Liveness-Enforcing Supervision of Bounded Ordinary Petri Nets Using Partial Order Methods

Kevin X. He, *Member, IEEE,* and Michael D. Lemmon

***Abstract*—This paper combines and refines recent results into a systematic way to verify and enforce the liveness of bounded ordinary Petri nets. The approach we propose is based on a partial-order method called *network unfolding*. Network unfolding maps the original Petri net to an acyclic *occurrence net*. A *finite prefix* of the occurrence net is defined to give a compact representation of the original net's reachability graph while preserving the causality between net transitions. A set of transition invariants denoted as *base configurations* is identified in the finite prefix. These base configurations capture all of the *fundamental executions* of the net system, thereby providing a modular way to verify and synthesize supervisory net systems. This paper proves necessary and sufficient conditions that characterize the original net's liveness and the existence of maximally permissive supervisory policies that enforce liveness.**

***Index Terms*—Petri nets, supervisory control, unfolding.**

I. Introduction

An ORDINARY Petri net is *live* if it is possible to reach any transition from any reachable marking. At each reachable marking, a *marking-based* supervisor disables the firing of a selected set of controllable transitions. The marking based supervisor is said to be *liveness-enforcing*, if its supervisory policy ensures that the controlled Petri net is live. If a liveness-enforcing supervisor exists, then we know that there also exists a *maximally permissive* supervisor [3]. A liveness-enforcing supervisor is said to be maximally permissive, if it allows the *supremal controllable sublanguage* [4] that enforces liveness.

A variety of theoretical results [5]–[8], and computational algorithms [9]–[10], have been developed to assess the liveness of certain classes of Petri nets. Most of these results were based on the fact that the liveness of a Petri net is closely related to the satisfiability of some properties on place invariants of the net, namely siphons and traps. A *siphon* is a subset of places once being emptied, will never again obtain new tokens, while a *trap* is a subset of places once marked, will always remain marked. It was shown in [5] and [8] that under certain structural constraints of the net, such as free-choiceness [5] or asymmetric choiceness [8], the liveness property is necessary-and-sufficiently determined by checking the coverability of every siphon at every reachable marking.

Previous results on the existence and construction of liveness-enforcing marking based supervisors were presented by Sreenivas [11], [12]. The verification test and the search for liveness-enforcing supervisors in these papers used the *KM-tree* [13] of the original Petri net. The KM-tree of a given Petri net is basically a reachability graph. It was proven in [11] that the existence of liveness-enforcing supervisors is undecidable for arbitrary Petri nets. However, for bounded Petri nets or Petri nets without uncontrollable transitions, the supervisor's existence was proven to be decidable. The major drawback of these results is the low computational efficiency. This drawback is partially due to the poor scalability of the reachability graph. Furthermore, these results overlooked the *causal relationship* among transitions. Transition t_1 is said to be in the *cause* of transition t_2, if t_1 either precedes or equals t_2. Causal relationships are important when dealing with nets with uncontrollable transitions, because these relationships help identify which transition in the cause of an uncontrollable transition t can be used to disable t while preserving the net's live behavior. Overlooking causal relationships is a major reason for these results' high computational complexity when dealing with nets with uncontrollable transitions.

Network unfolding originated from the notion of a *branching process* that was presented in [14]. A branching process unfolds a Petri net into an acyclic structure called an *occurrence net*. Occurrence nets were used to provide a concurrence semantics to nets [15]. McMillan in [15] used unfolding to avoid the state explosion problem in the verification of asynchronous circuits modeled by Petri nets. It was shown in [15] that network unfolding avoids enumerating the arbitrary interleaving of concurrent transitions and thus provides a compact way of describing the net system's state space. A *cut* (or restriction) of the occurrence net was presented in [15] and it was proven that the *finite prefix* resulting from this cut of the occurrence net enumerates all the reachable markings of the original net system. Extensions of McMillan's work to model checking were found in [16] and [17]. In [16], algorithms were developed to verify the reachability of a marking or the liveness of transitions for 1-safe Petri nets, while in [17], other properties such as boundedness and persistence were verified for a general class of nets.

This paper uses network unfolding to provide a systematic way to achieve liveness verification and synthesis of liveness-enforcing supervisors. Our work is based on a new finite prefix that is composed of the finite prefix defined in [15] and the first *tier* of *cutoff* transitions. The intuition behind our method is that while the finite prefix in [15] provides a compact representation

Manuscript received September 6, 2000; revised May 23, 2001 and October 5, 2001. Recommended by Associate Editor R. S. Sreenivas. This work was supported in part by the Army Research Office under Grant DAAD19-01-1-0743, the National Science Foundation under Grant NSF-ECS99-86917, and DARPA/ITO-NEST under Grant AF-F30602-01-2-0526.

K. X. He is with Pennie & Edmonds LLP, New York, NY 10036 USA (e-mail: hek@pennie.com).

M. D. Lemmon is with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556 USA (e-mail: lemmon@nd.edu).

Publisher Item Identifier 10.1109/TAC.2002.800641.

of the state space, the inclusion of cutoff transitions helps identify concurrent and causally related *base configurations*. Each base configuration is considered as a *fundamental execution* of the net system. These fundamental executions provide an efficient and modular way to analyze net behavior since the language generated from the net system can be described by the interleaving of fundamental executions. In particular, this paper shows that the liveness of a bounded ordinary Petri net can be verified by examining the interaction between fundamental executions. Furthermore, if a certain interaction violates liveness, a maximally permissive liveness enforcing supervisor can be developed to control-disable the undesirable interaction at its *critical marking*.

The remainder of this paper is organized as follows. Section II reviews definitions and concepts related to the supervisory control of Petri nets. Section III summarizes results related to network unfolding. Section IV presents results that verify the liveness of bounded ordinary Petri nets based on network unfolding. Section V extends the results in Section IV to develop maximally permissive liveness-enforcing marking based supervisors. Finally, Section VI concludes with directions of future research.

II. Supervisory Control of Petri Nets

This section reviews the definition of ordinary Petri nets and states the supervisory control problem. For more details on Petri nets, refer to [18], [19], and [20]. For more details on supervisory control refer to [3], [4], and [21]–[23].

A. Petri Nets

A Petri net $\mathcal{N}$ is represented by the 4-tuple, (S,T,F,W) where S is the set of *places*, T is the set of *transitions*, $F \subset (S\times T)\cup(T\times S)$ is a set of input arcs (from places to transitions) and output arcs (from transitions to places), and $W: F \to N$ is a mapping that assigns each arc in F a positive integer called a *weight*. A Petri net is called *ordinary* if the weight is 1 for all arcs in F. Since the weight is the same for all arcs, we drop the W and represent an ordinary Petri net as the 3-tuple, $\mathcal{N} = (S,T,F)$. In the sequel, we assume that the Petri net is ordinary.

We denote the *preset* of a transition $t \in T$ as $\bullet t$ and define it as the set of places, $s \in S$ such that $(s,t) \in F$. In a dual manner, we introduce the postset of a transition $t \in T$ as $t\bullet$ and define it as the set of places, $s \in S$ such that $(t,s) \in F$. We define presets and postsets of places in a similar way.

Let T_1 be a set of transitions of $\mathcal{N}$, we define the preset of T_1 as

$$\bullet T_1 = \{s \in S | s \in \bullet t \text{ for some } t \in T_1 \text{ and } s \notin t'\bullet, \text{ for all } t' \in T_1\}.$$

In a dual manner, we define the postset of T_1 as

$$T_1\bullet = \{s \in S | s \in t\bullet \text{ for some } t \in T_1 \text{ and } s \notin \bullet t', \text{ for all } t' \in T_1\}.$$

For example, in Fig. 1, if we take $T_1 = \{t_5, t_4\}$, then $\bullet T_1 = \{s_5, s_7\}$ and $T_1\bullet = \{s_4, s_8\}$.

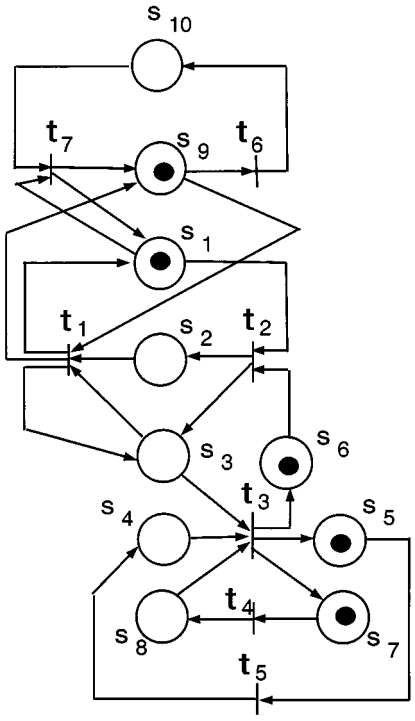

Fig. 1. An example net.

Note that the definition of $\bullet T_1$ ($T_1\bullet$) in this paper is different from its traditional definition $\bullet T_1 = \cup_{t\in T_1} \bullet t$ ($T_1\bullet = \cup_{t\in T_1} t\bullet$). The relationship between the new definition $(\bullet T_1)^{\text{new}}$ and the traditional one $(\bullet T_1)^{\text{old}}$ is that $(\bullet T_1)^{\text{new}} = (\bullet T_1)^{\text{old}} - (T_1\bullet)^{\text{old}}$ ($(T_1\bullet)^{\text{new}} = (T_1\bullet)^{\text{old}} - (\bullet T_1)^{\text{old}}$).

The current "state" of the Petri net is represented by the *marking* of the network. The marking $\mu: S \to Z^+$ is a mapping from the places onto nonnegative integers. The marking $\mu(s)$ of place s denotes the number of *tokens* in that place. Graphically, we represent places by empty circles, transitions by bars and tokens by small filled circles. Fig. 1 shows an example Petri net with initial marking $\mu_0 = [\mu_0(s_1)\,\mu_0(s_2), \ldots, \mu_0(s_{10})]^T = [1\,000\,111\,010]^T$.

The dynamics of ordinary Petri nets are characterized by the way in which the network marking evolves. We say that the transition t is *enabled* if $\mu(s) > 0$ for all $s \in \bullet t$. An enabled transition may *fire*. We introduce a firing function $q: T \to \{0,1\}$ such that $q(t) = 1$ if t is firing and is zero otherwise. If $\mu(s)$ and $\mu'(s)$ denote the marking of place p before and after the firing of enabled transition t, denoted by $\mu \xrightarrow{t} \mu'$, then

$$\mu'(s) = \begin{cases} \mu(s)+1 & \text{if } s \in t\bullet \setminus \bullet t \\ \mu(s)-1 & \text{if } s \in \bullet t \setminus t\bullet \\ \mu(s) & \text{otherwise.} \end{cases} \quad (1)$$

We define a *net system* Σ as the pair $(\mathcal{N}, \mu_0)$, where $\mathcal{N} = (S,T,F)$ is a Petri net and μ_0 is its initial marking. We say a sequence of transitions $\sigma = t_1 t_2 \ldots t_n$ is an *occurrence sequence*, if there exist markings $\mu_1, \mu_2, \ldots, \mu_n$ such that

$$\mu_0 \xrightarrow{t_1} \mu_1 \xrightarrow{t_2} \ldots \mu_{n-1} \xrightarrow{t_n} \mu_n$$

μ_n is the marking reached by the occurrence of σ, also denoted by $\mu_0 \xrightarrow{\sigma} \mu_n$. Given two markings μ_1 and μ_2, we say μ_2 is *reach-*

able from μ_1, if there exists an occurrence sequence σ' such that $\mu_1 \xrightarrow{\sigma'} \mu_2$. The *reachability graph* of network $\mathcal{N}$ is a labeled graph having the set of reachable markings of $\mathcal{N}$ as nodes and the relations $\xrightarrow{\sigma}$ between markings as edges.

We define $R(\mu_0)$ as the set of markings reachable from μ_0. We say a net system $(\mathcal{N}, \mu_0)$ is n-*safe* or *bounded*, if there exists a finite number n such that $\mu(s) \leq n, \forall\, s \in S, \forall\, \mu \in R(\mu_0)$, i.e., there exists no place that contains more than n tokens at any reachable marking. We say a net system is 1-safe, if $n = 1$. The net system in Fig. 1 is 1-safe. In the sequel, we assume the net system is bounded.

We say a transition t is reachable from a marking μ if there exists a marking μ' and an occurrence sequence σ' such that $\mu \xrightarrow{\sigma} \mu'$ and μ enables t. We say a place s is reachable from a marking μ if there exists a transition t such that t is reachable from μ and $s \in t\bullet$. Moreover, we say a set of transitions T_1 is reachable from a marking μ if every transition $t \in T_1$ is reachable from μ. We say a set of places S_1 is reachable from a marking μ if every place $s \in S_1$ is reachable from μ.

We denote a sequence of arcs $(x_1, x_2), (x_2, x_3), \ldots, (x_{N-2}, x_{N-1}), (x_{N-1}, x_N)$ in the net $\mathcal{N}$ as a *path*. We say that the net is *acyclic* if there is no path such that $x_1 = x_N$.

Let $\mathcal{N} = (S, T, F)$ be an acyclic net and $x_1, x_2 \in S \cup T$ be two *nodes* of $\mathcal{N}$. We define the *ordering relations* between x_1 and x_2 in the following way.

- We say x_1 *precedes* x_2, denoted as $x_1 < x_2$, if and only if there exists a sequence, σ, of arcs (also called a *path*) of the form

$$\sigma = (y_1, y_2), (y_2, y_3), \ldots, (y_j, y_{j+1}), (y_{j+1}, y_{j+2}), \ldots, (y_{N-1}, y_N) \tag{2}$$

  such that $y_1 = x_1$ and $y_N = x_2$. In Fig. 2, we can see that transition t_2 precedes transition t_1, since there exists the path $(t_2, s_2)(s_2, t_1)$.
- We say x_1 and x_2 are in *conflict*, denoted by $x_1 \# x_2$, if there exist distinct transitions $t_1, t_2 \in T$ such that $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ and $t_i < x_i$ for $i = 1, 2$. We say a node x is in *self conflict* if $x \# x$. This means that there is a node y preceding x such that x can be reached by more than one distinct occurrence sequence from y. In Fig. 2, transitions t_1, t_7 are in conflict since $t_2 < t_1$ and $\bullet t_2 \cap \bullet t_7 = \{s_1\}$.
- We say x_1 and x_2 are *concurrent*, denoted by $x_1 \| x_2$, if they are not in precedence and not in conflict. In Fig. 2, transitions t_4 and t_5 are concurrent.

A Petri net is said to be *finitary* if every node is preceded by a finite number of nodes.

A net system is said to be *deadlock free* if every reachable marking enables at least one transition. A net system is said to be *live* if for any reachable marking μ and any $t \in T$ there exists a marking μ_t reachable from μ and μ_t enables t.

B. Supervisory Control of Petri Nets

We define a *net supervisor* as a mapping $\mathcal{S}: R(\Sigma) \times T \rightarrow \{0, 1\}$, where $R(\Sigma)$ is the set of all reachable markings for the net system Σ. A transition t is said to be control enabled (control disabled) at a marking μ, if $\mathcal{S}(\mu, t) = 1$ ($\mathcal{S}(\mu, t) = 0$). A

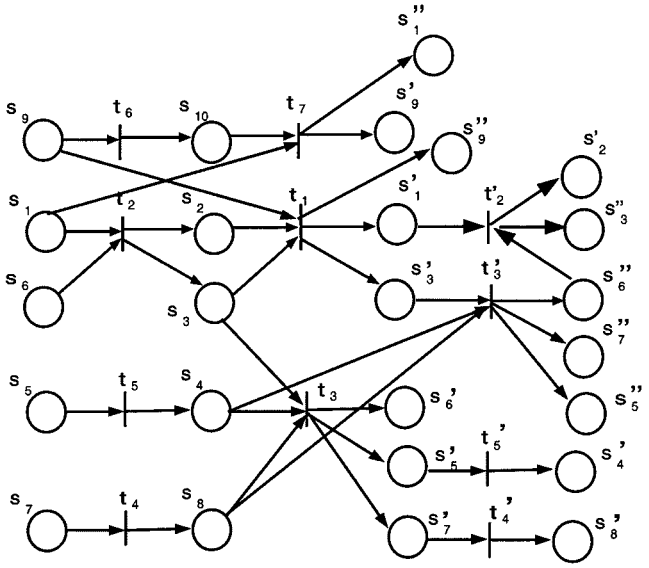

Fig. 2. An occurrence net.

supervised net system $\Sigma|\mathcal{S}$ is a net system in which only control-enabled transitions can fire.

Assume that the transitions of net system Σ can be partitioned into a set of *controllable*, T_c, and *uncontrollable* transitions, T_u. An uncontrollable transition in the controlled system Σ_c, is a transition that cannot be control disabled by the net supervisor. A controllable transition is a transition that is not uncontrollable. An admissible supervisor is one in which $\mathcal{S}(\mu, t) = 1$ for all $t \in T_u$ and any reachable marking μ of the supervised net system.

Consider a net system Σ and let T^* represent all finite length sequences of transitions in T. The formal language $L(\Sigma) \subset T^*$ is said to be the accepted language of Σ if and only if a string $\sigma \in L(\Sigma)$ is an occurrence sequence of Σ. In a similar way, the formal language accepted by the supervised net system $\Sigma|\mathcal{S}$ is denoted as $L(\Sigma|\mathcal{S})$. If $\mathcal{S}$ is an admissible supervisor then $L(\Sigma|\mathcal{S})$ is called a controllable sublanguage.

Consider a net system Σ with accepted language $L(\Sigma)$. We identify a set $R_f(\Sigma)$ of *forbidden markings* which is a subset of $R(\Sigma)$. The language accepted by a supervised net system $L(\Sigma|\mathcal{S})$ is said to be *legal* if there is no occurrence sequence $\sigma \in L(\Sigma|\mathcal{S})$ such that $\mu_0 \xrightarrow{\sigma} \mu_f$ where $\mu_f \in R_f(\Sigma)$. For a given set of forbidden markings $R_f(\Sigma)$, there may be many *legal* controllable sublanguages. The largest such language that contains all other legal controllable sublanguages is called the *supremal controllable sublanguage*. We say $\mathcal{S}$ is *maximally permissive*, if $\Sigma|\mathcal{S}$ accepts the supremal controllable language of $R_f(\Sigma)$. The objective in supervisory control synthesis is to find the admissible maximally permissive supervisor $\mathcal{S}$. Furthermore, since a bounded ordinary Petri net can be represented as a finite state machine, the results of [3] can be used to infer that the supremal controllable sublanguage always exists.

III. Network Unfolding

This paper uses a Petri net analysis method known as network unfolding [15], [16]. Unfolding is a partial-order method that

identifies collections of causally dependent conflict free transitions. This section reviews basic concepts and results about network unfolding [15], [16].

Given a network $\mathcal{N} = (S, T, F)$, let $\min(\mathcal{N})$ denote the set of places

$$\min(\mathcal{N}) = \{s \in S : {\bullet}s = \emptyset\}. \tag{3}$$

An *occurrence net* is a finitary acyclic net $\mathcal{N}' = (S', T', F')$ with initial marking μ_0 such that

1) $|{\bullet}s| \leq 1$ for every $s \in S'$;
2) no transition $t \in T$ is in self conflict;
3) $\mu_0(s) = 1$ if and only if $s \in \min(\mathcal{N}')$.

The acyclic net in Fig. 2 shows an example occurrence network.

Let $\mathcal{N}_1 = (S_1, T_1, F_1)$ and $\mathcal{N}_2 = (S_2, T_2, F_2)$ be two nets with initial markings μ_{01} and μ_{02}, respectively. A *net homomorphism*, $h: S_1 \cup T_1 \to S_2 \cup T_2$ is a mapping between nodes of $\mathcal{N}_1$ and $\mathcal{N}_2$ such that the following hold.

1) $h(S_1) \subseteq S_2$ and $h(T_1) \subseteq T_2$.
2) For every $t \in T_1$, the restriction of h to ${\bullet}t$ is a bijection between ${\bullet}t$ (in $\mathcal{N}_1$) and ${\bullet}h(t)$ (in $\mathcal{N}_2$). Similarly for the postsets $t{\bullet}$ and $h(t){\bullet}$. In other words, a net homomorphism preserves the preset and postset of transitions.
3) The restriction of h to $\min(N_1)$ is a bijection between $\min(\mathcal{N}_1)$ and $\min(\mathcal{N}_2)$. In other words, a net homomorphism also preserves the initial marking.

A *branching process* β of a net system $\Sigma = (\mathcal{N}, \mu_0)$ is a pair $\beta = (\mathcal{N}', h)$ such that

1) $\mathcal{N}' = (S', T', F')$ is an occurrence network;
2) h is a net homomorphism from $\mathcal{N}'$ to $\mathcal{N}$ such that if ${\bullet}t_1 = {\bullet}t_2$ and $h(t_1) = h(t_2)$, then $t_1 = t_2$.

Two branching processes $\beta_1 = (\mathcal{N}_1, h_1)$ and $\beta_2 = (\mathcal{N}_2, h_2)$ are said to be *isomorphic* if there is a bijective homomorphism h between them such that $h_2 \circ h = h_1$. This means $\mathcal{N}_1$ and $\mathcal{N}_2$ are different only in the labels (names) of their nodes and arcs. Given two occurrence nets $\mathcal{N}_1 = (S_1, T_1, F_1)$ and $\mathcal{N}_2 = (S_2, T_2, F_2)$, we say $\mathcal{N}_1$ *contains* $\mathcal{N}_2$, denoted as $\mathcal{N}_2 \subseteq \mathcal{N}_1$, if $S_2 \subseteq S_1, T_2 \subseteq T_1$ and for all $t \in T_2$, ${\bullet}t$ is the same in $\mathcal{N}_2$ as in $\mathcal{N}_1$, so is $t{\bullet}$. We say a branching process $\beta_1 = (\mathcal{N}_1, h_1)$ *contains* $\beta_2 = (\mathcal{N}_2, h_2)$ if $\mathcal{N}_2 \subseteq \mathcal{N}_1$. A branching process is *maximal* if it contains all other branching processes of a network $\mathcal{N}$. An *unfolding* of a net system Σ is the maximal branching process of Σ.

Let $\mathcal{N}' = (S', T', F')$ be the occurrence net obtained from the branching process of Σ. A set of transitions $C \subseteq T$ is a *configuration*, if the following holds:

1) if $t \in C$, then $t' < t$ implies $t' \in C$;
2) no two elements in C are in conflict.

Fig. 2 shows an occurrence net obtained in the unfolding of the network in Fig. 1. In the occurrence net, transitions $\{t_6, t_7\}$ form a configuration.

From the definition, we can see that if a transition t is in a configuration, then all the transitions preceding t should be in the same configuration. In addition, transitions that are concurrent with t can also be included in that configuration. Let $t \in T'$ be a transition of $\mathcal{N}'$, we denote $[t]$ as the set $\{t' \in T' | t' < t \text{ or } t' = t\}$. We call $[t]$ the *cause* of t. This notion of *cause* is the same as the notion of *local configuration* in [15]. For example, in Fig. 2, $[t_7] = \{t_6, t_7\}$. The following two lemmas describe some characteristics of $[t]$ and the configuration.

Lemma 1: For any transition t of an occurrence net $\mathcal{N}'$, $[t]$ forms a configuration.

Proof: First, it is clear that all transitions in $[t]$ satisfy the first condition in the definition of configuration. Second, there must not exist two transitions in conflict in $[t]$ since otherwise t is in self conflict. The lemma is therefore proved. •

Lemma 2: For a configuration C, if transition $t \in C$, then $[t] \subseteq C$.

Proof: Directly follows the definition of configuration. •

Lemma 3: A set of transitions C of $\mathcal{N}'$ forms a configuration, if and only if there exists a set of concurrent transitions T_c of $\mathcal{N}'$ such that $C = \cup_{t \in T_c}[t]$. Furthermore, the set T_c is unique for configuration C.

Proof:

Sufficiency: Let t_1 be a transition in C. Then there must exist a transition $t \in T_c$ such that $t_1 \in [t]$, and therefore, $[t_1] \subseteq [t] \subseteq C$. This means if $t \in C$ then $t' \in C, \forall\, t' < t$. Moreover, there must not exist two transitions t_1, t_2 in C such that $t_1 \# t_2$, since otherwise there must exist $t_1', t_2' \in T_c, t_1 < t_1', t_2 < t_2'$ and $t_1' \# t_2'$.

Necessity: Consider the set of transitions $T_c = \{t \in C | \nexists t' \in C, t < t'\}$. First, following the definition of configuration, all transitions in this set are concurrent. Second, any transition preceding a transition in T_c must also be in C, which means $\cup_{t \in T_c}[t] \subseteq C$. Finally, for any transition $t_1 \in C$, either $t_1 \in T_c$, or there exists a $t' \in T_c$ such that $t_1 < t'$. This means $t_1 \in [t']$, $t_1 \in \cup_{t \in T_c}[t]$ and therefore $C \subseteq \cup_{t \in T_c}[t]$. We thus conclude that T_c is the set of transitions satisfying the necessary condition.

Finally, we need to prove that the T_c defined in the necessity proof is a minimal set satisfying $C = \cup_{t \in T_c}[t]$ and this minimal set T_c is unique for configuration C. Supposing there is another set T_c' such that $C = \cup_{t' \in T_c'}[t']$, then for every $t' \in T_c$, there should be a $t \in T_c$ such that $t' < t$ or $t' = t$. We thus have $[t'] \subseteq [t]$ and $\cup_{t' \in T_c'}[t'] \subseteq \cup_{t \in T_c}[t]$. $\cup_{t' \in T_c'}[t'] = \cup_{t \in T_c}[t]$ holds only if there is a subset of T_c' equal to T_c. This means T_c is a subset of any other set T_c' satisfying $C = \cup_{t \in T_c'}[t]$, which is enough to prove that T_c is minimal and unique. •

Definition: Let C be a configuration, we denoted any transition in $T_c = \{t \in C | \nexists t' \in C, t < t'\}$ as an *end transition* of C.

We define the *cut* of a configuration C as $Cut(C) = (\min(\mathcal{N}') \cup C{\bullet}) \setminus {\bullet}C$. *Cut*s of configurations are used to represent reachable markings of the original net system. The link between *Cut*s and reachable markings can be described in the following way. Let C^m be the set of *Cut*s in the occurrence net $\beta = (\mathcal{N}', h)$. Let S be the set of places in the original net. Let $M: h(C^m) \to N^{|S|}$ be a mapping such that for all $Cut(C)$ in C^m, $M(h(Cut(C)))_i = |\{p \in Cut(C) | h(p) = s_i\}|$. In other words, the ith element of the vector $M(Cut(C)$ is the number of copies of place s_i in $Cut(C)$. It is has been proven [16] that given a configuration C, $M(h(Cut(C)))$ is a reachable marking of the original

net system. For example, in Fig. 2, $Cut(\{t_6, t_7\}) = \{s_1, s_5, s_6, s_7, s_9\} \cup \{s_1'', s_9'\} \backslash \{s_1, s_9\} = \{s_1'', s_5, s_6, s_7, s_9'\}$, and $h(\{s_1'', s_5, s_6, s_7, s_9'\}) = \{s_1, s_5, s_6, s_7, s_9\}$. We have $\mu = M(h(\{s_1'', s_5, s_6, s_7, s_9'\})) = M(\{s_1, s_5, s_6, s_7, s_9\}) = [1\,000\,111\,010]^T$. In other words, μ_0 is reachable from μ_0 after firing t_6, t_7.

An unfolding may be infinite, and therefore it makes sense to define a *finite prefix* of it for verification purposes. A branching process $\beta = (\mathcal{N}', h')$ is a *prefix* of the unfolding $\beta_m = (\mathcal{N}_m, h_m)$ if $\mathcal{N}'$ is contained by $\mathcal{N}_m$.

Definition: A transition t of the unfolding β_m is called a *cutoff* transition if there exists a smaller cause $[t'] \subset [t]$ such that $h_m(\mathrm{Cut}([t'])) = h_m(\mathrm{Cut}([t]))$, or $M(h_m(\mathrm{Cut}([t]))) = \mu_0$. A cutoff transition t of β_m is called a *post cutoff* transition, if $\forall t' \in \bullet(\bullet t), t'$ is a cutoff transition, i.e., t's immediate predecessors are all cutoff transitions.

Remark: The relationship between β_c and the finite prefix β_f defined in [15] can be illustrated as follows. Recall that $\beta_f = (\mathcal{N}_f, h_f)$ is obtained after removing all cutoff transitions from β_m. It is therefore easily seen that β_c contains β_f and the first "tier" of cutoff transitions in β_m. The inclusion of cutoff transitions allows us to find transition cycles (or transition invariants) which are important for the verification of liveness. (Details will appear later). Some initial effort that attempt to include cutoff transitions in the finite prefix to verify net properties can be found in [24].

Fig. 2 shows the finite prefix β_c of the unfolding of the net in Fig. 1. Note that in Fig. 2, transitions t_7, t_2', t_3' are cutoff transitions and β_f is obtained by removing t_7, t_2', t_3' from β_c.

In β_c, we say that a transition t is an *end transition*, if there is no transition t' such that $t < t'$. Note that an end transition is either a cutoff transition, or a *deadlocked* transition that precedes no transition in β_m. Consider the occurrence net shown in Fig. 2. In that figure, transition t_7, t_2', t_4', t_5' are end transitions. Among them, t_7, t_2' are cutoff transitions and t_4', t_5' are end transitions that precede no transition in β_m.

The following lemmas were proven in [25].

Lemma 4: A marking μ is a reachable marking of the original net system Σ if and only if there is a configuration C of $\mathcal{N}_f$ such that $\mu = M(h_f(Cut(C))$.

Proof: See [25]. •

Lemma 5: If $\mathcal{N}$ is n-safe, then $\mathcal{N}_f$ is finite.

Proof: See [25]. •

It is obvious to see that $\mathcal{N}_c$ enumerates all the reachable markings of the original net system, since $\mathcal{N}_c$ contains $\mathcal{N}_f$. It is also clear that $\mathcal{N}_c$ is finite if $\mathcal{N}_f$ is finite, since there are finite number of end transitions in $\mathcal{N}_f$. The following lemma proves that β_c enumerates all the transitions reachable from μ_0.

Lemma 6: For any transition t of Σ that is reachable from μ_0, there exists a transition t_c of $\mathcal{N}_c$ such that $h_c(t_c) = t$.

Proof: Note that for a configuration C of the unfolding $\beta_m = (\mathcal{N}_m, h_m)$, all the transitions in the original net $\mathcal{N}$ which are enabled by $M(h_m(Cut(C)))$ should have their image appear in $\mathcal{N}_m$. Since every reachable marking of $\mathcal{N}$ maps to a Cut of $\mathcal{N}_f$ and $\mathcal{N}_c$ contains $\mathcal{N}_f$, then any reachable transition must have their image appear in $\mathcal{N}_c$. Another version of the proof can be found in [24]. •

Lemma 7: Let C_1 and C_2 be two configurations of $\mathcal{N}_f$ and $C_1 \subseteq C_2$, then in the original net system $\Sigma = (\mathcal{N}, \mu_0)$, $M(h_f(Cut(C_2)))$ is reachable from $M(h_f(Cut(C_1)))$.

Proof: By Lemma 3, there exists a minimal set of concurrent transitions T_{c_1} of $\mathcal{N}'$ such that $C_1 = \cup_{t \in T_{c_1}}[t]$ and a minimal set of concurrent transitions T_{c_2} of $\mathcal{N}'$ such that $C_2 = \cup_{t \in T_{c_2}}[t]$. Since $C_1 \subseteq C_2$, then for every $t_1 \in T_{c_1}$, there must exists a $t_2 \in T_{c_2}$ such that $t_1 < t_2$ or $t_1 = t_2$. Pick a transition $t_1 \in T_{c_1}$, and look for the associated transition $t_2 \in T_{c_2}$. We can see that t_2 is not in conflict with any transition in T_{c_1}, since otherwise t_2 and transitions in T_{c_1} are in the same configuration. Since t_2 is not in conflict with any transition in T_{c_1}, then all transitions in $[t_2] \backslash [t_1]$ can be fired one after another and it follows that $M(h_f(Cut(C_1 \cup [t_2])))$ is reachable from $M(h_f(Cut(C_1)))$. Now, pick another transition in T_{c_1} and continue the above process, by induction we conclude that $M(h_f(Cut(C_2)))$ is reachable from $M(h_f(Cut(C_1)))$. •

We conclude this section with an example to illustrate the unfolding process. In the occurrence net, we define the *depth* of a place s as the number of transitions preceding s. Places having the same depth are called a *tier*. A tier encapsulates a set of reachable markings. A new tier is formed by enumerating all the markings reached from a marking in the old tier after firing one transition. Unfolding is carried out tier by tier until every transition enabled by the latest tier is a cutoff transition.

Specifically, consider the original net system shown in the left part of Fig. 3. The initially marked places $\{s_1, s_5, s_6, s_7, s_9\}$ form the first tier. The second tier is formed by enumerating the cuts of all configurations consisting of transitions enabled by the first tier. In this example, there are four transitions enabled by the first tier, namely $\{t_2, t_4, t_5, t_6\}$. Since these transitions are concurrent, then every element in the power set of $\{t_2, t_4, t_5, t_6\}$ is a configuration and the second tier is formed by firing every element in the power set of $\{t_2, t_4, t_5, t_6\}$. It is worth noticing that in forming the second tier, we do not enumerate arbitrary interleaving of t_2, t_4, t_5, t_6 and this avoids the state explosion problem that occurs in constructing the reachability graph.

Having formed the second tier, the third tier can be constructed by enumerating the cuts of all configurations that contain a new transition that is enabled by a cut in the second tier. For example, we add transition t_7 and places s_1'', s_9' to the occurrence net since t_7 is enabled by $Cut(\{t_6\})$ and $h_c((Cut(\{t_6, t_7\}))) = \{s_1, s_9, s_5, s_6, s_7\}$. Note also that t_7 is a cutoff transition since $M(\{s_1, s_9, s_5, s_6, s_7\}) = \mu_0$. After the third tier is obtained, the fourth tier can be constructed in the same way. In obtaining the fourth tier, it is easy to see that t_3' is also a cutoff transition since $M(h_c(Cut([t_3'])) = M(\{s_1, s_9, s_5, s_6, s_7\}) = \mu_0$. It is also true that t_2' is a cutoff transition since $t_3' < t_2'$. Note that t_2' is not a post cutoff transition since $t_1 < t_2'$ and t_1 is not a cutoff transition. Transitions t_4' and t_5' are not cutoff transitions. But no transition will succeed t_4' and t_5' since no transition is enabled at marking $M(h_c(Cut([t_4'] \cup [t_5'])) = M(\{s_2, s_4, s_6, s_8, s_9\})$. It can be seen that any transition enabled by the fourth tier will be a post cutoff transition since it succeeds both t_7 and t_2'. Therefore, it is obvious that the fourth tier is the final tier in the branching process.

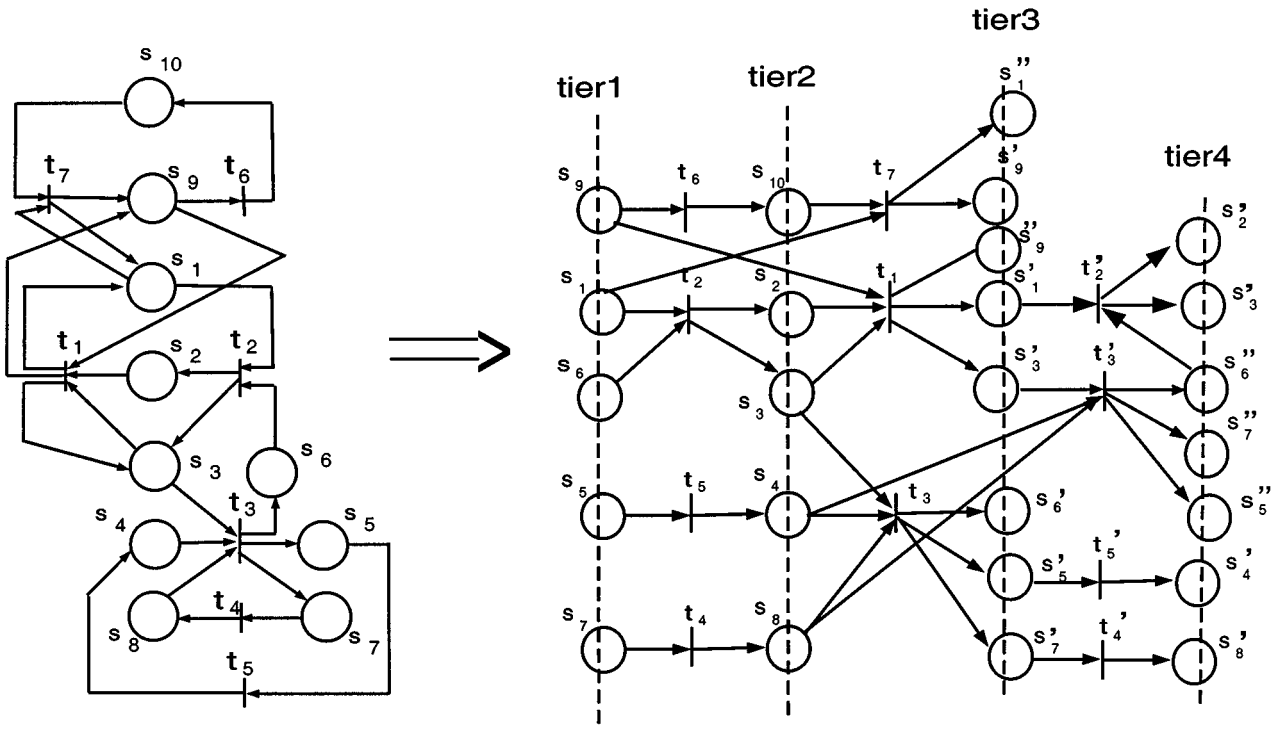

Fig. 3. An unfolding example.

It is also worth noticing that deadlocks in the original net system can be identified in the unfolding process. In the example, we can see that the cut of configuration $\{t_2, t_3, t_4, t_5, t_4', t_5'\}$ is a deadlocked marking since it enables no transition. We can also see that the critical transition that leads to this deadlock is t_3, since after the firing of t_3 the net system has no choice but to reach the deadlock. These informations are used in Section IV to obtain necessary and sufficient conditions that characterize the liveness of the original net system.

Remark: Notice that although the above example only shows the unfolding of a 1-safe net system, the unfolding of bounded but unsafe net systems can be achieved following the same steps. The only difference is that if a place in the original net systems contains more than one token, then we need to make as many copies of this place in the occurrence net as the number of tokens in that place. To further illustrate this point, consider the unfolding of the same example net with place s_9 containing 2 tokens initially. The first tier of the unfolding, therefore, consists of one copy of places s_1, s_5, s_6, s_7 and two copies of s_9 of the original net. The second and following tiers are still formed by enumerating all the markings reachable from the previous tier. The determination of cutoff transitions still follows the same rule. The complete occurrence net for the unsafe net system is shown in Fig. 4.

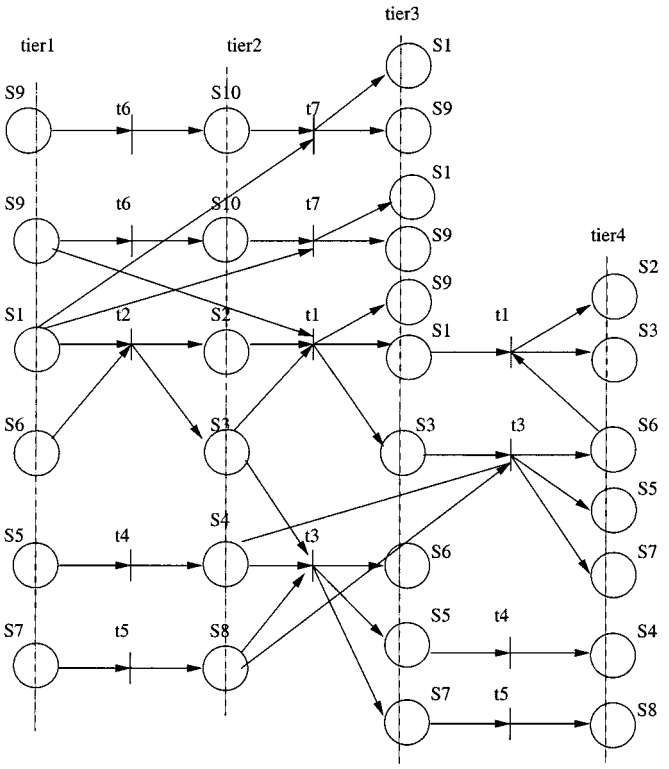

Fig. 4. The unfolding of an unsafe net system.

Remark: The computational complexity of unfolding depends, to a large degree, on the structure of the original net system. It has long been recognized that unfolding can reduce the space complexity required to represent systems containing a large number of concurrent executions. This means that an unfolding's occurrence net can be much more compact than the standard reachability graph. Due to this advantage, unfolding has been used as a verification tool for electronic circuits [24]. The time complexity of constructing the occurrence net is equal to the time-complexity of constructing the reachability graph. This is because both constructions require an enumeration of the net system's reachable markings. Given the speed, however,

of modern computers, space complexity (e.g., the amount of memory required to store the graph) may be more important than the construction's time-complexity.

Remark: The problem addressed in this paper is intimately connected to liveness verification in bounded net systems. There may be problems for which the algorithm's computational complexity is unattractive. The key, of course, is to identify a subclass of the original problem for which the algorithm works well (i.e., with polynomial complexity). For unfolding methods, this sub-class of problems can obviously be characterized by the minimal number of configurations in the net system. In other words, the number of configurations can be thought of as a measure of the "dimension" of the concurrent system. Systems with a large number of concurrent configurations are "complex" and for these systems we cannot expect unfolding to provide any greater benefits than the exhaustive enumeration found in the reachability graph. For those systems, however, in which there are a handful of concurrent configurations unfolding methods are attractive analysis tools.

IV. Liveness Verification

This section proves necessary and sufficient conditions that characterize the liveness of bounded ordinary Petri nets. These conditions are based on the intuition that since the interleaving of all fundamental executions completely characterizes a system's behavior, then liveness can be verified by identifying local (or global) deadlocks between fundamental executions of the net system. The existence of a (local or global) deadlock means that at least part of the system cannot proceed and therefore some transitions can never be enabled (or re-enabled). The next concern is the liveness of every net transition when there is no deadlock in the system. Note that although in this case every part of the system can proceed, some transitions may still never be re-enabled since they do not belong to any transition cycle.

The preceding discussion provided an intuitive characterization of the liveness condition presented in this paper. The remainder of this section proves this characterization in detail. Specifically, Section IV-A demonstrates how *deadlocked configurations* can be identified in the finite prefix. Section IV-B shows how transition cycles can be determined by identifying *cut cycles* in the *cut graph*. Finally, Section IV-C summarizes and proves the main theorem for liveness verification.

Remark: It is worth noticing that this paper characterizes the liveness of bounded Petri nets in a different way than did [1]. The difference can be explained as follows. Recall that in [1], the liveness of a bounded Petri net was characterized by the liveness of base configurations and the absence of *cyclic locks* between base configurations. Intuitively, the liveness of base configurations means that any "sequential" execution of base configurations does not cause either deadlock or the unreachability of transitions, while the absence of cyclic locks means that the "concurrent" execution of base configurations does not result in any (local or global) deadlock. This section refines the results in [1] by encapsulating all of the "sequential" or "concurrent" deadlocks in the notion of *deadlocked configurations* and characterizing the repeatability of transitions in *cut cycles* in the *cut graph*. Details of these refinements and justifications of their advantages appears in Sections IV-A–C.

A. Characterization of Local Deadlocks

Let $\Sigma = (\mathcal{N}, \mu_0)$ be a net system and β_c be the finite prefix of its unfolding defined earlier. We define a *base configuration* $BC(t^e)$ as the cause of t^e where t^e is an end transition of β_c. We say a base configuration $BC(t^e)$ is *deadlock free*, if t^e is a cutoff transition. We say $BC(t^e)$ is deadlocked, if t^e is not a cutoff transition. Note that in a deadlock free base configuration, assuming t' is the transition such that $h_c(Cut([t'])) = h_c(Cut([t^e]))$, then transitions in the set $[t^e]\backslash[t']$ forms a transition cycle (since $Cut[t']$ and $Cut[t^e]$ represent the same marking). Note also that a deadlocked base configuration represents a set of occurrence sequences that leads to a local deadlock, since there is no transition succeeding t^e in the unfolding β_m.

In the finite prefix in Fig. 3, t_7, t_2', t_4', t_5' are the end transition of the finite prefix β_c. Therefore, there are four base configurations in β_c, namely $BC(t_7), BC(t_2'), BC(t_4'), BC(t_5')$. Among them, $BC(t_7)$ and $BC(t_2')$ are deadlock free, since t_7 and t_2' are cutoff transitions. $BC(t_4')$ and $BC(t_5')$ are deadlocked, since t_4', t_5' are not cutoff transitions. It is easy to see that transition sequences t_6t_7 and $t_2t_1t_3'$ form two transition cycles since the marking vector returns to the initial marking after firing these sequences of transitions. It can also be seen that the net system will reach a deadlock once (the image of) all transitions in $[t_4']$ or $[t_5']$ are fired consecutively.

A local deadlock can be characterized by the occurrence sequences that lead to this deadlock. In the occurrence net of the finite prefix β_c, these occurrence sequences are represented by configurations.

Recall from Lemma 3 that for a configuration C in β_c, there exists a unique set of transitions T_c such that $C = \cup_{t \in T_c}[t]$. We call every transition $t \in T_c$ an *end transition* of configuration C. A configuration C_1 is said to be a *deadlock free configuration*, if every end transition of C_1 is a cutoff transition. In other words, C_1 is the union of several (concurrent) deadlock free base configurations. A configuration C_1 is said to be a *deadlocked configuration*, if it is not a subset of any deadlock free configuration. A deadlocked configuration is said to be *minimal*, if it does not contain any other deadlocked configuration. A deadlocked configuration is said to be *maximal*, if it is not a subset of any other deadlocked configuration. A deadlocked base configuration is a deadlocked configuration.

Intuitively, a minimal deadlocked configuration represents a *critical point* where a deadlock free path and a deadlocked path diverge, while a maximal deadlocked configuration encapsulates all the paths that lead to a (local or global) deadlock. The following lemma proves that the existence of a deadlocked configuration implies that there exist transitions that cannot be enabled repeatedly after the deadlock occurs.

Lemma 8: If there exists a deadlocked configuration C in β_c, then there must exist a transition t in the original net such that t cannot be enabled repeatedly from $M(h_c(Cut([C_m])))$, where C_m is the maximal deadlocked configuration that contains C.

Proof: Let T_{c_m} be the set of end transitions of C_m. Apparently, there must exist a transition $t \in T_{c_m}$ such that t is not

a cutoff transition, since otherwise C_m will be deadlock free. If t is not an end transition of β_c, then for any transition t_1 succeeding t, $h_c(t_1)$ is not reachable from $M(h_c(Cut(C_m)))$. This is because if t_1 is reachable from $M(h_c(Cut(C_m)))$, then $C_m \cup [t_1]$ is a deadlocked configuration and that contradicts the fact that C_m is maximal. If t is an end transition of β_c, then t cannot be enabled repeatedly from $M(h_c(Cut(C_m)))$. This is because if t can be enabled repeatedly from $M(h_c(Cut(C_m)))$, then that means the deadlocked base configuration $BC(t)$ keeps receiving tokens from other deadlock-free base configurations in C_m. As a result, any place $s \in h_c(t\bullet \setminus \bullet t)$ will contain an infinite number of tokens. This contradicts the fact that the original net system is bounded. •

In the finite prefix in Fig. 3, since $BC(t'_4), BC(t'_5)$ are deadlocked base configurations, then any configuration that contains a portion of these two base configurations is a deadlocked configuration. These deadlocked configurations are, $[t_3]$, $[t'_4]$ and $[t'_5]$. Also notice that configuration $C = \{t_2, t_6\}$ is also a deadlocked configuration, since $BC(t'_3)$, $BC(t_7)$ are the only two deadlock free configurations and since C is not a subset of either of them. Among the four deadlocked configurations, $[t_3]$ is a minimal deadlocked configuration, $[t'_4]$, $[t'_5]$ are maximal deadlocked configurations, $\{t_2, t_6\}$ is both a minimal deadlocked configuration and a maximal deadlocked configuration. For the three maximal deadlocked configurations, it is easy to see that t_7 is not reachable from $M(h_c(Cut(\{t_2, t_6\})))$ and t_3 is not reachable from either $M(h_c(Cut([t'_4])))$ or $M(h_c(Cut([t'_5])))$.

Remark: The notion of *deadlocked configurations* introduced in this section encapsulates all the (local or global) deadlocked caused by "sequential" execution of base configurations and all the cyclic locks [1]. The advantage of this encapsulation is the reduction of computation in identifying deadlocks. Based on the example shown in Fig. 3, it can be seen that deadlocked configurations can be identified on-the-fly in the unfolding process. The identification of cyclic locks as defined originally in [1], on the other hand, may be computationally more expensive since it searches through every transition in every combination of concurrent base configurations.

B. Characterization of Transition Cycles

It is easy to see that, when there is no deadlocked configuration in the finite prefix, firing transitions in different base configurations either concurrently or sequentially will reach the same final marking. This means that the absence of deadlocks enables us to examine the net system's behavior by only observing the "sequential" (as opposed to "parallel" or "concurrent") execution of base configurations. This feature allows us to determine the liveness of transitions by only verifying the repeatability of base configurations. This section uses a structure called a *cut graph* to characterize the reachability between base configurations and proves that the original net is live if and only if every maximal cycle in the cut graph contains a copy of every transition in the original net.

Let $BC(t_1^e)$, $BC(t_2^e)$ be two deadlock free base configurations in β_c. We say $BC(t_2^e)$ is *reachable* from $BC(t_1^e)$, denoted as $BC(t_1^e) \rightarrow BC(t_2^e)$, if $[t_1] \subset [t_2^e]$, where $t_1 \in BC(t_1^e)$ and $h_c(Cut([t_1])) = h_c(Cut([t_1^e]))$. Note that in this case the set of transitions in $BC(t_2^e)$ that is reachable from $M(h_c(Cut([t_1^e])))$ is $[t_2^e] \backslash [t_2]$, where $t_2 < t_2^e$ and $h_c(Cut([t_2])) = h_c(Cut([t_2^e]))$. A *cut graph* (B, A) is a directed graph such that B is the set of all base configurations in β_c and $A \subseteq B \times B$ is a set of directed arcs such that $(BC(t_1^e), BC(t_2^e)) \in A$ if and only if $BC(t_1^e) \rightarrow BC(t_2^e)$.

The semantics of a cut graph can be explained as follows. Each node $BC(t_1^e)$ in the graph is associated with a base configuration. Each arc $(BC(t_1^e), BC(t_2^e))$ means that base configuration $BC(t_2^e)$ is reachable from base configuration $BC(t_1^e)$. The cut graph therefore simply maps the causal dependency between different base configurations without regard to the sequential or concurrent nature of these configurations.

We denote a cycle $(BC(t_1^e), BC(t_2^e))$, $(BC(t_2^e), BC(t_3^e))$, $\ldots, (BC(t_k^e), BC(t_1^e))$ in the cut graph as a *cut cycle*. We say that a cut cycle is *live*, if either of the following conditions holds:

- $h_c(\cup_{j=1}^{k} [t_j^e] \backslash [t_j]) = T$, where T is the set of all transitions in the original net $\mathcal{N}$ and $\forall\ j,\ t_j \in BC(t_j^e)$, $h_c(Cut([t_j])) = h_c\left(Cut\left([t_j^e]\right)\right)$;
- $M\left(h_c\left(Cut\left([t_k^e]\right)\right)\right) = \mu_0$.

Note that the first condition means that the cut cycle includes all the transitions in the original net. The second condition means that after traversing all base configurations in a cut cycle, the marking vector returns to the initial marking. We say that a transition t is *in* a cut cycle $(BC(t_1^e), BC(t_2^e))$, $(BC(t_2^e), BC(t_3^e)), \ldots, (BC(t_k^e), BC(t_1^e))$, if $t \in \cup_{j=1}^{k} [t_j^e] \backslash [t_j]$, where $t_j < t_j^e$, $h_c(Cut([t_j])) = h_c(Cut([t_j^e]))$. We say a cut cycle is *contained* in another cut cycle, if every node and every arc of this cut cycle also belong to the other cut cycle. We say that a cut cycle is *maximal*, if it is not contained in any other cut cycle.

In Fig. 3, we can see that $BC(t_7) \rightarrow BC(t'_2)$, $BC(t'_2) \rightarrow BC(t_7)$, $BC(t'_2) \rightarrow BC(t'_2)$, $BC(t_7) \rightarrow BC(t_7)$, $BC(t'_2) \rightarrow BC(t'_4)$, $BC(t'_2) \rightarrow BC(t'_5)$, $BC(t_7) \rightarrow BC(t'_4)$, $BC(t_7) \rightarrow BC(t'_5)$. The cut graph is shown in Fig. 5. In the cut graph, we can see that there are three cut cycles, namely cycle $(BC(t'_2), BC(t'_2))$, cycle $(BC(t_7), BC(t_7))$ and cycle $(BC(t'_2), BC(t_7))$, $(BC(t_7), BC(t'_2))$. The union of these three cut cycles forms a live maximal cut cycle.

A node $BC(t_1^e)$ in the cut graph is said to be a *dead node*, if there exists no node that is reachable from $BC(t_1^e)$. It is easy to see that any dead node is a deadlocked base configuration and it is trivial to prove that a cut graph does not contain any deadlock if and only if there is no deadlock configuration in the finite prefix. In Fig. 5, node $BC(t'_4)$ and node $BC(t'_5)$ are deadlocks.

Remark: The true value of cut graphs lies in the fact that they represent a higher level abstraction of the net system's executions. Specifically, a cut graph provides information on how *fundamental executions* or base configurations of the net system interact. A cut graph encapsulates all the concurrency between fundamental executions into the sequential execution of base configurations. This encapsulation of concurrency greatly reduces the complexity involved in the liveness verification and supervision in the sense that liveness verification and supervision can be performed by only observing the reachability between base configurations without worrying about how they interleave.

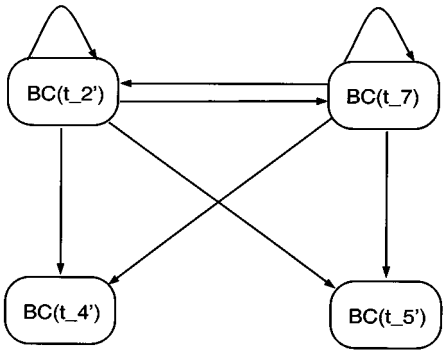

Fig. 5. The cut graph of the finite prefix in Fig. 2.

Lemma 9: If there does not exist any deadlocked configuration in β_c, then the original net is live if and only if every maximal cut cycle is live.

Proof:

Sufficiency: we only need to proof that for any reachable marking μ and any transition t in the original net, t is reachable from μ. From Lemma 4, we know that for any μ reachable from μ_0, there is a configuration C such that $M(h_c(Cut[C])) = \mu$. From Lemma 3, we know that there exists a set of concurrent transitions $t_1, \ldots, t_k$ such that $C = \cup_{i=1}^{k}[t_i]$. Note that since there is no deadlock, then any end transition is a cutoff transition. Pick a transition $t_i \in \{t_1, \ldots, t_k\}$ such that $M(h_c(Cut(C \cup [t_i^e])))$ is reachable from $M(h_c((Cut(C)))$ (t_i^e is the cutoff transition succeeding t_i.) This transition t_i always exists, since otherwise C is a deadlocked configuration. Now, if there exists a copy of transition t, denoted as t^0, such that $t^0 \in [t_i^e]\backslash[t_i]$, then apparently t is reachable from μ, since in order to reach $M(h_c(Cut(C \cup [t_i^e])))$, we have to first reach $M(h_c(Cut(C \cup [t^0])))$. If there does not exist such a t^0, then find another base configuration $BC(t_{i1}^e)$ such that $BC(t_i^e) \rightarrow BC(t_{i1}^e)$ and $M(h_c(Cut(C \cup [t_{i1}^e]\backslash[t_i^e])))$ is reachable from $M(h_c(Cut(C)))$. This base configuration $BC(t_{i1}^e)$ always exists since otherwise $C \cup [t_i^e]$ will be a deadlocked configuration. If t^0 exists in $[t_{i1}^e]\backslash[t_{i1}]$, (where $t_{i1} < t_{i1}^e$, $h_c(Cut([t_{i1}])) = h_c(Cut([t_{i1}^e]))$), then the proof is done. If t^0 does not exist in $[t_{i1}^e]\backslash[t_{i1}]$, then we look for another base configuration $BC(t_{i2}^e)$ that is reachable from $BC(t_{i1}^e)$ and recheck the existence of t^0. Continue this process and we are guaranteed to find t^0 before completing a maximal cut cycle, since every live maximal cut cycle contains a copy of t.

Necessity: Suppose that there is a maximal cut cycle $BCM = (BC(t_1^e), BC(t_2^e)), (BC(t_2^e), BC(t_3^e)), \ldots, (BC(t_{k-1}^e), BC(t_k^e)), (BC(t_k^e), BC(t_1^e))$ that is not live. It follows that there must exist a transition t in the original net such that no copy of t is contained in BCM. Since the cut cycle BCM is maximal, we know that when firing transitions in BCM, we cannot diverge to another maximal cut cycle. This means t will not be reachable from any marking μ reached in traversing BCM. Therefore, we see that the original net is not live.

•

C. The Main Theorem

Combining results in Sections IV-A and B, we obtain the following theorem that verifies the liveness of bounded ordinary Petri nets.

Theorem 1: A bounded net system $(\mathcal{N}, \mu_0)$ is live if and only if the finite prefix $\beta_c = (\mathcal{N}_c, h_c)$ of $\mathcal{N}$'s unfolding satisfies the following two conditions:

1) there does not exist any deadlock configuration in β_c;
2) every maximal cut cycle in the cut graph is live.

Proof:

Sufficiency: Directly follows from the sufficiency proof of Lemma 9.

Necessity: Directly follows from Lemma 8 and the necessity proof of Lemma 9.

•

The net system in Fig. 1 is not live, since it contains several deadlocked configurations.

Remarks: The computational complexity involved in the verification of liveness depends heavily on the computation spent in obtaining the finite prefix, since deadlocked configurations and the cut graph can be easily constructed during the unfolding process. Although it is true that there exist cases where the construction of finite prefixes does not scale well with the size of the original net, the verification method in this paper is still worth doing, because it provides valuable information for supervisory control. The unfolding method extracts strings of causally related transitions and this causal relationship can help develop necessary and sufficient conditions for the existence of liveness-enforcing supervisors. Exploring the causal relationship among transitions is extremely useful when the original net contains uncontrollable transitions. In such cases, the controller needs to decide, among all the transitions preceding the uncontrollable transition, which transition to disable in order to obtain the maximally permissive liveness-enforcing supervisor. The Section V discusses how the information obtained in the network unfolding can be used to construct the maximally permissive liveness-enforcing supervisor.

V. Maximally Permissive Marking Based Liveness-Enforcing Supervision

This section extends the liveness verification result presented in Section IV to synthesize a maximally permissive marking based liveness-enforcing supervisor. Recall that a supervisor is a mapping that returns a control input for every observable output and the control input imposes a restriction on the original net system's behavior. We make the conjecture that to enforce liveness in a maximally permissive manner, we need to impose the restriction right onto the *critical point* where live executions and deadlocked executions are about to diverge. Understanding that the original net system's behavior can be characterized by the interleaving of base configurations or fundamental executions, we can use the result from Section IV to identify interleavings of fundamental executions that can lead to system deadlock and to identify *critical transitions* that cause the deadlock's irreversible occurrence. The maximally permissive supervisor is

obtained by control disabling the firing of critical transitions at those markings that enable these transitions.

It is worth noticing that although the intuition remains the same, the characterization of the existence of maximally permissive liveness enforcing supervisors in this section is different from but conceptually equivalent to the characterization in [2]. This difference follows directly from the refinement of liveness characterization in Section III.

We start our discussion by formally defining the notion of a marking based supervisor. A *marking based supervisor* $\mathcal{P}: \mathrm{Re}(\mu_0) \rightarrow \{0,1\}^{|T|}$ is a one to one mapping that returns a $|T|$-dimensional binary vector for every reachable marking. (T is the set of transitions in the original net). The supervisor $\mathcal{P}$ permits the firing of transition t_i at marking μ, only if $\mathcal{P}(\mu)_i = 1$. We say the transition t_i is *state enabled* at marking μ, if at marking μ, all input places to a transition t_i contain a token. We say the transition t_i is *control enabled* (*control-disabled*) at μ, if $\mathcal{P}(\mu)_i = 1$ ($\mathcal{P}(\mu)_i = 0$). A transition in the supervised net has to be state enabled and control enabled before it can fire.

Denote $R^{\mathcal{P}}(\mu_0)$ as the set of reachable markings of the supervised net system, we say that a marking based supervisor $\mathcal{P}$ *enforces* the liveness of the original net system $(\mathcal{N}, \mu_0)$, if every transition of the original net system is reachable from every marking in $R^{\mathcal{P}}(\mu_0)$. A marking based supervisor is said to be *liveness-enforcing*, if it enforces the liveness of the original net system. We say a net system is *completely controllable*, if every transition of the net system is controllable.

We say that a transition t in the finite prefix β_c is a *critical transition*, if the following conditions hold:

- t is an end transition of some minimal deadlocked configuration;
- there exists a base configuration $BC(t^e)$ such that $[t]\backslash t \subset BC(t^e)$ and $BC(t^e)$ is a node of some live cut cycle; but there does not exist any $BC(t^e)$ such that $[t] \subseteq BC(t^e)$ and $BC(t^e)$ is a node of some live cut cycle.

Intuitively, the first condition means that the firing of a critical transition t may cause a local or global deadlock, while the second condition means that the firing of t may lead to the unreachability of certain transitions, since $[t]$ does not belong to any live cut cycle.

In the occurrence net in Fig. 3, t_6, t_2 are critical transitions. They are the two end transitions of the minimal deadlocked configuration $\{t_6, t_2\}$. t_3 is another critical transition, since it is the end transition of minimal deadlocked configuration $[t_3]$.

Let C be a configuration in β_c, we say that $M(Cut(C))$ is a *critical marking*, if the following conditions hold:

- there exist a minimal deadlock configuration C_d and a critical transition t such that $C \cup t = C_d$;
- there exists a critical transition t such that $[t] \subseteq C \cup t$ and there does not exist any $BC(t^e)$ such that $[t] \subseteq BC(t^e)$ and $BC(t^e)$ is a node of some live cut cycle.

Intuitively, a critical marking represents a state that is "one step away" from an execution that is not live. In Fig. 2, the critical markings are $M(Cut(\{t_2\})) = \{s_9, s_2, s_3, s_5, s_7\}$, $M(Cut(\{t_6\})) = \{s_{10}, s_1, s_6, s_5, s_7\}$, and $M(Cut(\{t_2, t_4, t_5\})) = M(Cut([t_3]\backslash t_3)) = \{s_9, s_2, s_3, s_4, s_8\}$.

The following theorem verifies the existence of liveness-enforcing marking based supervisors for completely controllable nets.

Theorem 2: There exists a liveness-enforcing supervisor for a completely controllable bounded net system $(\mathcal{N}, \mu_0)$, if and only if there exist at least one live cut cycle in the cut graph.

Proof:

Sufficiency: The sufficient part is proven in a constructive way. In other words, we first derive a marking based supervisor $\mathcal{P}$ and then prove $\mathcal{P}$ enforces liveness. Let μ be a reachable marking of the original net system, then there must exist a configuration C of $\mathcal{N}_c$ such that $M(h_c(Cut(C))) = \mu$. For a transition $t_j \in T$, let $\mathcal{P}(\mu)_j = 0$, if there exists a transition $t \in h_c^{-1}(t_j)$ such that the following conditions are satisfied:

1) $M(Cut(C))$ is a critical marking;
2) t is a critical transition;
3) $M(Cut(C))$ enables t.

We let $\mathcal{P}(\mu)_j = 1$, if otherwise.

We now want to prove that the supervisor $\mathcal{P}$ enforces the liveness of $(\mathcal{N}, \mu_0)$. We need to prove that any transition of the original net system is reachable from any marking μ reachable from μ_0 under $\mathcal{P}$. It can be seen from its construction that $\mathcal{P}$ disables all the occurrence sequences that lead to deadlock or cut cycles that are not live. In other words, $\mathcal{P}$ only permits the firing of live cut cycles. Liveness of the supervised net system is therefore guaranteed since every transition is reachable in any live cut cycle.

Necessity: Assuming there does not exist any live cut cycle in the cut graph, then there should not exist any liveness-enforcing supervisor. This is because for any cut cycle that is not control-disabled by the supervisor, we know from lemma 9 that there should be some transition that is not reachable.

•

There exists a liveness-enforcing supervisor for the net system in Fig. 1, since there exists a live cut cycle $(BC(t_2'), BC(t_7)), (BC(t_7), BC(t_2'))$. To enforce its liveness, we want to disable all the critical transitions at there critical markings. The complete supervisor is shown in Fig. 6. The first column in the figure shows the current configuration, i.e., the set of transitions that have been fired. The second column shows the cut of each configuration. The third column shows the marking vector corresponding to each cut and the last column shows the control vector associated with each marking vector. The derivation of the supervisor can be explained as follows. Note that the supervisor is a mapping between reachable markings and boolean control vectors. The value of each element in the control vector corresponds to the enabling or disabling of a transition in the original net. Each reachable marking can be represented by the cut of a configuration in the occurrence net. The table in Fig. 6 actually shows the mapping from cuts of configurations to control vectors. Specifically, we determine the cut of every configuration in the occurrence net and obtain

Configuration	Cut	Marking	Control - vector
∅	{ s1, s5, s6, s7, s9 }	[1000111010] ^{T}	[1 1 1 1 1 11] ^{T}
{ t2 }	{ s2, s3, s5, s7, s9 }	[0110101010] ^{T}	[1 1 1 1 1 01] ^{T}
{ t6 }	{ s1, s5, s6, s7, s10 }	[1000111001] ^{T}	[1 0 1 1 1 11] ^{T}
{ t2, t4 }	{ s2, s3, s5, s8, s9 }	[0110100110] ^{T}	[1 1 1 1 1 01] ^{T}
{ t6, t4 }	{ s1, s5, s6, s8, s10 }	[1000110101] ^{T}	[1 0 1 1 1 11] ^{T}
{ t2, t5 }	{ s2, s3, s4, s8, s9 }	[0110100110] ^{T}	[1 1 1 1 1 01] ^{T}
{ t6, t5 }	{ s1, s4, s6, s8, s10 }	[1000110101] ^{T}	[1 0 1 1 1 11] ^{T}
{ t2, t4, t5 }	{ s2, s3, s4, s8, s9 }	[0111000110] ^{T}	[1 1 0 1 1 01] ^{T}
{ t 6, t4, t5 }	{ s2, s3, s4, s8, s10 }	[0111000101] ^{T}	[1 0 1 1 1 11] ^{T}
Otherwise	/	/	[1 1 1 1 1 11] ^{T}

Fig. 6. The liveness-enforcing supervisory policy for the example net system.

the first and second columns of the table. We then look for the reachable marking associated with each cut and obtain the third column. Finally, we examine every transition enabled by a cut. If the cut and the transition satisfy the conditions stated in the sufficiency proof, then the element of the control vector is set to 0 (meaning disable). The element is set to 1, if otherwise. We thus obtain the control vectors associated with every cut and complete the last column of the supervisor table.

The following corollary proves the maximal permissiveness of the liveness-enforcing supervisory policy constructed in the sufficiency proof of Theorem 2.

Corollary 1: Assuming the net system is completely controlled, the supervisor $\mathcal{P}$ constructed in the sufficiency proof of Theorem 2 is maximally permissive.

Proof: Assuming there exists a more permissive liveness-enforcing supervisor $\mathcal{P}_1$, then $\mathcal{P}_1$ will control enable some critical transition t. The enabling of t will lead to either a deadlock or a cut cycle that is not live. In either cases, some transition succeeding t will not be reachable from $M(h_c(Cut(C \cup t)))$. Therefore, we see that $\mathcal{P}_1$ cannot enforce liveness and hence the contradiction. •

Another important property of the supervisory policy constructed in the sufficiency proof is that it only requires the controllability of all the critical transitions. The following corollary summarizes this point.

Corollary 2: To apply the supervisor $\mathcal{P}$ constructed in the sufficiency proof of Theorem 2, only the controllability of all the critical transitions is needed.

Proof: Follows directly from the construction of $\mathcal{P}$. •

Remark: Corollaries 1 and 2 show that the supervisor constructed in the sufficiency proof of Theorem 2 is *maximally permissive*. Moreover, Corollary 2 shows that the maximally permissive supervisor $\mathcal{P}$ only requires the controllability of critical transitions. This means that to construct $\mathcal{P}$, the controllability requirement is minimal. Furthermore, it can be clearly seen from the construction that to obtain the supervisor, we only need to decide the control vectors at critical markings and these control vectors are easily computed once the critical transitions are identified. Recall that a minimal deadlocked configuration is a configuration reached after firing a critical transition at a critical marking. Critical markings and critical transitions can be efficiently identified since minimal deadlocked configurations can be identified on the fly in the unfolding process. These characteristics indicate that the supervisory policy presented in this paper is computationally more efficient and at the same time less restrictive. These improvements come mainly from the use of network unfolding, which efficiently enumerates the reachable marking and preserves the ordering relationship among transitions of the original net system.

Remark: The supervisory policy in Fig. 6 is for a safe network. The same procedure can be used to construct a supervisory policy for the unsafe network whose unfolding was shown in Fig. 4. In this occurrence net, the deadlocked configurations are $\{t_2, t_6, t_6\}$ and $\{t_2, t_3, t_4, t_5\}$. The supervisory policy shown in Fig. 7 simply disables the occurrence of these configurations.

We now move to net systems with uncontrollable transitions. From Corollary 2, we know that the controllability of every critical transition is crucial in constructing the maximally permissive liveness-enforcing supervisor. If a certain critical transition t is uncontrollable, it is natural to look for some controllable transition t' in $[t]$ in order to control disable t. There may be multiple controllable transitions in $[t]$ to chose from. One rule for choosing t' is that control disabling t' should not disable any other live behavior. The following paragraph defines this rule.

Let $T^{\mathrm{ctl}} \subseteq T$ denote the set of controllable transitions. For every critical transition t in the net system's finite prefix β_c, we define $T^{[t]} \subseteq [t]$ as the set of transitions such that the following conditions hold for every transition $\bar{t} \in T^{[t]}$:

1) $\bar{t} = t$;
2) $\nexists t'$ such that $h_c(t') \neq h_c(t), {}^\bullet t' \cap {}^\bullet t \neq \emptyset, \bar{t} < t'$, and $[t']$ is a subset of some base configuration in a live cut cycle.

We say that a critical transition t is *cause controllable*, if $\exists t_1 \in T^{[t]}$ such that $h_c(t_1) \in T^{\mathrm{ctl}}$.

From the definition of $T^{[t]}$, we can see that since $\bar{t} \leq t$, $\forall\, \bar{t} \in T^{[t]}$, then control disabling any transition in $T^{[t]}$ will automatically control disable t. Furthermore, the second item means control disabling $\bar{t}$ cannot disable any live cut cycle. Therefore, the controllability of t is equivalent to the controllability of any transition in $T^{[t]}$ in constructing the liveness-enforcing supervisory policy.

In the occurrence net in Fig. 3, if t_4, t_5 are controllable, then for critical transition t_3, $T^{[t_3]} = \{t_4, t_5\}$. Therefore, control disabling either t_4 or t_5 will also disable t_3. Note that although $t_2 \in [t_3], t_2$ is not in $T^{[t_3]}$, because $t_2 < t_1, {}^\bullet t_1 \cap {}^\bullet t_3 = s_3$ and $[t_1]$ is a subset of base configuration $BC^{t_3'}$ that is in a live cut cycle.

The following definitions of *pre-critical transition* and *pre-critical marking* represents the critical point to implement maximally permissive liveness-enforcing supervisor when the critical transition is uncontrollable. These definitions are used in the proof of Theorem 3.

Let t be a critical transition in β_c. A transition $t^{\mathrm{pre}} \in T^{[t]}$ is said to be a *pre-critical* transition, if $\nexists t' \in T^{[t]}$ such that $t^{\mathrm{pre}} < t'$ and $h_c(t') \in T^{\mathrm{ctl}}$. In other words, there is no other

Configuration	Cut	Marking	Control-Vector
∅	{s1,s5,s6,s7,s9,s9}	[1000111020]^T	[1 1 1 1 1 1 1]^T
{t2, t6}	{s2,s3,s5,s7,s9,s10}	[0110101011]^T	[1 1 1 1 1 0 1]^T
{t6, t6}	{s1,s5,s6,s7,s10,s10}	[1000111002]^T	[1 0 1 1 1 1 1]^T
{ t2, t6, t4 }	{s2,s3,s5,s8,s9,s10}	[0110100111]^T	[1 1 1 1 1 0 1]^T
{ t6, t6, t4 }	{s1,s5,s6,s8,s10,s10}	[1000110102]^T	[1 0 1 1 1 1 1]^T
{ t2, t6, t5 }	{s2,s3,s4,s7,s9,s10}	[0111001011]^T	[1 1 1 1 1 0 1]^T
{ t6, t6, t5 }	{s1,s4,s6,s7,s10,s10}	[1001011011]^T	[1 0 1 1 1 1 1]^T
{ t2, t4, t5 }	{s2,s3,s4,s8,s9,s9}	[0111000120]^T	[1 1 0 1 1 1 1]^T
{t6, t4, t5 }	{s1,s4,s6,s8,s9,s10}	[1001010111]^T	[1 1 0 1 1 1 1]^T
{t2, t6, t4, t5 }	{s2,s3,s4,s8,s9,s10}	[0111000111]^T	[1 1 1 1 1 0 1]^T
{t6, t6, t4, t5 }	{s1,s4,s6,s8,s10,s10}	[1001010111]^T	[1 0 1 1 1 1 1]^T
otherwise	/	/	[1 1 1 1 1 1 1]^T

Fig. 7. The liveness-enforcing supervisory policy for the unsafe net system.

controllable transition that succeeds t^{pre} in $T^{[t]}$. In the occurrence net in Fig. 3, the precritical transition for t_3 is t_4 or t_5, if t_4 or t_5 is controllable. It is worth noticing that there is no pre-critical transition for t_2 or t_6. This means if either t_2 or t_6 is uncontrollable, then we cannot avoid the deadlock caused by firing t_2t_6 consecutively and therefore there does not exist a liveness-enforcing supervisor.

Let C_d be a minimal deadlocked configuration in β_c and let T_{c_d} denoted the set of end transitions of C_d. For any transition $t \in T_{c_d}$, let t^{pre} be the controllable precritical transition of t. Define a *preminimal* deadlocked configuration C_d^{pre} as the configuration that satisfies $C_d^{\text{pre}} = \cup_{t \in T_{c_d}} [t^{\text{pre}}]$. In other words, a preminimal deadlocked configuration represents the state "right after" we lose control of a deadlock. In the occurrence net in Fig. 3, the pre-minimal deadlocked configuration of $[t_3]$ is $\{t_4, t_5\}$, if t_4 and t_5 are controllable.

Let C be a configuration in β_c, we say that $M(Cut(C))$ is a *precritical marking*, if the following conditions hold:

- there exist a preminimal deadlocked configuration C_d^{pre} and a pre-critical transition t^{pre} such that $C \cup t^{\text{pre}} = C_d^{\text{pre}}$;
- there exists a pre-critical transition t^{pre} such that $[t^{\text{pre}}] \subseteq C \cup t^{\text{pre}}$ and there does not exist any $BC(t^e)$ such that $[t^{\text{pre}}] \subseteq BC(t^e)$ and $BC(t^e)$ is a node of some live cut cycle.

Intuitively, a precritical marking represents a state that is one "one step away" from an uncontrollable behavior that is not live. In the occurrence net in Fig. 3, $M(Cut(\{t_4\}))$ and $M(Cut(\{t_5\})$ are two precritical markings.

The following theorem verifies the existence of liveness-enforcing supervisory policies for n-safe net systems with arbitrary set of controllable transitions.

Theorem 3: For a bounded net system $(\mathcal{N}, \mu_0)$, there exists a liveness-enforcing supervisor if and only if there exists a live cut cycle in the cut graph and every critical transition is cause controllable.

Proof:

Sufficiency: Knowing that we need to control-disable pre-critical transitions at precritical markings, we can construct the liveness-enforcing supervisor in a way similar to that in Theorem 2 as follows. Let μ be a reachable marking of the original net system and C be the configuration C in $\mathcal{N}_c$ such that $M(h_c(Cut(C))) = \mu$. For a transition $t_j \in T$, let $\mathcal{P}(\mu)_j = 0$, if there exists a transition $t \in h_c^{-1}(t_j)$ such that the following conditions are satisfied:

1) $M(Cut(C))$ is a pre-critical marking;
2) t is a precritical transition;
3) $M(Cut(C))$ enables t.

We let $\mathcal{P}(\mu)_j = 1$, if otherwise.

Necessity: First, it is clear from the necessity proof of Theorem 2 that if there does not exist any live cut cycle in the cut graph, then there does not exist any liveness-enforcing supervisor. Second, it is also clear that if any critical transition is not cause controllable, then the net system cannot be prohibited from reaching certain deadlock or some cut cycle that is not live.

•

Policy P_1

Configuration	Cut	Marking	Control - vector
∅	{ s1, s5, s6, s7, s9 }	[1000111010]^T	[1 1 1 1 0 11]^T
{ t4 }	{ s1, s5, s6, s8, s9 }	[1000110110]^T	[1 1 1 1 0 11]^T
{ t2 }	{ s2, s3, s5, s7, s9 }	[0110101010]^T	[1 1 1 1 0 01]^T
{ t6 }	{ s1, s5, s6, s7, s10 }	[1000111001]^T	[1 0 1 1 0 11]^T
{ t2, t4 }	{ s2, s3, s5, s8, s9 }	[0110100110]^T	[1 1 1 1 0 01]^T
{ t6, t4 }	{ s1, s5, s6, s7, s10 }	[1000111001]^T	[1 0 1 1 0 11]^T
Otherwise	/	/	[1 1 1 1 1 11]^T

Fig. 8. The first liveness-enforcing supervisor.

In Fig. 1, assume that only transitions t_2, t_4, t_5, t_6 are controllable, it can be seen that the maximally permissive supervisor constructed in the sufficiency proof of Theorem 2 does not apply since critical transition t_3 is not controllable. However, since $t_4, t_5 \in T^{[t_3]}$, then we can still disable t_3 by control-disabling precritical transitions t_4 or t_5 before the firing of t_3. Note that the supervisor can control-disable either t_4 or t_5, since there is no controllable transition between t_4' (t_5') and t_3. Based on this observation, two liveness-enforcing supervisors can be derived as shown in Figs. 8 and 9.

Policy P_2

Configuration	Cut	Marking	Control - vector
∅	{ s1, s5, s6, s7, s9 }	[1000111010]^T	[1 1 1 0 1 11]^T
{ t5 }	{ s1, s4, s6, s7, s9 }	[1001011010]^T	[1 1 1 0 1 11]^T
{ t2 }	{ s2, s3, s5, s7, s9 }	[0110101010]^T	[1 1 1 0 1 01]^T
{ t6 }	{ s1, s5, s6, s7, s10 }	[1000111001]^T	[1 0 1 0 1 11]^T
{ t2, t5 }	{ s2, s3, s4, s7, s9 }	[0110101010]^T	[1 1 1 0 1 01]^T
{ t6, t5 }	{ s1, s4, s6, s7, s10 }	[1001011001]^T	[1 0 1 0 1 11]^T
Otherwise	/	/	[1 1 1 1 1 11]^T

Fig. 9. The second liveness-enforcing supervisor.

Configuration	Cut	Marking	Control - vector
∅	{ s1, s5, s6, s7, s9 }	[1000111010]^T	[1 1 1 1 1 11]^T
{ t4 }	{ s1, s5, s6, s8, s9 }	[1000110110]^T	[1 1 1 1 0 11]^T
{ t5 }	{ s1, s4, s6, s7, s9 }	[1001011010]^T	[1 1 1 0 1 11]^T
{ t2 }	{ s2, s3, s5, s7, s9 }	[0110101010]^T	[1 1 1 1 1 01]^T
{ t6 }	{ s1, s5, s6, s7, s10 }	[1000111001]^T	[1 0 1 1 1 11]^T
{ t2, t4 }	{ s2, s3, s5, s8, s9 }	[0110100110]^T	[1 1 1 1 0 01]^T
{ t6, t4 }	{ s1, s5, s6, s7, s10 }	[1000111001]^T	[1 0 1 1 0 11]^T
{ t2, t5 }	{ s2, s3, s4, s7, s9 }	[0110101010]^T	[1 1 1 0 1 01]^T
{ t6, t5 }	{ s1, s4, s6, s7, s10 }	[1001011001]^T	[1 0 1 0 1 11]^T
Otherwise	/	/	[1 1 1 1 1 11]^T

Fig. 10. The maximally permissive supervisory policy.

Given the controllability condition in Theorem 3. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two liveness-enforcing supervisors. It was proven in [11] that the following supervisor $\bar{\mathcal{P}}$ also enforces liveness:

$$\bar{\mathcal{P}}(m)_i = \begin{cases} \mathcal{P}_1(m)_i \wedge \mathcal{P}_2(m)_i, & \text{if } m \in R^{\mathcal{P}_1} \cap R^{\mathcal{P}_2}, \\ \mathcal{P}_1(m)_i, & \text{if } m \in R^{\mathcal{P}_1} - R^{\mathcal{P}_2}, \\ \mathcal{P}_2(m)_i, & \text{if } m \in R^{\mathcal{P}_2} - R^{\mathcal{P}_1}, \\ 1^m, & \text{if otherwise.} \end{cases} \quad (4)$$

This implies that a maximally permissive (minimally restrictive) liveness-enforcing supervisor exists if and only if there exists a liveness-enforcing supervisor. The maximally permissive supervisor for the example net in Fig. 1 can be derived based on the two supervisors in Fig. 8. Fig. 10 shows the maximally permissive liveness-enforcing supervisor.

VI. Conclusion

This paper provides a systematic way to verify and enforce the liveness of bounded ordinary Petri nets based on a partial order method called network unfolding. A salient feature of this paper is the intuition that the execution of the net system can be represented by the interleaving of its fundamental executions. Network unfolding helps extract fundamental executions and characterize their interleavings, thereby providing a computationally efficient way to verify and enforce liveness. Moreover, this paper uses the cut graph to verify the liveness of fundamental executions. A cut graph represents every fundamental execution as a node and encapsulates all the interleavings between fundamental executions into its arcs. A cut graph represents a higher level abstraction of system executions, since when deadlock is absent it suffices to check the sequential executions between fundamental executions to verify liveness. Another feature of our approach is the preservation of causality between net transitions. Causality is important for supervisor synthesis since it provides the optimal alternative when the critical transition is uncontrollable.

Future work involves extending these results to general verification and synthesis problems of Petri nets and applying this systematic approach to real world applications. One promising direction is to apply the net synthesis approach to distributed software systems. Results following this direction can be found in [26] and [27].

References

[1] K. X. He and M. D. Lemmon, "Liveness verification of discrete-event systems modeled by n-safe Petri nets," in *Proc. 21st Int. Conf. Application Theory Petri Nets*, Lecture Notes in Computer Science 1825, Nielsen and Simpson, Eds., June 2000, pp. 227–243.

[2] ——, "On the existence of liveness-enforcing supervisory policies of discrete-event systems modeled by n-safe Petri nets," in *Proc. 2000 IFAC Conf. Control Systems Design, Special Session Petri Nets*, Kozak and Huba, Eds., Bratislava, Slovak Republic, June 2000, pp. 323–328.

[3] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optimiz.*, vol. 25, no. 1, pp. 206–230, 1987.

[4] W. M. Wonham and P. J. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM J. Control Optimiz.*, vol. 25, no. 3, pp. 637–659, May 1987.

[5] F. Commoner, "Deadlocks in Petri nets," Applied Data Research, Inc., CA-7206-2311, 1972.

[6] H. Ridder and K. Lautenbach, "Liveness in bounded Petri nets which are covered by T-invariants," in *Lecture Notes in Computer Science*. New York: Springer-Verlag, 1994, vol. 815, pp. 358–375.

[7] K. Barkaoui, "Liveness of Petri nets and its relations with deadlocks, traps, and invariants," Laboratoire CEDRIC-CNAM, Paris, France, 92-06, 1995.

[8] K. Barkaoui and J. F. Pradat-Peyre, "On liveness and controlled Siphons in Petri nets," in *Proc. 17th Int. Conf. Application Theory Petri Nets*, vol. 1091, Lecture Notes in Computer Science, 1996, pp. 57–72.

[9] J. C. Corbett and G. S. Avrunin, "Using integer programming to verify general safety and liveness properties," *Formal Meth. Syst. Design: An Int. J.*, vol. 6, no. 1, pp. 97–123, Jan. 1995.

[10] P. Kemper and F. Bause, "An efficient polynomial-time algorithm to decide liveness and boundedness of free choice nets," in *Proc. 13th Int. Conf. Application Theory Petri Nets*, vol. 616, Lecture Notes in Computer Science, K. Jensen, Ed., 1992, pp. 263–278.

[11] R. S. Sreenivas, "On the existence of supervisory control in discrete event dynamic systems modeled by controlled Petri nets," *IEEE Trans. Automat. Contr.*, vol. 42, pp. 928–945, July 1997.

[12] ——, "On supervisory policies that enforce liveness in complete controlled Petri nets with directed cut-places and cut-transitions," *IEEE Trans. Automat. Contr.*, vol. 44, pp. 1221–1225, June 1999.

[13] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ: Prentice-Hall, 1981.

[14] J. Engelfriet, "Branching processes of Petri nets," *Acta Informatica*, vol. 28, pp. 575–591, 1991.

[15] K. McMillan, "Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits," in *Computer Aided Verification, Fourth Int. Workshop, CAV'92*, vol. 663, Lecture Notes in Computer Science, B. V. Bochmann and D. K. Probst, Eds., 1992, pp. 164–177.
[16] J. Esparza, "Model checking using net unfoldings," in *TAPSOFT'93: Theory Practice Software Development. 4th Int. Joint Conf. CAAP/FASE*, vol. 668, Lecture Notes in Computer Science, M. G. Gaudel and J. P. Jouannaud, Eds., 1993, pp. 613–628.
[17] A. Kondratyev, M. Kishinevsky, A. Taubin, and S. Ten, "Structural approach for the analysis of Petri nets by reduced unfoldings," in *Proc. 17th Int. Conf. Application Theory Petri Nets*, vol. 1091, Lecture Notes in Computer Science, 1996, pp. 346–365.
[18] W. Reisig, *Petri Nets*. Berlin, Germany: Springer-Verlag, 1985.
[19] T. Murata, "Petri nets: Properties, analysis, and applications," *Proc. IEEE*, vol. 77, pp. 541–580, Apr. 1989.
[20] J. Desel and J. Esparza, *Free Choice Petri Nets Cambridge Tracts in Theoretical Computer Science 40*. Cambridge, U.K.: Cambridge Univ.Press, 1995.
[21] A. Giua and F. Dicesare, "On the existence of Petri net supervisors," in *Proc.31st IEEE Int.Conf.Decision Control*, Tucson, AZ, pp. 3380–3385.
[22] Y. Li and W. M. Wonham, "Control of vector discrete-event systems I-the base model," *IEEE Trans.Automat.Contr.*, vol. 38, pp. 1214–1227, Aug. 1993.
[23] ——, "Control of vector discrete-event systems II-controller synthesis," *IEEE Trans.Automat.Contr.*, vol. 39, pp. 512–530, Mar. 1994.
[24] A. Semenov, "Verification and synthesis of asynchronous control circuits using petri net unfoldings," Ph.D. dissertation, 1998.
[25] K. McMillan, *Symbolic Model Checking*. Norwell, Massachusetts: Kluwer, 1993.
[26] M. D. Lemmon and K. X. He, "Supervisory plug-ins for distributed software," in *Workshop Proceedings for Software Engineering and Petri Nets*, M. Pezze and S. M. Shatz, Eds., Denmark, June 2000, DAIMI PB-548, pp. 155–172.
[27] K. X. He and M. D. Lemmon, "Dynamic reconfiguration of distributed software objects," in *Proc. 2000 IEEE Int. Conf. Systems, Man and Cybernetics*, Nashville, TN, Oct. 2000.

Kevin X. He (S'96–M'00) received the B.S. degree from Tsinghua University, Beijing, China, and the M.S. and Ph.D. degrees in electrical engineering, from the University of Notre Dame, Notre Dame, IN, all in electrical engineering, in 1995, 1998, and 2001, respectively.

From December 2000 to January 2002, he was a Member of the Technical Staff at Lucent Technologies' Mobility Solutions Group, Naperville, IL. His work at Lucent involved the design and implementation of base station controllers (BSC) of UMTS wireless networks. Currently, he serves as a Technical Advisor at the intellectual property law firm of Pennie & Edmonds, LLP, New York, NY. His current research interests comprise the general applications of discrete-event systems and, in particular, distributed software systems and wireless communication systems.

Michael D. Lemmon received the B.S. degree in electrical engineering from Stanford University, Stanford, CA, in 1979. After seven years as a Guidance and Control Systems Aerospace Engineer, he received the M.S. and Ph.D. degrees in electrical and computer engineering from Carnegie-Mellon University, Pittsburgh, PA, in 1987 and 1990, respectively.

Since 1990, he has been affiliated with the University of Notre Dame, Notre Dame, IN, where he currently holds an appointment as Associate Professor of Electrical Engineering. He served as program chair for the 5th International Workshop on Hybrid Systems in 1997, and the 1999 International Symposium on Intelligent Control. His research interests are in high-autonomy control systems, networked control systems, supervisory control, hybrid dynamical systems theory, and the application of control technology to real-time software engineering.

Dr. Lemmon is a former Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS, and is currently an Associate Editor for the IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY. Between 1996 and 2000, he chaired the IEEE working group on hybrid dynamical systems.