. However, for the purpose of identifying fault propagation mechanisms in sequential operations, it is necessary to mark the occurrence time of each event in the fault tree. It should be noted that a set of time stamps can be systematically generated by executing the backward Petri net. Initially, a definite time can be assigned to the top event. The time stamp of each intermediate (or basic) event in the lower layer can be calculated by subtracting the time delay of the fired transition from the time stamp of the event in the upper layer. This calculation can be repeated until all events in the fault tree are marked with time stamps. For illustration purposes, let us consider the backward Petri net in Figure 2a as an example. It can easily be found that the proposed techniques can be used to synthesize the corresponding fault tree in Figure 4a.

As mentioned previously, an event in a fault tree can be represented by introducing a token (or tokens) into the corresponding place in the Petri net. An empty place is interpreted as the nonoccurrence of event(s) associated with the place under consideration. Because a place without tokens cannot enable any of its output transitions, the above two fault-tree construction rules are really not applicable. However, if inhibitor arcs are used in building the system model, then at least one of the corresponding BPNs should contain arcs with zero weights. This special feature can be found in Figure 2b. In this situation, the negation of $p_7(1)$, i.e., $p_7(0)$, is actually a sufficient condition for the occurrence of p_9-(1). Thus, there is a need to pinpoint the root causes that nullify the prerequisite for inhibiting a transition in the FPN model. This task can be achieved by developing a *complementary subtree* for the precondition of each inhibitor arc. A brief description of the subtree development techniques is given below.

Notice that, if the system model is built with the approach suggested by Wang and Chang,[16] then a place in the FPN always represents a discrete value of either an equipment state or a process condition. Depending on the modeling needs, multiple places can be used to reflect different values of the same state or condition. If the place under consideration is used to represent the *only* value of a state or condition needed in the system model, then the corresponding event should be used as the top event to develop a preliminary subtree, and then logical negation operation should be applied to generate the required complementary subtree on the basis of DeMorgan's theorem. Specifically, the following two steps should be carried out in sequence:

Subtree Construction, Identify a zero-weight arc in the backward Petri net. Insert a token at its output place. Apply the OR-gate and AND-gate rules repeatedly to build a preliminary subtree.

Negation Operation. Change every OR gate in the preliminary subtree into an AND gate and vice versa. Replace all events with their negation events.

If this two-step procedure is followed to develop the subtree associated with $p_7(0)$, then the complete fault tree presented in Figure 4b can be easily obtained. On the other hand, if more than one place is employed in the FPN to represent several different values of a state/condition, then the development of a complementary subtree is even more straightforward. The negation event associated with the place under consideration can simply be attached with an OR gate. The input events of this gate can be obtained by inserting a token in each of the places denoting other values of the same state or condition. The subtree can then be developed accordingly on the basis of the standard OR-gate and AND-gate rules.

5. Modified State Equation

As mentioned before, the backward Petri net is used in this work as a bookkeeping device in the deductive reasoning process. Although a BPN is intuitively easy to comprehend, it is not suitable for computer manipulation. Thus, a modified version of the state equation[20] is used instead for the purpose of monitoring token flows in synthesizing a fault tree. Specifically, this equation can be written as

$$\mathbf{m}_{k+1} = \mathbf{m}_k + \mathbf{B}\mathbf{u}_k \qquad k = 0, 1, 2, \ldots \tag{1}$$

where $\mathbf{u}_k$ is a column vector whose entries are binary numbers such that a 1 at the jth position indicates that transition j should be fired at the kth firing, whereas a 0 means otherwise; $\mathbf{m}_k$ is a column vector whose ith entry is the token number in place i; and $\mathbf{B}$ is a place-to-transition incidence matrix. A negative integer at the (i, j)th position of matrix $\mathbf{B}$ represents the negative value of the weight of an arc from the ith place to the jth transition. A positive integer at the (i, j)th position denotes the weight of an arc from the jth transition to the ith place. If the weight of a transition-to-place arc in the BPN is zero, a very small positive number ϵ ($0 < \epsilon \ll 1$) is assigned to the corresponding entry in the

Table 3. Marking Record in Synthesizing the Fault Tree in Figure 2a

k:	0	1	2	3	4
enabled transitions:	t_4	t_1	t_2	t_3	−
p_1	0	0	1	1	1
p_2	0	0	1	1	1
p_3	0	0	0	1	1
p_4	0	0	0	1	1
p_5	0	0	0	0	1
p_6	0	1	0	0	0
p_7	0	1	1	0	0
p_8	0	1	1	1	0
p_9	1	0	0	0	0

Table 4. Marking Record in Synthesizing the Fault Tree in Figure 2b

k:	0	1	2	3	3	4
enabled transitions:	t_4	t_1	t_3	−	t_2	−
p_1	0	0	1	1	1	1
p_2	0	0	1	1	1	1
p_3	0	0	0	0	0	1
p_4	0	0	0	0	0	1
p_5	0	0	0	1	1	1
p_6	0	1	0	0	0	0
p_7	0	ϵ	ϵ	ϵ	1	0
p_8	0	1	1	0	0	0
p_9	1	0	0	0	0	0

incidence matrix. Finally, only one possibility is associated with a zero entry, i.e., the corresponding arc does not exist at all.

Notice that the fault-tree construction rules can be implemented in sequence on the basis of the modified state equation. In particular, a specific version of the two-layer structure given in Figure 3 can be created after eq 1 has been executed exactly *once*. From the definition given above, it is clear that the current locations of tokens in a BPN are recorded in the vector $\mathbf{m}_k$. This vector is referred to as a system *marking* in the present paper. The application of OR-gate and AND-gate rules can be equivalently described with the firing-induced token flows from the current marking $\mathbf{m}_k$ to a future marking $\mathbf{m}_{k+1}$. To apply the OR-gate rule, it is necessary to identify a place in the current marking that enables at least one transition and also to identify these enabled transitions. To use the AND-gate rule, it is necessary to determine the output places connected to each of the enabled transitions and also to determine the token numbers in these places after firing. All of these tasks can be accomplished by implementing an algorithm developed on the basis of the modified state equation (see algorithm A in Appendix I).

The fault-tree construction rules can obviously be implemented repeatedly until none of the transitions in the backward Petri net are enabled. However, if the BPN contains zero-weighted arcs, the final marking obtained with the above approach might contain small positive numbers, i.e., ϵ. These entries must be developed further to produce the complementary subtrees. For illustration purpose, let us again consider the BPNs in Figure 2a and b. Their incidence matrices can be expressed as

$$\mathbf{B} = \begin{bmatrix} +1 & 0 & 0 & 0 \\ +1 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 \\ 0 & +1 & 0 & 0 \\ 0 & 0 & +1 & 0 \\ -1 & 0 & 0 & +1 \\ 0 & -1 & 0 & x \\ 0 & 0 & -1 & +1 \\ 0 & 0 & 0 & -1 \end{bmatrix} \qquad (2)$$

where the value of x is +1 in the case of Figure 2a and ϵ in the case of Figure 2b. The sequence of markings obtained for the synthesis of the fault tree in Figure 4a is presented in Table 3. On the other hand, the main tree in Figure 4b can be obtained by applying algorithm A repeatedly until the third marking (see Table 4). The preliminary subtree can be developed simply by replacing the ϵ value in $\mathbf{m}_3$ with 1 and then using the same procedure to generate the last marking.

6. Stalled Transitions

The occurrence of an event in the fault tree can be conveniently simulated by placing a token in the corresponding place in the system model. If the tree is constructed strictly according to the proposed OR-gate and AND-gate rules, then the implied assumption is that the place under consideration can be reached *only* by firing at least one of its *input* transitions in the FPN at the occurrence time. However, it should be noted that another possibility might have been ignored in this reasoning process, i.e., the given place might have acquired a token at a prior time and its *output* transitions might have all been stalled since then. Let us again consider the Petri net in Figure 1 as an example and further assume that it is actually a partial version of the net given in Figure 5a. Notice that three more discrete places (p_{10}−p_{12}) and one more untimed transition t_5 are included in this net. Notice also that all new arcs are normal except for the inhibitor arc connecting t_5 and p_{11}. It is obvious that a token can be introduced in p_8 at a given time θ^* if p_5 acquired one at time $\theta^* - 2$. On the other hand, it is also possible that p_8 had already obtained a token before $\theta^* - \Delta\theta$, where $\Delta\theta$ denotes a finite period. The requirement to keep this token in place p_8 at time θ^* is to maintain either a nonempty p_{11} or an empty p_{12} throughout the time interval $[\theta^* - \Delta\theta, \theta^*]$.

Clearly, the latter scenario in the above example cannot be deduced if the proposed fault-tree construction approach is applied to the original system model in Figure 5a. To facilitate proper use of the reasoning procedures developed previously, the original Petri net must be first converted to the one shown in Figure 5b. Notice that an additional discrete place (p_{13}) and three additional transitions (t_6−t_8) are introduced in this modified net. The added transitions t_6 and t_7 are untimed, but the delay time of t_8 is $\Delta\theta$. The two output arcs from p_{11} (and also the two from p_{12}) are logically complementary. In other words, the pair always consists of a normal arc and also an inhibitor arc. Notice also that all other added arcs are normal. Because the failure modes caused by the stalled transitions are now incorporated in the modified net as a substructure connecting to p_8 with its *input* transition t_8, the findings generated with the proposed reasoning procedure should be comprehensive.

An important conclusion can be drawn from the above example, that is, as long as the place under consideration is connected (with a normal or weighted arc) to one or more output transitions having other places as inputs, the same technique should be employed to

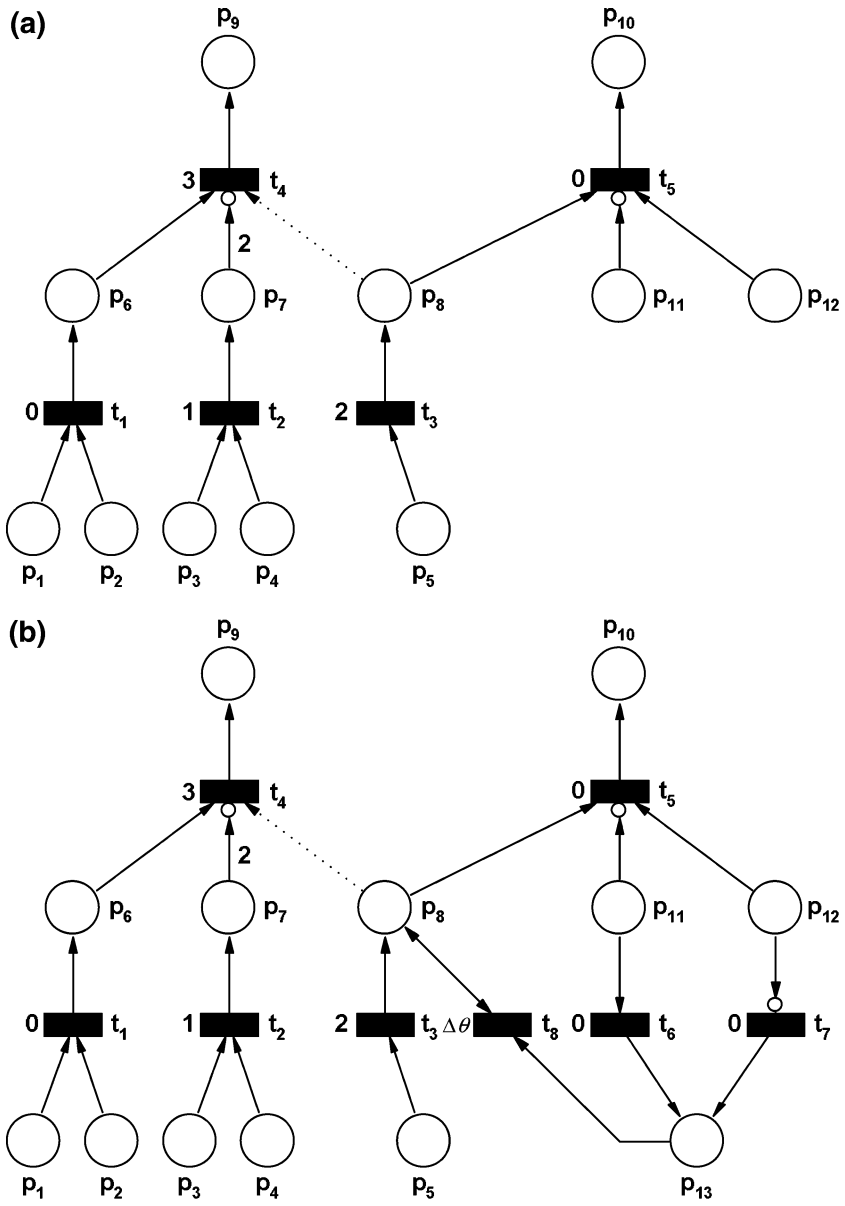

Figure 5. (a) Original and (b) modified system Petri nets.

generate a complete listing of all possible failure mechanisms. This type of place can often be found in the proposed FPN models. In particular, the equipment states are usually described by multiple places, and the steps of operation are always represented by transitions interconnecting them.[16] As a result, the model modification routine described here is, in fact, an indispensable task in the PN-based fault identification procedure. Finally, it should be noted that the dead time of the time-delayed transition in the attached substructure, e.g., t_8 is Figure 5b, must be specified in actual applications. A selection criterion for these time delays is given in the following section after the method for discretizing the time horizon has been properly explained.

7. Fault Identification Procedure

Our fault identification procedure can be summarized with the flowchart given in Figure 6. Instead of providing a detailed description, a simple example is used here to illustrate this procedure. Let us consider the mixing process depicted in Figure 7. Here, the desired product is produced in tank 3 by mixing raw materials stored in the other two tanks. Initially, the amount of liquid A in tank 1 is 1 m^3, and the amount of liquid B in tank 2 is 2 m^3. The mixing operation begins when both valve V1 and valve V2 are opened by an operator. The operator is instructed to close valve V2 whenever tank 1 is empty or vice versa. It is assumed that tank 1 can be emptied in 1 h and tank 2 in 2 h.

The results obtained by implementing each step of the fault identification procedure are provided in the following paragraphs.

(i) Setting the Time Horizon for Fault Identification. To avoid generating an excessively large list of insignificant scenarios, the scope of fault identification must obviously be limited. For a noncyclic batch process such as the one presented here, only the faults or failures occurring during the planned operating period are considered to be the candidate causes of an undesirable condition in the final product. In the present example, this planned production period is 1 h given that operation should be terminated when tank 1 is emptied. On the other hand, if the batch operation is cyclic in nature, the time horizon is restricted only to

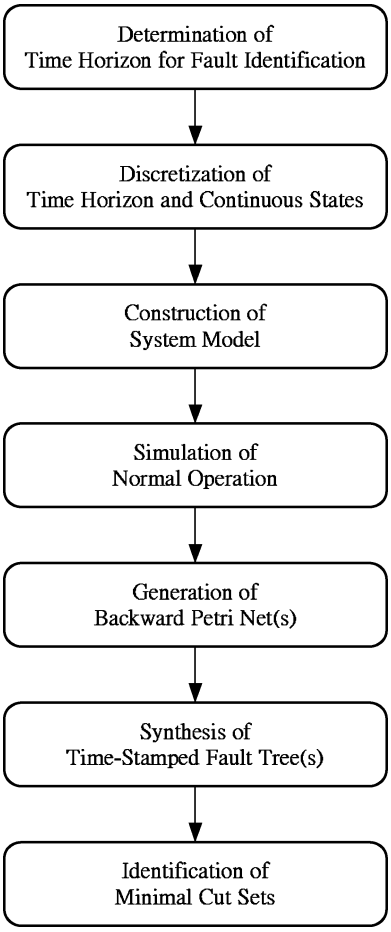

Figure 6. Fault identification procedure.

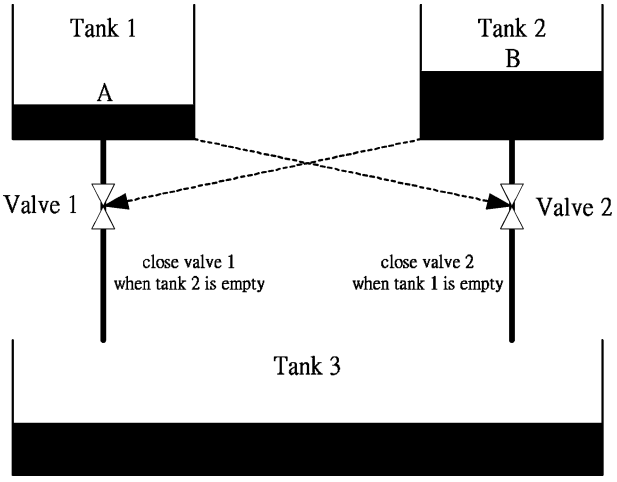

Figure 7. Process flow diagram of a mixing process.

one complete cycle prior to the consequence under investigation.

(ii) Discretizing the Time Horizon and Continuous States. Notice that implementing any operating step in a recipe should always cause a change (or changes) in the equipment state(s) of one or more process units. The main objective of this step is to ensure that all such changes can be properly modeled in a Petri net. Thus, the time horizon is classified into several distinct time periods according to the completion times of operating steps, and the value of each continuous variable is discretized into several ranges in such a way that the equipment state of any unit can be uniquely characterized in each time period. In particular, the value of a continuous variable in each time period is qualitatively treated as *one* level (range) and represented by a discrete place. As the number of steps in the operation increases, the numbers of time periods and qualitative levels (places) required to describe the corresponding changes in state certainly increase as well. On the other hand, it should also be noted that only a few continuous variables are needed to describe the state of a process unit. Because the number of units in a realistic batch process and the number of steps in its operating recipe are both finite, it is our belief that the size of the corresponding Petri net will still be manageable with a computer in fault identification applications.

Let us now consider the mixing process. The entire time horizon can be regarded as one period because only one action, i.e., closing V2, is performed during normal operation. More specifically, the time interval $0 \le \theta < 1$ is defined as period 1 in our analysis. The periods before and after the normal operation, i.e., $\theta < 0$ and $\theta > 1$, are referred to as periods 0 and 2, respectively. Let us next use the symbol Q to represent the liquid volume; this continuous variable can be qualitatively assigned to three discrete values, i.e., (1) value 0 ($Q = 0$), (2) value 1 ($1 \ge Q > 0$), and (3) value 2 ($2 \ge Q > 1$). Finally, notice that the instance of an operating action (i.e., $\theta = 1$) is not included in any of the periods. This practice is used to avoid ambiguity in describing the transition process between two discrete states.

(iii) Building the System Model. As mentioned previously, a procedure for constructing the system model has already been developed by Wang and Chang.[16] The same approach is followed here.

The batch operation in Figure 7 can be modeled with the FPN shown in Figure 8. Notice that, for simplicity, this model only contains components in the first three levels of the hierarchy shown in Table 1. The first-level component is the operator. For simplicity, only the two planned operating steps performed *after* the opening of valaves V1 and V2 are modeled here. The places PC(1) and PC(2) are used to represent the operating commands to close V1 and V2, respectively. The equipment states of V1 (and V2) in the second level can be described with two discrete places representing the open and closed positions, respectively. Two types of valve failures are considered in this example, i.e., sticking and failing to close. The corresponding failure models are also included in this Petri net. If a token is introduced in the place representing the event "valve sticking", this token should disable a transition representing the action of changing the valve position. The corresponding token is therefore locked in its input place. On the other hand, if a valve is originally open and the event "valve failing to close" occurs during operation, such a failure always moves the token from the place representing the open position to the other place denoting the opposite valve state. The tanks in this example are the third-level components. To characterize their equipment states, the stored liquid volumes are described with discrete places representing three qualitative values, i.e., 0, 1, and 2. Notice that the dead times of all time-delayed transitions are set to 1 h. Finally, the possibility of sensor failure is excluded in this example, and the sensor measurements are assumed to be identical to the equipment states of tanks 1 and 2.

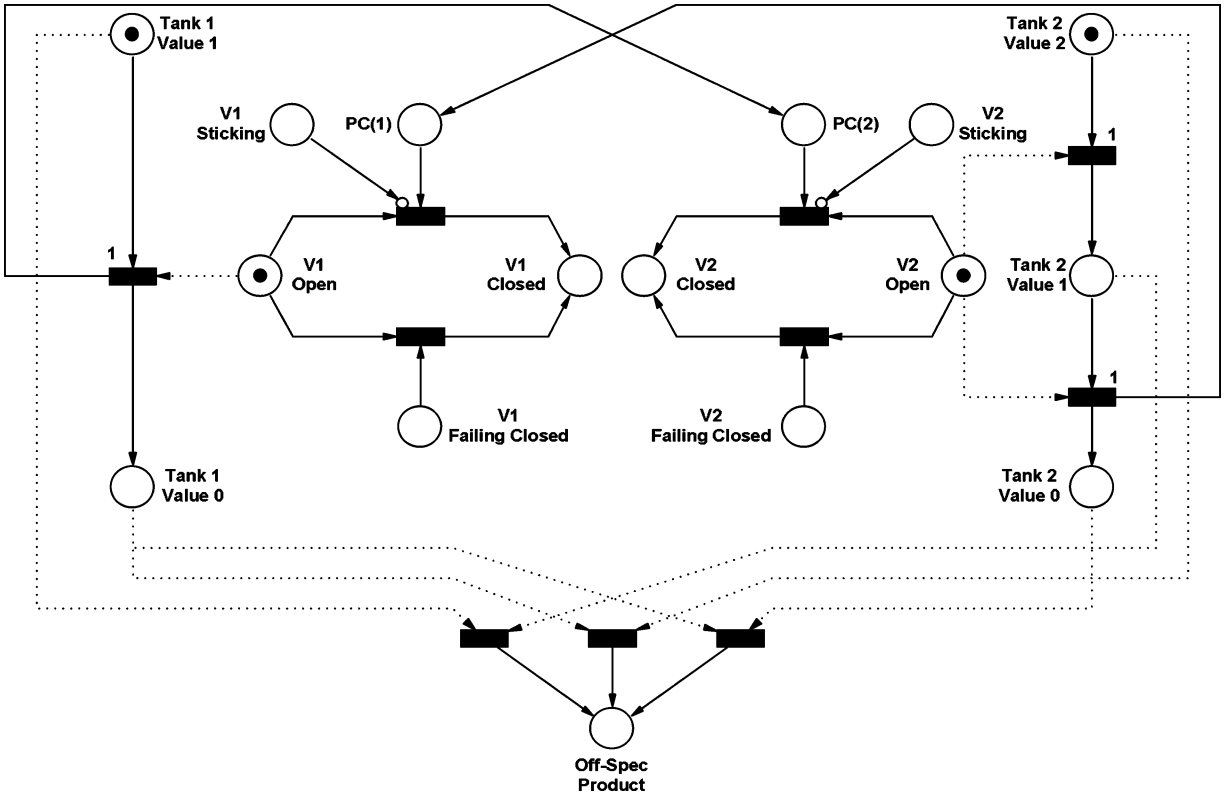

Figure 8. System Petri net of a mixing process.

Table 5. Detailed Simulation Results of Normal Operation for the Mixing Example

place	period 1	period 2	period 3
tank 1 value 1	1	0	0
tank 1 value 0	0	1	1
V1 open	1	1	1
V1 closed	0	0	0
V1 sticking	0	0	0
V1 failing to close	0	0	0
PC(1)	0	0	0
tank 2 value 2	1	0	0
tank 2 value 1	0	1	1
tank 2 value 0	0	0	0
V2 open	1	0	0
V2 closed	0	1	1
V2 sticking	0	0	0
V2 failing to close	0	0	0
PC(2)	0	0	0

In this example, it is assumed that the desired ratio of A/B in the product is 1 and also that the flow rates from the two feed tanks are the same as long as neither of them is empty and both V1 and V2 are open. Thus, it can be deduced that an off-specification product can be produced after the end of operation only if the final volumes in tanks 1 and 2 are not normal. Specifically, the undesirable consequence might be the direct result of the following conditions: (a) The liquid volumes in tanks 1 and 2 are both 1. (b) The liquid volumes in tanks 1 and 2 are 0 and 2, respectively. (c) The liquid volumes in tanks 1 and 2 are both null (value 0).

(iv) Simulating Normal Operation. The equipment states and process conditions of all components at different stages of the normal operation should be simulated by executing the system model. For our example here, let us start from an initial marking given in Figure 8. The results presented in Table 5 can be obtained by firing the enabled transitions sequentially.

(v) Generating the Backward Petri Net. Before converting the system model to the backward Petri net, it is necessary to identify places in the forward net to which further modifications must be introduced. This task can be achieved by comparing the local net configuration of a candidate place with Figure 5a. If a match is confirmed, the FPN model should be modified by attaching one or more substructures to this place according to Figure 5b. The time delays in these substructures should be selected in such a way that failures causing the stalled transition(s) could occur in every discretized time period prior to the period under consideration. Only after all required modifications have been introduced into FPN can the backward Petri net be produced accordingly by following the conversion procedure described in section 3.

From the system model shown in Figure 8, it can be observed that there are five candidate places, i.e., (1) tank 1 value 1, (2) V1 open, (3) tank 2 value 2, (4) tank 2 value 1, and (5) V2 open. This forward net can be modified by attaching five corresponding substructures to these candidate places. Further, because the designated consequence is in period 2 and there is only one period during operation, the time delays in these substructures are all set to be 1 h, i.e., the length of period 1. The resulting BPN can be found in Figure 9.

(vi) Synthesizing the Time-Stamped Fault Trees. The timed-stamped fault tree of a given BPN can be constructed according to algorithm B (see Appendix II). Notice that, in implementing this algorithm, a set of termination conditions is employed (in step 3) to reduce the repetitive effort invested in the deductive reasoning process. These conditions should be checked every time a specific version of the two-layer fault-tree structure is created. A summary of these conditions can be found in Appendix III.

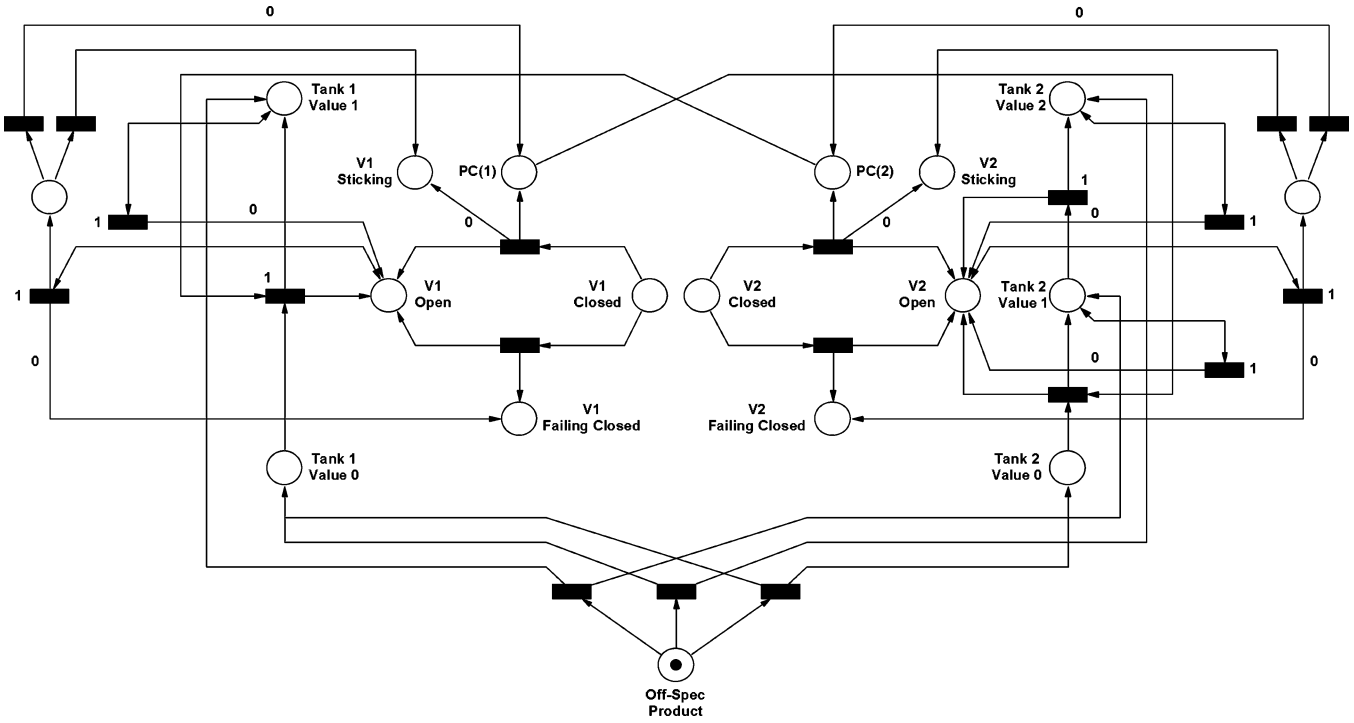

Figure 9. Backward Petri net of a mixing process.

Table 6. Output Data of the Fault-Tree Synthesis Program for the Mixing Process

off-spec product ST = [2]	−	OR	−	AND	−	tank 1 value 1 ST = [2] − tank 2 value 1 ST = [2] (condition 1)
			−	AND	−	tank 1 value 0 ST = [2] (condition 1) − tank 2 value 2 ST = [2]
			−	AND	−	tank 1 value 0 ST = [2] (condition 1) − tank 2 value 0 ST = [2]
tank 1 value 1 ST = [2]	−	OR	−	AND	−	tank 1 value 1 ST = [1] (condition 1) − NOT V1 open ST = [1]
tank 2 value 2 ST = [2]	−	OR	−	AND	−	tank 2 value 2 ST = [1] (condition 1) − NOT V2 open ST = [1]
tank 2 value 0 ST = [2]	−	OR	−	AND	−	tank 2 value 1 ST = [1] − V2 open ST = [1] (condition 1)
NOT V1 open ST = [1]	−	OR	−	AND	−	V1 closed ST = [1]
NOT V2 open ST = [1]	−	OR	−	AND	−	V2 closed ST = [1]
V1 closed ST = [1]	−	OR	−	AND	−	V1 failing to close ST = [1] − V1 open ST = [1] (condition 1)
			−	AND	−	NOT V1 sticking ST = [0] (condition 4) − PC(1) ST = [0−1] (condition 4) − V1 open ST = [0] (condition 4)
V2 closed ST = [1]	−	OR	−	AND	−	V2 failing to close ST = [1] − V2 open ST = [1] (condition 1)
			−	AND	−	NOT V2 sticking ST = [0] (condition 4) − PC(2) ST = [0−1] (condition 4) − V2 open ST = [0] (condition 4)
tank 2 value 1 ST = [1]	−	OR	−	AND	−	tank 2 value 2 ST = [0] (condition 4) − V2 open ST = [0] (condition 4)

The time-stamped fault tree of the mixing process was synthesized automatically with a simple MATLAB program coded on the basis of algorithm B. The data format of its output can be found in Table 6. It is clear that a fault tree can be unambiguously drawn according to such output data (see Figure 10). Notice that a number of branches are severed from the fault tree. For example, the basic events "tanks 2 value 2 in period 0" and "V2 open in period 0" can be removed on the grounds that the occurrence times of these events exceed the given horizon in this work., i.e., the fourth termination condition is satisfied in this case. The other branches are eliminated on the basis of the AND logic.

(vii) Identifying the Minimal Cut Sets. From the fault tree shown in Figure 10, the causes of the top event can be clearly identified, i.e., "V1 failing to close in period 1" and "V2 failing to close in period 1." Notice that the failure "V2 sticking in period 1" is not included in the fault tree. This is reasonable given that the time horizon in the present case ends at the time of the valve-closing action, i.e., 1 h. In other words, the corresponding off-specification product can only be produced after the scheduled completion time of the batch process. If a waiting period (say, 0.5 h) is added to the operatng procedure, then a minimal cut set involving V2 sticking can also be obtained with the proposed fault identification procedure.

8. Application

A realistic example involving an air-drying process is presented here to demonstrate the feasibility of the proposed procedure in practical applications. Essential details of the implementation results are given in the following subsections.

8.1. Process Description. Although a description of the example process can be found in Shaeiwitz et al.,[21] a brief review is provided here for the sake of completeness. Figure 11 is the flow diagram of a sequential process for drying instrument air using two fixed alumina beds, denoted bed I and bed II. Ambient air that contains water vapor enters the process via stream 9 and passes through one of the two beds, where the moisture is adsorbed. The dried air leaves the system in stream 25. To maintain a steady supply of dry air, the above two beds are employed alternately. When one bed is removing moisture from the inlet air, the other is being regenerated and then cooled. Regeneration involves passing hot air through a bed that has been loaded to capacity with water. After the water has been

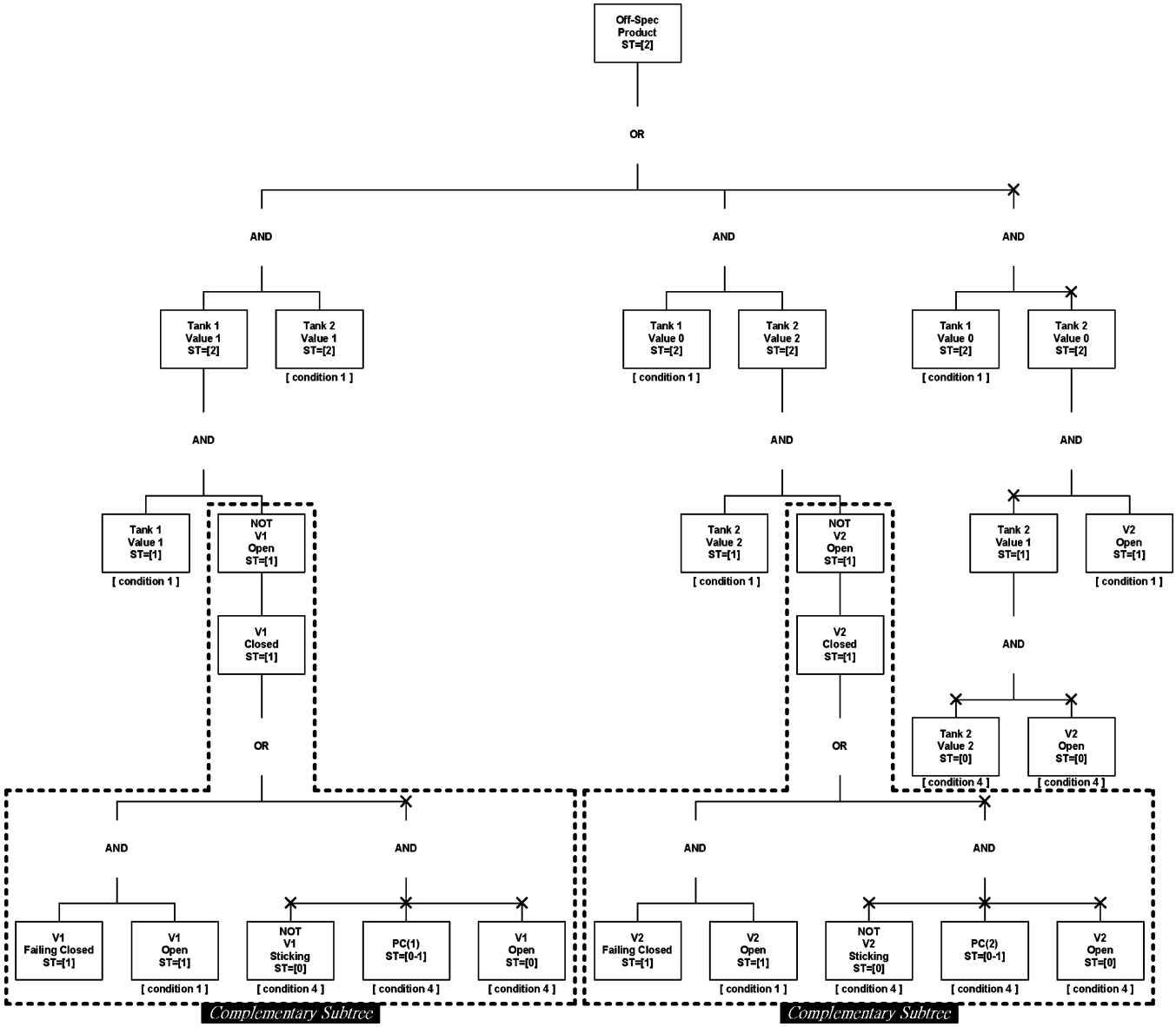

Figure 10. Time-stamped fault tree with the top event occurring in period 2 for a mixing process.

stripped from the alumina, the hot air flows through a condenser where the water vapor can be removed. This air is then recycled via the proportionating valve to the operating bed. The hot regenerated bed is cooled with unheated ambient air before being rotated back into service. The same regeneration and cooling procedures are followed for the other bed when it becomes saturated. The sequential operating steps of this drying process are executed with a timer, a three-way valve (3W), and two four-way valves (4W-I and 4W-II). Both beds experience the same operating cycle. Table 7 gives the detailed recipe applied during normal operation in a complete cycle. For convenience in illustration, it is further assumed that emergency response procedures simply do not exist in this case.

8.2. FPN Model. In this example, the batch process depicted in Figure 11 is modeled with the FPN shown in Figure 12. This model contains components in the first three levels of the hierarchy.

The operating steps specified in a recipe are executed sequentially by a first-level component. In general, each operating step can be characterized in terms of two elementary actions: (1) confirmation of an initiation signal and (2) execution of a command. In this system, the first-level component is the timer, which can be viewed as a device assembled with an internal clock and a controller. The initiation signals are generated by the clock. Four clock states, denoted $P(1)-P(4)$, are described in Figure 12. Each is associated with a time period in the operating cycle. The places PS(i) ($i = 1-4$) can be considered as the clock signals marking the switching times of two successive operating periods. In other words, the placement of a token in the place PS(i) can be used to mark the beginning (or end) of an operating period. It is assumed in this example that the elapsed time of each operating period is the same, namely, 6 h. Hence, the delay time associated with each transition connecting to PS(i) is set to 6 time units. On the other hand, the operating command issued by the controller is always concerned with a change in the equipment state of a second-level component. The place PC(i) ($i = 1-4$) is used to reflect the status of the ith operating command. Notice that the order in which these commands are issued can be arranged according to a given recipe. In this example, two timer failures are used in the system model, i.e., the spurious commands to 4W-I

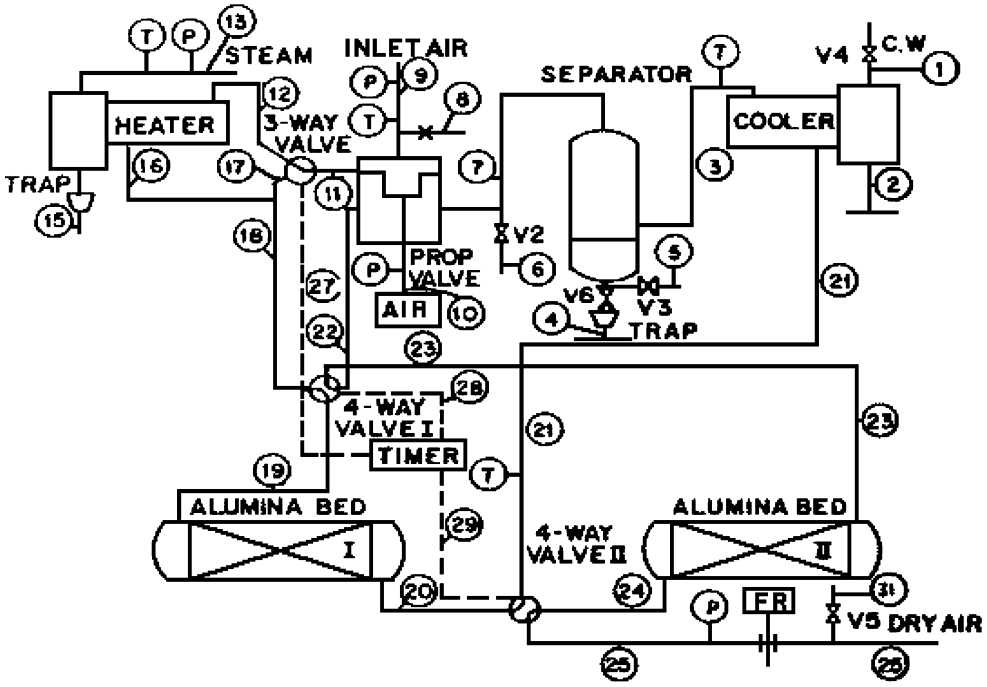

Figure 11. Process flow diagram of a utility air-drying process.

Table 7. Operating Recipe for Fixed-Bed Air Drying Process

time period	valve position			bed status	
	3W	4W-I	4W-II	bed I	bed II
1	11 → 12	18 → 19 22 → 23	20 → 21 24 → 25	regeneration	in service
2	11 → 17	18 → 19 22 → 23	20 → 21 24 → 25	cooling	in service
3	11 → 12	18 → 23 22 → 19	20 → 25 24 → 21	in service	regeneration
4	11 → 17	18 → 23 22 → 19	20 → 25 24 → 21	in service	regeneration

Table 8. Relationships between Valve Positions and Stream Connections

valve	valve position	stream connection
3W	+	11 → 12
	−	11 → 17
4W-I	+	18 → 19 and 22 → 23
	−	18 → 23 and 22 → 19
4W-II	+	20 → 21 and 24 → 25
	−	20 → 25 and 24 → 21

and 4W-II. Notice from Table 7 that both four-way valves are switched every two operating periods when period 1 or 3 begins. It is assumed that erroneous operating commands can be issued by the controller to turn one or both of these valves to the wrong position-(s) at the starting times of period 2 and 4.

The second-level components are the three-way valve and the two four-way valves. The position of 3W determines the route of inlet air flow for regeneration or cooling. The fresh air can either be directed to the heater or simply bypass it. The position of 4W-I defines the connections between the alumina beds and their air supplies. The air consumed in each bed can be taken either from the system inlet or from the lower port of the proportionating valve. The position of 4W-II governs the destinations of the exit air flows from these two beds, i.e., the air can be either discharged or recycled. Every valve in this system can be switched to only two alternative positions denoted by PV(+) and PV(−) in the FPN model. The relationships between the valve positions and the stream connections are shown in Table 8. Notice that the model structures for the three valves in Figure 12 are essentially the same and there is only one failure mode, i.e., valve sticking.

Six third-level components can be readily identified from the P&ID given in Figure 11, i.e., the two alumina beds, the heater, the cooler, the separator, and the proportionating valve. In addition, joint connecting lines 16−18 can also be treated as a level-3 component, i.e., mixer. Aside from the adsorption beds, the conditions of all process units should remain unchanged if none of these units are affected by failures. Without loss of generality, let us limit the scope of fault identification to component failures in levels 1 and 2 only. Thus, all component models in the third level (except those for alumina beds) can be neglected in our analysis. The equipment states of each alumina bed can be characterized in terms of two parameters, i.e., the bed temperature (denoted by T) and the water content (denoted by M). Although these two parameters are continuous, in each case, a set of discretized quantities is employed here to describe the transient behaviors qualitatively. It is assumed in this example that the main factor controlling the bed temperature is the temperature of incoming air. Within one time period, the hot air could increase the bed temperature to an upper bound (value 1), and conversely, the unheated cool air could decrease the temperature to a lower limit (value 0). On the other hand, the water content of an alumina bed in a time period is assumed to be affected by two factors in the previous period, namely, (1) the temperature of the inlet air and (2) the water content in the bed. If the bed is unsaturated (value 0) or half-saturated (value 1), the water content can be increased by the passing of unheated air as long as the bed temperature is at value 0. In the former case, the unsaturated bed can reach the saturation level (value 2) in two time periods or the half-saturation level (value 1) in one. In the latter case, the bed should be saturated in one time period. Because

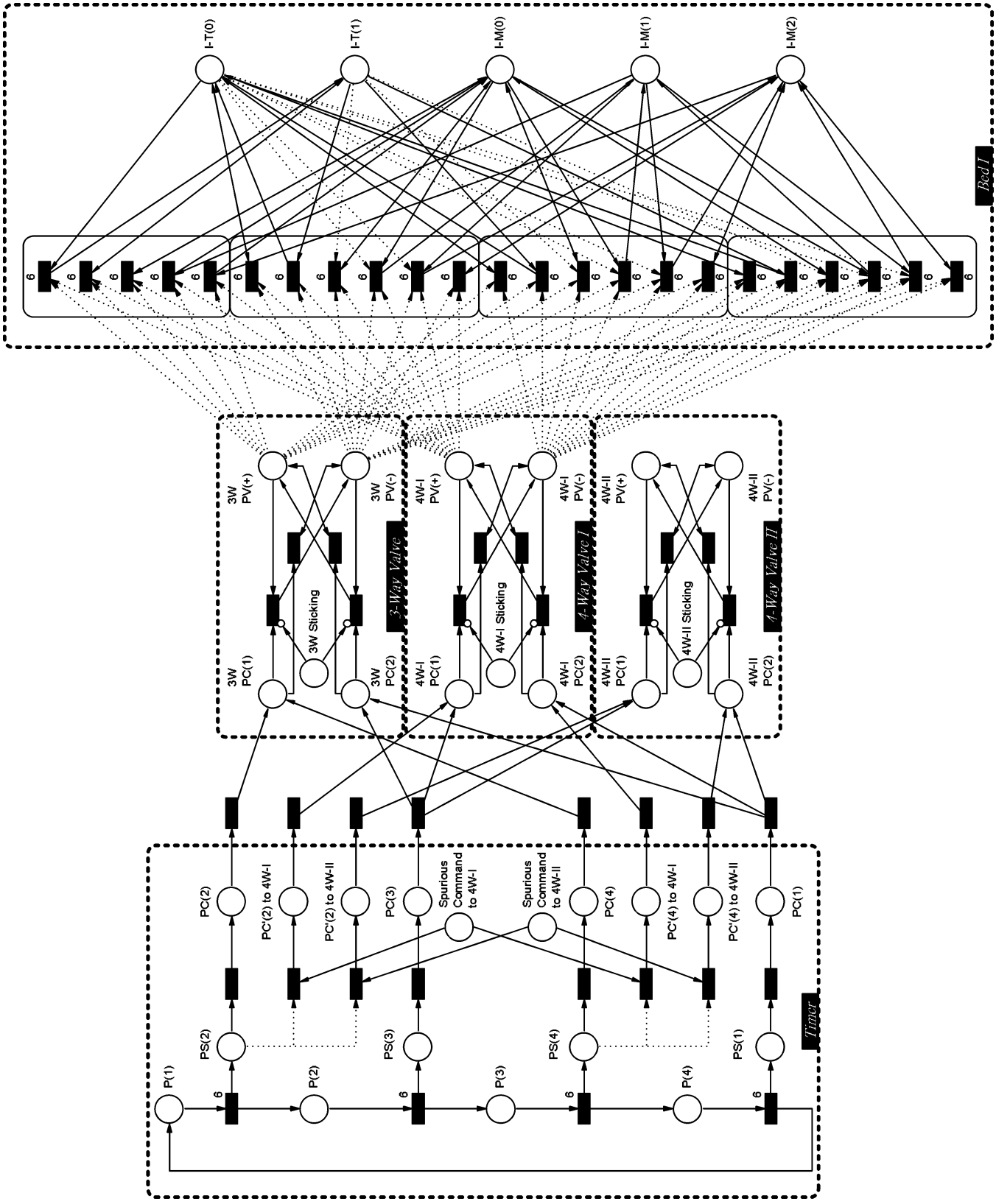

Figure 12. System Petri net of a utility air-drying process.

a saturated bed cannot be used for dehumidification purposes, hot air should then be introduced to strip water from the alumina. Here, it is assumed that the bed can be dried "completely" in one time period. A summary of the complete component model for beds I and II can be found in Table 9. For the sake of clarity, the component model of bed II is eliminated from the FPN in Figure 12. However, this missing part can easily be reconstructed according to the fourth column of Table 9.

The equipment states of a third-level component can be monitored via measurement instruments in the

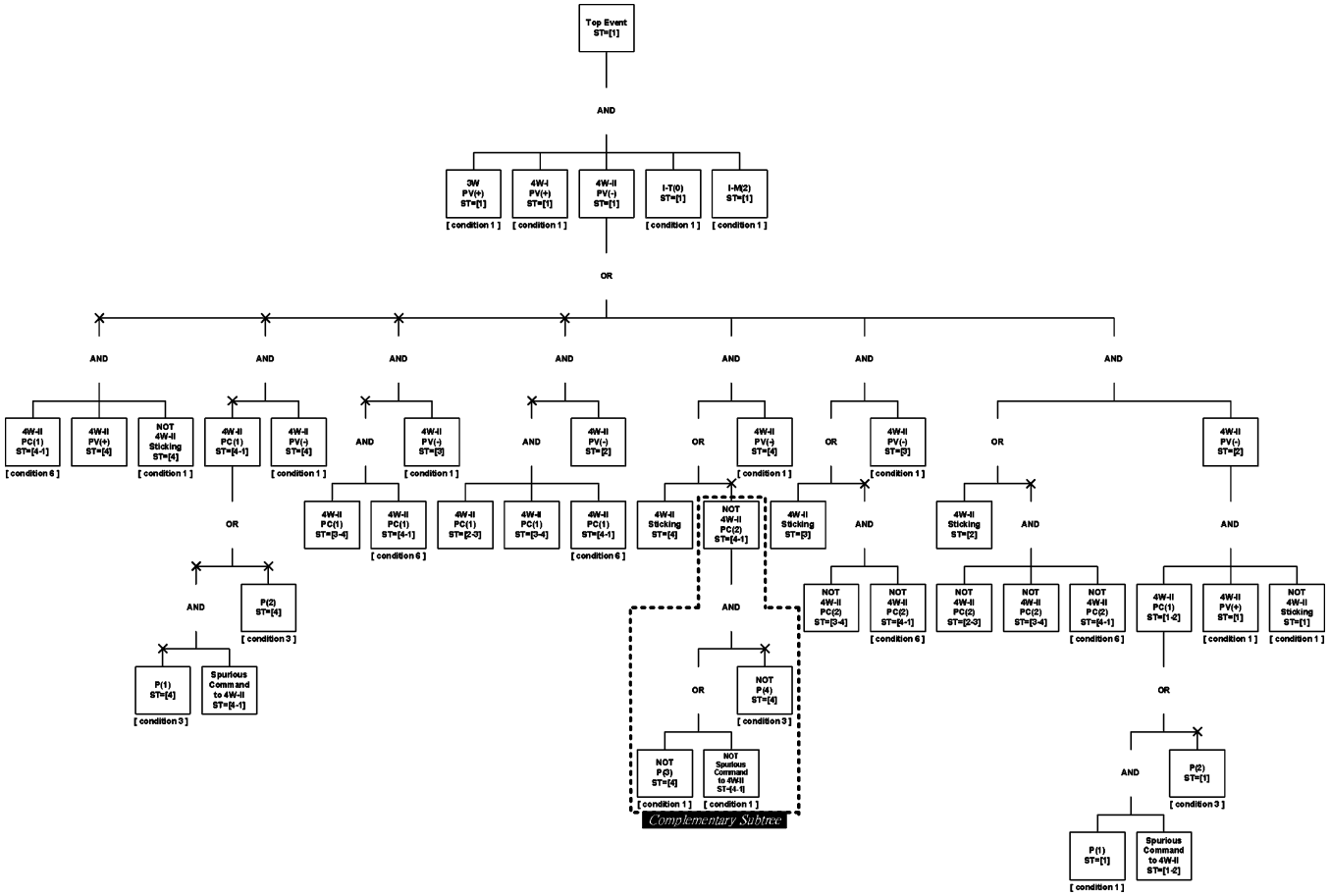

Figure 13. Time-stamped fault tree with the top event occurring in period 1 for a utility air-drying process.

Table 9. Changes in Bed States during One Operating Period

3W	4W-I	bed I	bed II
+	+	I-$T(0) \rightarrow$ I-$T(1)$	II-$T(0) \rightarrow$ II-$T(0)$
		I-$T(1) \rightarrow$ I-$T(1)$	II-$T(1) \rightarrow$ II-$T(0)$
		I-$M(0) \rightarrow$ I-$M(0)$	II-$M(0)$ + II-$T(1) \rightarrow$ II-$M(0)$
			II-$M(0)$ + II-$T(0) \rightarrow$ II-$M(1)$
		I-$M(1) \rightarrow$ I-$M(0)$	II-$M(1)$ + II-$T(0) \rightarrow$ II-$M(2)$
		I-$M(2) \rightarrow$ I-$M(0)$	II-$M(2) \rightarrow$ II-$M(2)$
−	+	I-$T(0) \rightarrow$ I-$T(0)$	II-$T(0) \rightarrow$ II-$T(0)$
		I-$T(1) \rightarrow$ I-$T(0)$	II-$T(1) \rightarrow$ II-$T(0)$
		I-$M(0)$ + I-$T(1) \rightarrow$ I-$M(0)$	II-$M(0)$ + II-$T(1) \rightarrow$ II-$M(0)$
		I-$M(0)$ + I-$T(0) \rightarrow$ I-$M(1)$	II-$M(0)$ + II-$T(0) \rightarrow$ II-$M(1)$
		I-$M(1)$ + I-$T(0) \rightarrow$ I-$M(2)$	II-$M(1)$ + II-$T(0) \rightarrow$ II-$M(2)$
		I-$M(2) \rightarrow$ I-$M(2)$	II-$M(2) \rightarrow$ II-$M(2)$
+	−	I-$T(0) \rightarrow$ I-$T(0)$	II-$T(0) \rightarrow$ II-$T(1)$
		I-$T(1) \rightarrow$ I-$T(0)$	II-$T(1) \rightarrow$ II-$T(1)$
		I-$M(0)$ + I-$T(1) \rightarrow$ I-$M(0)$	II-$M(0) \rightarrow$ II-$M(0)$
		I-$M(0)$ + I-$T(0) \rightarrow$ I-$M(1)$	
		I-$M(1)$ + I-$T(0) \rightarrow$ I-$M(2)$	II-$M(1) \rightarrow$ II-$M(0)$
		I-$M(2) \rightarrow$ I-$M(2)$	II-$M(2) \rightarrow$ II-$M(0)$
−	−	I-$T(0) \rightarrow$ I-$T(0)$	II-$T(0) \rightarrow$ II-$T(0)$
		I-$T(1) \rightarrow$ I-$T(0)$	II-$T(1) \rightarrow$ II-$T(0)$
		I-$M(0)$ + I-$T(1) \rightarrow$ I-$M(0)$	II-$M(0)$ + II-$T(1) \rightarrow$ II-$M(0)$
		I-$M(0)$ + I-$T(0) \rightarrow$ I-$M(1)$	II-$M(0)$ + II-$T(0) \rightarrow$ II-$M(1)$
		I-$M(1)$ + I-$T(0) \rightarrow$ I-$M(2)$	II-$M(1)$ + II-$T(0) \rightarrow$ II-$M(2)$
		I-$M(2) \rightarrow$ I-$M(2)$	II-$M(2) \rightarrow$ II-$M(2)$

fourth level. However, there is no need to discuss the component models of these instruments because they are not used in implementing the operating Period procedure.

8.3. Fault Identification. Let us assume that a possible hazard for fault identification can be described as "H_2O concentration in stream 25 is too high during time period k", where k can be 1, 2, 3, or 4. This is because, if the outlet air contains too much water vapor, a large number of valuable instruments downstream can be damaged. According to the process description, this undesirable consequence could be directly caused by the following conditions: (i) the temperature of the served bed is too high, (ii) the adsorbents in the served bed are saturated, and (iii) the inlet air temperature in the served bed is too high. Notice that each of these conditions can be characterized by a unique set of five different places in the system model. Two of them are used to represent the bed states, and the rest are for the valve states. For example, cause ii can be described with the set {3W PV(+), 4W-I PV(+), 4W-II PV(−), I−$T(0)$, I−$M(2)$}. The combinations of places that result in at least one of the above three sufficient conditions in bed I are listed in Table 10. The same combinations can be obtained for bed II if the valve positions of 4W-I

Table 10. Direct Causes of Undesirable Consequence (Bed I Conditions)

conditions	set no.	places
i	1	{3W PV(+), 4W-I PV(+), 4W-II PV(−), I-$T(1)$, I-$M(0)$}
	2	{3W PV(−), 4W-I PV(+), 4W-II PV(−), I-$T(1)$, I-$M(0)$}
	3	{3W PV(+), 4W-I PV(−), 4W-II PV(−), I-$T(1)$, I-$M(0)$}
	4	{3W PV(−), 4W-I PV(−), 4W-II PV(−), I-$T(1)$, I-$M(0)$}
ii	5	{3W PV(+), 4W-I PV(+), 4W-II PV(−), I-$T(0)$, I-$M(2)$}
	6	{3W PV(−), 4W-I PV(+), 4W-II PV(−), I-$T(0)$, I-$M(2)$}
	7	{3W PV(+), 4W-I PV(−), 4W-II PV(−), I-$T(0)$, I-$M(2)$}
	8	{3W PV(−), 4W-I PV(−), 4W-II PV(−), I-$T(0)$, I-$M(2)$}
iii	9	{3W PV(+), 4W-I PV(+), 4W-II PV(−), I-$T(0)$, I-$M(0)$}
	10	{3W PV(+), 4W-I PV(+), 4W-II PV(−), I-$T(0)$, I-$M(1)$}

Table 11. Root Causes of the Undesirable Consequence Occurring in Period 1 for Bed I

set no.	root causes
1	3W sticking [3] + spurious command to 4W-I [3−4] + 4W-II sticking [4] 3W sticking [1] + spurious command to 4W-I [3−4] + 4W-II sticking [4] 3W sticking [3] + 4W-II sticking [3] + spurious command to 4W-I [3−4] 3W sticking [1] + 4W-II sticking [3] + spurious command to 4W-I [3−4] spurious command to 4W-II [1−2] + 4W-II sticking [2] + 3W sticking [3] + spurious command to 4W-I [3−4] 3W sticking [1] + spurious command to 4W-II [1−2] + 4W-II sticking [2] + spurious command to 4W-I [3−4]
2	ϕ
3	ϕ
4	ϕ
5	4W-II sticking [4] 4W-II sticking [3] spurious command to 4W-II [1−2] + 4W-II sticking [2]
6	4W-II sticking [4] + 3W sticking [4] 4W-II sticking [3] + 3W sticking [4] spurious command to 4W-II [1−2] + 4W-II sticking [2] + 3W sticking [4] 3W sticking [2] + 4W-II sticking [4] 3W sticking [2] + 4W-II sticking [3] spurious command to 4W-II [1−2] + 4W-II sticking [2] + 3W sticking [2]
7	4W-II sticking [4] + 4W-I sticking [4] 4W-II sticking [3] + 4W-I sticking [4] spurious command to 4W-II [1−2] + 4W-II sticking [2] + 4W-I sticking [4] 4W-I sticking [3] + 4W-II sticking [4] 4W-I sticking [3] + 4W-II sticking [3] spurious command to 4W-II [1−2] + 4W-II sticking [2] + 4W-I sticking [3] spurious command to 4W-I [1−2] + 4W-I sticking [2] + 4W-II sticking [4] spurious command to 4W-I [1−2] + 4W-I sticking [2] + 4W-II sticking [3] spurious command to 4W-I [1−2] + spurious command to 4W-II [1−2] + 4W-I sticking [2] + 4W-II sticking [2]
8	4W-II sticking [4] + 4W-I sticking [4] + 3W sticking [4] 4W-II sticking [3] + 4W-I sticking [4] + 3W sticking [4] spurious command to 4W-II [1−2] + 4W-II sticking [2] + 4W-I sticking [4] + 3W sticking [4] 4W-I sticking [3] + 4W-II sticking [4] + 3W sticking [4] 4W-I sticking [3] + 4W-II sticking [3] + 3W sticking [4] spurious command to 4W-II [1−2] + 4W-II sticking [2] + 4W-I sticking [3] + 3W sticking [4] spurious command to 4W-I [1−2] + 4W-I sticking [2] + 4W-II sticking [4] + 3W sticking [4] spurious command to 4W-I [1−2] + 4W-I sticking [2] + 4W-II sticking [3] + 3W sticking [4] spurious command to 4W-I [1−2] + spurious command to 4W-II [1−2] + 4W-I sticking [2] + 4W-II sticking [2] + 3W sticking [4] 3W sticking [2] + 4W-II sticking [4] + 4W-I sticking [4] 3W sticking [2] + 4W-II sticking [3] + 4W-I sticking [4] spurious command to 4W-II [1−2] + 3W sticking [2] + 4W-II sticking [2] + 4W-I sticking [4] 3W sticking [2]+4W-I sticking [3] + 4W-II sticking [4] 3W sticking [2]+4W-I sticking [3] + 4W-II sticking [3] spurious command to 4W-II [1−2] + 3W sticking [2] + 4W-II sticking [2] + 4W-I sticking [3] spurious command to 4W-I [1−2] + 3W sticking [2] + 4W-I sticking [2] + 4W-II sticking [4] spurious command to 4W-I [1−2] + 3W sticking [2] + 4W-I sticking [2] + 4W-II sticking [3] spurious command to 4W-I [1−2] + spurious command to 4W-II [1−2] + 3W sticking [2] + 4W-I sticking [2] + 4W-II sticking [2]
9	ϕ
10	ϕ

Table 12. Root Causes of the Undesirable Consequence Occurring in Period 2 for Bed I

set no.	root causes
1	3W sticking [1] + spurious command to 4W-II [1−2] 4W-II sticking [4] + 3W sticking [1] 4W-II sticking [3] + 3W sticking [1] 3W sticking [3] + spurious command to 4W-II [1−2] 3W sticking [3] + 4W-II sticking [4] 3W sticking [3] + 4W-II sticking [3]
2	spurious command to 4W-II [1−2] 4W-II sticking [3] 4W-II sticking [4]
3	3W sticking [1] + spurious command to 4W-I [1−2] + spurious command to 4W-II [1−2] 4W-II sticking [4] + 3W sticking [1] + spurious command to 4W-I [1−2] 4W-II sticking [3] + 3W sticking [1] + spurious command to 4W-I [1−2] 3W sticking [3] + spurious command to 4W-I [1−2] + spurious command to 4W-II [1−2] 3W sticking [3] + 4W-II sticking [4] + spurious command to 4W-I [1−2] 3W sticking [3] + 4W-II sticking [3] + spurious command to 4W-I [1−2]
4	spurious command to 4W-I [1−2] + spurious command to 4W-II [1−2] 4W-II sticking [4] + spurious command to 4W-I [1−2] 4W-II sticking [3] + spurious command to 4W-I [1−2]
5	ϕ
6	3W sticking [4] + spurious command to 4W-II [1−2] 3W sticking [2] + spurious command to 4W-II [1−2] 4W-II sticking [4] + 3W sticking [4] 4W-II sticking [4] + 3W sticking [2] 4W-II sticking [3] + 3W sticking [4] 4W-II sticking [3] + 3W sticking [2]
7	4W-I sticking [4] + 3W sticking [1] + spurious command to 4W-II [1−2] 4W-I sticking [4] + 4W-II sticking [4] + 3W sticking [1] 4W-II sticking [3] + 4W-I sticking [4] + 3W sticking [1] 4W-I sticking [3] + 3W sticking [1] + spurious command to 4W-II [1−2] 4W-I sticking [3] + 4W-II sticking [4] + 3W sticking [1] 4W-I sticking [3] + 4W-II sticking [3] + 3W sticking [1] 3W sticking [3] + 4W-I sticking [4] + spurious command to 4W-II [1−2] 3W sticking [3] + 4W-I sticking [4] + 4W-II sticking [4] 3W sticking [3] + 4W-II sticking [3] + 4W-I sticking [4] 3W sticking [3] + 4W-I sticking [3] + spurious command to 4W-II [1−2] 3W sticking [3] + 4W-I sticking [3] + 4W-II sticking [4] 3W sticking [3] + 4W-II sticking [3] + 4W-I sticking [3]
8	4W-I sticking [4] + spurious command to 4W-II [1−2] 4W-I sticking [4] + 4W-II sticking [4] 4W-II sticking [3] + 4W-I sticking [4] 4W-I sticking [3] + spurious command to 4W-II [1−2] 4W-I sticking [3] + 4W-II sticking [4] 4W-I sticking [3] + 4W-II sticking [3]
9	ϕ
10	ϕ

and 4W-II in this table are changed to the opposite positions and all prefixes of bed states are replaced by II.

The proposed deductive reasoning procedures can be applied to each of the five places in all possible sets. For example, the time-stamped fault tree corresponding to the set {3W PV(+), 4W-I PV(+), 4W-II PV(−), I−T(0), I−M(2)} occurring in period 1 can be found in Figure 13. The minimal cut sets of this fault tree are (1) {4W-II sticking in period 4}, (2) {4W-II sticking in period 3},

Table 13. Root Causes of the Undesirable Consequence Occurring in Period 3 for Bed I

set no.	root causes
1	3W sticking [1] + 4W-I sticking [1] 3W sticking [1] + 4W-I sticking [2] spurious command to 4W-I [3−4] + 4W-I sticking [4] + 3W sticking [1] 3W sticking [3] + 4W-I sticking [1] 3W sticking [3] + 4W-I sticking [2] 3W sticking [3] + spurious command to 4W-I [3−4] + 4W-I sticking [4]
2	ϕ
3	3W sticking [1] 3W sticking [3]
4	ϕ
5	ϕ
6	3W sticking [4] + 4W-I sticking [1] 3W sticking [4] + 4W-I sticking [2] spurious command to 4W-I [3−4] + 3W sticking [4] + 4W-I sticking [4]
7	4W-I sticking [3] 7 4W-I sticking [4]
8	3W sticking [4] 4W-I sticking [4] + 3W sticking [2] 4W-I sticking [3] + 3W sticking [2]
9	4W-I sticking [1] 4W-I sticking [2] spurious command to 4W-I [3−4] + 4W-I sticking [4]
10	ϕ

Table 14. Root Causes of the Undesirable Consequence Occurring in Period 4 for Bed I

set no.	root causes
1	4W-I sticking [2] + 3W sticking [3] 4W-I sticking [1] + 3W sticking [3] 3W sticking [1] + 4W-I sticking [2] 3W sticking [1] + 4W-I sticking [1]
2	4W-I sticking [2] 4W-I sticking [1]
3	ϕ
4	ϕ
5	ϕ
6	3W sticking [4] + 4W-I sticking [2] 3W sticking [4] + 4W-I sticking [1]
7	4W-I sticking [4] + 3W sticking [3] 4W-I sticking [4] + 3W sticking [1]
8	4W-I sticking [4] 3W sticking [4]
9	3W sticking [1] + spurious command to 4W-I [3−4]
10	3W sticking [3] + spurious command to 4W-I [3−4]

and (3) {4W-II sticking in period 2, spurious command to 4W-II between periods 1 and 2}. All root causes of the designated consequence occurring in periods 1−4 can be identified in a similar fashion, and the results are summarized in Tables 11−14. In these tables, the occurrence periods or instances of the events/conditions are specified in square brackets. It is assumed in this example that the valve-sticking failures can occur in each of the four time periods and that spurious commands can only be issued at instances between consecutive periods. All identified failure mechanisms were validated by executing the FPN model. In addition, each fault propagation pattern can be reasoned manually to verify the correctness of these results. Let us consider the scenarios resulting from the failures "4W-II sticking" occurring in periods 3 and 4 as examples, i.e., the cases associated with rows 16 and 17 in the second column of Table 11. Obviously, if 4W-II sticks in period 3 or 4, the system should still behave normally during periods 3 *and* 4. Notice that the outlet air from bed I is scheduled to be recycled to the cooler in period 1 of the next cycle. However, as a result of the 4W-II failure, this air is discharged to stream 25 instead. Thus, it can be concluded that the conditions in the fifth set of Table 11 can indeed be realized from "4W-II sticking in period 3" or "4W-II sticking in period 4".

As mentioned before, the level-1 and level-2 component failures reported in Tables 11−14 are only the root causes of the abnormal bed I states listed in Table 10. It should be noted that the undesirable consequence "H_2O concentration in stream 25 is too high" can also be attributed to bed II conditions. The corresponding root causes can be easily generated by subtracting two periods from the occurrence periods listed in the second column of Tables 11−14. Let us again consider set 5 in Table 11 as an example to demonstrate this procedure. The set of places causing condition ii in bed II can be easily obtained by following the conversion techniques described previously, i.e., {3W PV(+), 4W-I PV(−), 4W-II PV(+), II-T(0), II-M(2)}. The occurrence period of this set for bed II should be obtained by subtracting 2 from the occurrence period of set 5, i.e., period 3. In the same manner, all occurrence times of the corresponding root causes can be generated from the occurrence periods and instances listed in the second column of Table 11. In other words, the corresponding cut sets are (1) {4W-II sticking in period 1}, (2) {4W-II sticking in period 2}, and (3) {4W-II sticking in period 4, spurious command to 4W-II between periods 3 and 4}.

Finally, notice that the advantages of the proposed Petri-net-based approach can be clearly observed from a comparison between the above results and those obtained with digraphs. Specifically, the fault propagation scenarios identified by Shaeiwitz et al.[21] are limited to only the cases in which the basic and top events occur in the same operating period. In other words, the possibilities of earlier failures causing the designated consequence in a later time period (or cycle) are not considered in the conventional fault-tree analysis. This restriction can be successfully removed with Petri nets according to the approach described in the present work.

9. Conclusions

A systematic deductive reasoning procedure is presented in this paper to identify all possible causes of system hazards in sequential operations. This procedure is carried out on the basis of the backward Petri nets transformed from the original system model. The corresponding fault trees can also be constructed accordingly to represent the deduction process. From the results obtained in the application example, it can be observed that the proposed approach can indeed be used for the design of computer programs to automate the cause-finding operation in hazard analysis.

Acknowledgment

This work is supported by the National Science Council of the ROC government under Grant NSC91-2214-E-006-013.

Appendix I: Algorithm A

1. Let flag = 0.

2. Select a positive entry in the current marking $\mathbf{m}_k$ (say, m_i). IF all positive entries in $\mathbf{m}_k$ have been examined and flag = 0, then stop.

3. Select a column vector of **B** (say, $\mathbf{b}_j$) with a negative entry b_{ij} at the corresponding position. If none can be found and flag = 0, then return to step 2. If none can be found and flag = 1, then go to step 6.

4. If $m_i < -b_{ij}$, then return to step 3. If $m_i \geq -b_{ij}$, then the transition associated with $\mathbf{b}_j$ is enabled. The output places of this transition are those corresponding to the positive entries in $\mathbf{b}_j$. Let flag = 1.

5. Repeat steps 3 snd 4.

6. A firing vector $\mathbf{u}_k$ can be assembled by placing values of 1 in the entries corresponding to the enabled transitions and values of 0 otherwise.

7. Compute the entries of future marking $\mathbf{m}_{k+1}$ according to eq 1. The resulting token numbers in the output places of the fired transitions can be found in $\mathbf{m}_{k+1}$.

Appendix II: Algorithm B

1. Let $k = 0$. Write down the top event of the fault tree with a time stamp. Generate the corresponding marking $\mathbf{m}_k$.

2. Perform algorithm A to determine $\mathbf{m}_{k+1}$. Generate a specific two-layer fault-tree configuration accordingly. Compute the occurrence times of its input events.

3. Check each new *positive* entry of $\mathbf{m}_{k+1}$. Replace those that satisfy the *termination conditions* (see Appendix III) with values of 0. Let $k = k + 1$ and then $\mathbf{m}_f = \mathbf{m}_k$.

4. Repeat steps 2 and 3 until algorithm A is not applicable. The resulting fault tree is referred to as the *main tree*.

5. If all entries in $\mathbf{m}_f$ are integers, then the fault-tree synthesis process can be terminated. Otherwise, go to the next step.

6. Select a value ϵ in $\mathbf{m}_f$. If the corresponding place in the FPN is used to represent the only value of a state or condition, replace it with 1 and go to step 7. Otherwise, replace it with 0 and also replace the entries denoting the other values of the same state/condition with values of 1. Go to step 8.

7. Let $\mathbf{m}_k = \mathbf{m}_f$. Repeat steps 2 and 3 to develop a preliminary subtree. Perform the negation operation on the preliminary subtree to produce the complementary subtree. Go to step 9.

8. Let $\mathbf{m}_k = \mathbf{m}_f$. Use the events associated with the values of 1 in the marking as top events and repeat steps 2 and 3 to develop the corresponding preliminary subtrees. Connect these top events as inputs to an OR gate to build the resulting complementary subtree.

9. Attach the complementary subtree to the main tree.

10. Repeat steps 6 and 9 until all ϵ's have been exhausted.

Appendix III: Termination Conditions

1. Normal Events. A time-stamped input event in the two-layer fault-tree structure might be a condition occurred during normal operation. In this work, it is assumed that a normal state cannot be caused by any combination of faults and/or failures. Consequently, there is no need to construct the fault tree further from this event.

2. Repetitive Events. If multiple events are characterized by the same condition and time stamp, then only one of them should be developed further.

3. Impossible Events. If the occurrence possibility of a specific time-stamped event can be ruled out from the outset, then this event should be removed from the fault tree.

4. Time Stamps Beyond the Horizon. Because failures or faults initiating beyond the time horizon are not considered in this work, input events with time stamps exceeding the given horizon must all be removed from the fault tree.

5. Mutually Exclusive Events. If mutually exclusive events under an AND gate are discovered, the AND gate should be eliminated.

6. Supersets. If the input events under one AND gate comprise a subset of those under another in the two-layer fault-tree structure, the latter AND gate should be removed altogether.

Literature Cited

(1) Lapp, S. A.; Powers, G. J. Computer-aided synthesis of fault-trees. *IEEE Trans. Reliab.* **1977**, *R-26*, 2.

(2) Chang, C. T.; Hwang, H. C. New development of the digraph-based techniques for fault-tree synthesis. *Ind. Eng. Chem. Res.* **1992**, *31*, 1490.

(3) Kumamoto, H.; Henley, E. J. Safety and reliability synthesis of systems with control loops. *AIChE J.* **1979**, *20*, 376.

(4) Kelly, B. E.; Lees, F. P. The propagation of faults in process plants: 1. Modeling of fault propagation. *Reliab. Eng.* **1986**, *16*, 3.

(5) Kelly, B. E.; Lees, F. P. The propagation of faults in process plants: 2. Fault tree synthesis. *Reliab. Eng.* **1986**, *16*, 39.

(6) Vaidhyanathan, R.; Venkatasubramanian, V. HAZOP Expert: An Expert System for Automating HAZOP Analysis. *Process Saf. Prog.* **1996**, *15* (2), 80.

(7) Vaidhyanathan, R.; Venkatasubramanian, V. A Semi-Quantitative Reasoning Methdology for Filtering and Ranking HAZOP Results in HAZOP Expert. *Reliab. Eng. Syst. Saf.* **1996**, *53*, 185.

(8) Kuo, D. H.; Hsu, D. S.; Chang, C. T.; Chen, D. H. Prototype for integrated hazard analysis. *AIChE J.* **1997**, *43*, 3 (6), 1494.

(9) Allen, D. J.; Rao, M. S. M. New Algorithms for the Synthesis and Analysis of Fault Trees. *Ind. Eng. Chem. Fundam.* **1980**, *19*, 79.

(10) Andrews, J. D.; Morgan, J. M. Application of Digraph Method of Fault Tree Construction to Process Plant. *Reliab. Eng.* **1986**, *14*, 85.

(11) Chang, C. T.; Hwang, K. S. Studies on the Digraph-based Approach for Fault-Tree Synthesis 1. The Ratio-Control Systems. *Ind. Eng. Chem. Res.* **1994**, *33*, 1520.

(12) Chang, C. T.; Hsu, D. S.; Hwang, D. M. Studies on the Digraph-based Approach for Fault-Tree Synthesis 2. The Trip Systems. *Ind. Eng. Chem. Res.* **1994**, *33*, 1700.

(13) Srinivasan, R.; Venkatasubramanian, V. Automating HAZOP analysis of batch chemical plants: Part I. The knowledge representation framework. *Comput. Chem. Eng.* **1998**, *22* (9), 1345.

(14) Srinivasan, R.; Venkatasubramanian, V. Automating HAZOP analysis of batch chemical plants: Part II. Algorithms and application. *Comput. Chem. Eng.* **1998**, *22* (9), 1357.

(15) Wang, Y. F.; Wu, J. Y.; Chang, C. T. Automatic Hazard Analysis of Batch Operations with Petri Nets. *Reliab. Eng. Syst. Saf.* **2002**, *76* (1), 91.

(16) Wang, Y. F.; Chang, C. T. A Hierarchical Approach to Construct Petri Nets for Modeling the Fault Propagation Mechanisms in Sequential Operations. *Comput. Chem. Eng.* **2003**, *27* (2), 259.

(17) Peterson, J. L. *Petri Net Theory and the Modeling of Systems*; Prentice Hall: Englewood Cliffs, NJ, 1981.

(18) David, R.; Alla, H. Petri net for modeling of dynamic systems—A survey. *Automatica* **1994**, *30* (2), 175.

(19) Ramchandani, C. *Analysis of Asynchronous Concurrent Systems by Petri Nets*; Project MAC, TR-120; Massachusetts Institute of Technology: Cambridge, MA, 1974.

(20) Murata, T. Petri nets: Properties, analysis and applications. *Proc. IEEE* **1989**, *77* (4), 541.

(21) Shaeiwitz, J. A.; Lapp, S. A.; Powers, G. J. Fault Tree Analysis of Sequential Systems. *Ind. Eng. Chem. Process Des. Dev.* **1977**, *16* (4), 529.

Received for review July 29, 2003
Revised manuscript received March 19, 2004
Accepted March 23, 2004

IE034026L