At the Intersection of Computing- and Control-Theory: A Tutorial on Liveness Enforcing Supervisory Policies for Arbitrary Petri Nets*

Arun Raman[1] and R.S. Sreenivas[1]

***Abstract*— We present a tutorial-introduction to the synthesis of *Liveness Enforcing Supervisory Policies* (LESPs) for *Discrete-Event Dynamic Systems* (DEDS) modeled by *Petri-Nets* (PNs). The tutorial is aimed at researchers unfamiliar with the area, and the objective is to develop a working knowledge of concepts and results from the area. We start by introducing PNs as a modeling tool for DEDS. We motivate the *liveness problem* through examples. Following this, we present relevant theoretical results on the computational aspects of LESP synthesis for PN models of DEDS, along with other results and methods. We conclude the paper by listing some new directions in the area.**

I. Introduction and Motivation

A *Discrete Event Dynamic System* (DEDS) is a discrete-state system, where the discrete-state changes at discrete-time instants due to the occurrence of *events*. The occurrence of events requires certain conditions or constraints to be satisfied in the system. The conditions for an event, in several cases, are naturally dependent on the availability of resources. For example, in a queuing system, where a person leaving the queue and entering service is an event, the change in queue length is subject to the availability of servers (resources). On the other hand, in many other cases, an event can be abstractly interpreted as being dependent on availability of resources. For example, in a program, if executing the statements inside an *IF* loop is an event, then the condition required by the *IF* expression can be interpreted as the conditions that are required for the occurrence of the event. The occurrence of an event in a DEDS creates new conditions that results in a different set of events that could occur, and this process can repeat as often as necessary.

One of the ways of modeling such *condition-event* systems is by *Petri Nets* (PNs) [1], [2]. PNs are directed bipartite graphs; in which the two classes of nodes are *transitions* (which correspond to *events*) and *places* (which correspond to *conditions*). The directed arcs connecting places to transitions, capture the conditions that must be met for an event (a transition) to be (resp. state-) enabled. Similarly, the directed arcs connecting transitions to places encode the consequence of the event that is represented by the transition. When the prerequisite conditions to an event are satisfied, we say the (transition representing the) event is *enabled*. The occurrence of the event is captured by the *firing* of the transition that represents it. It is not imperative that an enabled event (transition) *does* occur (fire) every time. The firing of an enabled-

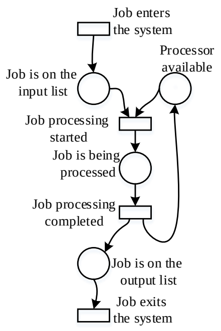

Fig. 1: An illustrative PN model of a simple computer system

transition (occurrence of an event) subtracts an appropriate number of (integer-valued) *tokens* from each of its input places, and adds an appropriate number of tokens to its output places. The specifics of how many tokens are removed from (added to) the input (output) place(s) is determined by the arc weights connecting the nodes. In its graphical representation, transitions are represented by rectangles and places by circles, along with weighted-arcs that go from places to transitions, or transitions to places. The integer-valued tokens associated with each place are represented by filled-circles that reside within the circles representing each place.

PNs are a popular modeling formalism for DEDS because they provide abundant structural information about the system, and they are amenable to mathematical analysis. Manufacturing- and Service-Systems; Database-Systems; Traffic-Networks; Integrated Command, Control, Communication and Information (C^3I) systems; etc., are examples of DEDS which have been modeled and analyzed by PNs. Fig. 1 presents a PN model of the flow of jobs in a computer system ([2]). If there is a job to be processed and if the processor is available, then the event start job processing occurs, and one of the processors is taken up by the job. The processor moves to the idle state and becomes available as soon as the job processing is completed.

Fig. 2 shows a PN model of a weak multiplier ([2]); which is an example where an event has an abstract interpretation. The initial number of tokens in places p_x and p_y is equal to the value of the two quantities, x and y respectively, that are to be multiplied. Place $p_{x \cdot y}$ stores the output of the multiplication. Place p_1 acts like an *enable* place and the multiplication operation will begin only if it has a token. The nominal operation of the net is as follows. At the initial

*This work was supported in part by the *Arthur Davis Faculty Scholar Endowment* at the University of Illinois at Urbana-Champaign.

[1]Coordinated Science Laboratory & Industrial and Enterprise Systems Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA {raman12, rsree}@illinois.edu

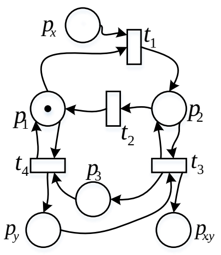

Fig. 2: Petri-Net model of a weak multiplier

condition, since only places p_x, p_y and p_1 have tokens, only transition t_1 is enabled; which upon firing, moves one token from p_x to p_2. Transition t_3 is now enabled as p_2 and p_y have tokens. Firing of t_3 can copy y tokens from place p_y, putting them in p_3 and $p_{x \cdot y}$, the output places of t_3. Now, t_2 can fire, moving the token from p_2 back to p_1. This enables t_4, which can copy the y tokens from p_3 back into p_y, without losing the token in p_1. This entire process can be repeated exactly x times, each time putting y tokens in $p_{x \cdot y}$. after which the marking of place p_x is reduced to zero, and the net must stop. The total number of tokens in place $p_{x \cdot y}$ is then the product of x and y. This is the best case scenario as the number of output tokens is exactly $x \cdot y$. Since t_3 can fire no more than x times, we can guarantee that the number of tokens in $p_{x \cdot y}$ never exceeds $x \cdot y$. However, the token in p_2 enables both transitions t_3 and t_2, and it is possible for t_2 to fire before all y tokens have been copied from p_y to p_3, and added to $p_{x \cdot y}$. In this case, the number of tokens in $p_{x \cdot y}$ will be less than $x \cdot y$. Hence, this model simulates the multiplication operation in a weak sense.

In the rest of this section, we will motivate the liveness problem in PNs through examples. Consider the PN N_1 of Fig. 3. Transitions t_1, t_2, t_3 and t_5 need at least two tokens in their respective inputs places to be enabled, while transitions t_4 and t_6 are enabled if their input places have at least one one token. The firing of t_1 and t_2 places three tokens in their output places (p_2 and p_1 respectively). N_1 has four places, and we represent the *marking* (state) of the net as a non-negative, integral-vector of size four. An initial token load of (2 0 0 0) means that p_1 has two tokens whilst there are zero tokens in all other places. The firing of transition t_3 from this initial state removes two tokens from p_1 and adds a token to place p_3. We represent this state-event dynamic as: (2 0 0 0) $\xrightarrow{t_3}$ (0 0 1 0). If t_4 fires next, we reach the state (0 2 0 0). We can represent the result of the firing of t_3 and t_4 together as a string: (2 0 0 0) $\xrightarrow{t_3 t_4}$ (0 0 1 0). Note that $t_4 t_3$ is not a valid firing string from (2 0 0 0) as t_4 can only fire if there is at least one token in p_3. Now, consider the string: (2 0 0 0) $\xrightarrow{t_3 t_4 t_5 t_6}$ (1 0 0 0). It is easy to see that none of the transitions are enabled at (1 0 0 0). Consequently, none of the transitions can fire at this marking and the system is now in, what is commonly called, a *hanged* state.

This is formally defined as follows. A DEDS is said to

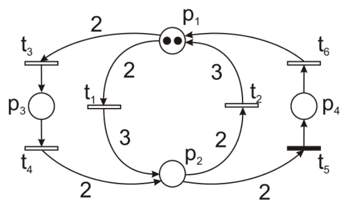

Fig. 3: Example 1

be in a *livelocked* state if there is at least one activity that can never be completed. This is different from a *deadlocked* state in which all the activities of the system cannot be completed. A PN is said to be *live* if it is possible to fire any transition, although not necessarily immediately, from any marking that is reachable from the *initial marking*. From a control-theoretic point of view, it is of interest to analyze the liveness property of DEDS modeled by PNs. The framework partitions the set of transitions into controllable transitions and uncontrollable transitions. The supervisory policy enforces liveness by preventing the firing of a subset of controllable transitions (which correspond to controllable activities/events). The complementary set of uncontrollable transitions represent uncontrollable activities/events, that cannot be influenced by the policy in any way. It is to be noted that a supervisory policy can only prevent the firing of a controllable transition, it cannot force a transition to fire. This has the potential of complicating the behavioral analysis of PN models, as it may require an exhaustive enumeration of all possible sequence of transition-firings before any determination can be made.

The class of supervisory policies, which are basically control algorithms, considered in this article are (binary-valued) functions that take the marking of the system as input and output a 0 (enable) or a 1 (disable) for every controllable transition. When the supervisory policy enables a transition, we say that the transition is control-enabled. For completeness, we say that an uncontrollable transition is always control-enabled. A transition can fire only if it is state- and control-enabled. We do not make any other assumption— the (state- and control-) enabled transitions can fire at any rate, in any sequence and at any time. In the graphical representation of PNs, controllable (uncontrollable) transitions are represented by filled (unfilled) rectangles. In the remainder of this section, we will systematically try to come up with a procedure to synthesize an LESP for an arbitrary PN.

Coming back to Fig. 3, t_5 is the only controllable transition. In the preceding discussion, we identified (1 0 0 0) as one of the livelocked states. For this particular example, we can reverse-engineer an LESP using this livelocked state. The net reached this marking because the firing of t_5 reduced the token load of the system to a level that t_1 and t_3 are not enabled anymore. We look to avoid reaching such a state. With careful eye-balling, we see that the loops $p_1 t_1 p_2 t_2 p_1$ and $p_1 t_3 p_3 t_4 p_2 t_2 p_1$ increase the token load of the system, whereas $p_1 t_1 p_2 t_5 p_4 t_6 p_1$ and $p_1 t_3 p_3 t_4 p_2 t_5 p_4 t_6 p_1$ reduce the token load of the system. An obvious LESP is to first increase the token load of the system to a high enough level, and then control-enable t_5 so that reduction by the $t_5 p_4 t_6$ branch does not render it livelocked. In fact, this supervisory policy will enforce liveness on any initial marking that is larger than or equal to $\{(2\ 0\ 0\ 0), (0\ 0\ 1\ 0), (0\ 2\ 0\ 0), (0\ 0\ 0\ 2)\}$. Moreover, there does not exist an LESP for an initial marking that is smaller than any of these elements. This observation is in line with a common intuition that more is better. However, this is not necessarily true for liveness. Consider the net N_2 in Fig. 4. The net can be made live for an initial marking of (0 0 0 0 1). But if the initial marking is (0 0 0 0 2), then the uncontrollable transition t_6 is enabled, the firing of which would empty the tokens. Now, transition t_1 of N_2 takes in one token and generates three, that is, it increases the token load of the system. On the other hand, transition t_6 decreases the token load of N_2. As with the net N_1, one possible LESP for N_2 would be to prevent putting even number of tokens in p_5 until there are sufficient number of tokens in places p_1 to p_4.

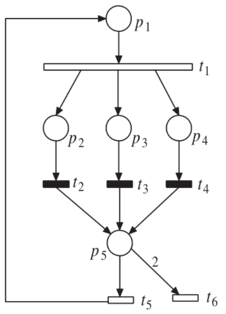

Fig. 4: Example 2

Consider the net N_3 in Fig. 5. N_3 does not have a branch that increases or decreases the token load of the system. It is clear that the main bottleneck in enforcing liveness in N_3 is transition t_4. The LESP should sequence the firing of t_1 and t_2 such that both the input places of t_4, p_3 and p_5, can receive tokens. Based on the three examples, we can loosely formulate a method for synthesizing an LESP as follows:

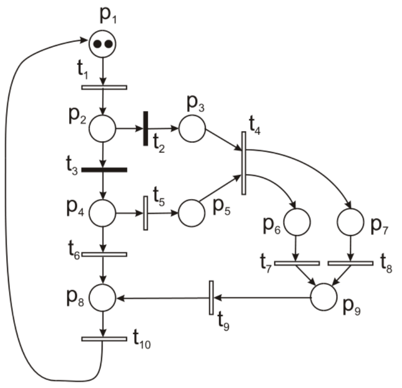

Fig. 5: Example 3

1) Identify the permissible states and states-to-avoid. We use $\Delta(N)$ (formally defined in the next section) to characterize the set of permissible-states.
2) Characterize the boundary between the permissible-states and the states-to-avoid.
3) Synthesize a policy that constrains the system to permissible-states.

Using the above steps (or otherwise), we intend to develop a procedure/algorithm that takes a general PN structure as input and gives the LESP (in some form) as the output. This procedure does not assume any structure of the net— there can be as many places, transitions and arcs, connected in an arbitrary way in the net. We introduce the formal notations and definitions in the next section.

II. Notations and Definitions

We use $\mathcal{N}$ ($\mathcal{N}^+$) to denote the set of non-negative (positive) integers. A *Petri net structure* $N = (\Pi, T, \Phi, \Gamma)$ is an ordered 4-tuple, where $\Pi = \{p_1, \ldots, p_n\}$ is a set of n *places*, $T = \{t_1, \ldots, t_n\}$ is a collection of m *transitions*, $\Phi \subseteq (\Pi \times T) \cup (T \times \Pi)$ is a set of *arcs*, and $\Gamma : \Phi \rightarrow \mathcal{N}^+$ is the *weight* associated with each arc. The *initial marking function* (or the *initial marking*) of a PN structure N is a function $\mathbf{m}^0 : \Pi \rightarrow \mathcal{N}^n$, which identifies the number of *tokens* in each place. We will use the symbol $N(\mathbf{m}^0)$ to denote a PN structure N along with its initial marking $\mathbf{m}^0$.

We define the sets $^\bullet x := \{y | (y, x) \in \Phi\}$ and $x^\bullet := \{y | (x, y) \in \Phi\}$. A transition $t \in T$ is said to be enabled at a marking $\mathbf{m}^i$ if $\forall p \in {^\bullet t}, \mathbf{m}^i(p) \geq \Gamma(p, t)$. The set of enabled transitions at marking $\mathbf{m}^i$ is denoted by the symbol $T_e(N, \mathbf{m}^i)$. An enabled transition $t \in T_e(N, \mathbf{m}^i)$ can fire, which changes the marking $\mathbf{m}^i$ to $\mathbf{m}^{i+1}$ according to $\mathbf{m}^{i+1}(p) = \mathbf{m}^i(p) - \Gamma(p, t) + \Gamma(t, p)$. Given an initial marking $\mathbf{m}^0$, the set of *reachable markings* for $\mathbf{m}^0$, which is denoted by $\Re(N, \mathbf{m}^0)$, is defined as the set of markings generated by all valid firing strings starting with marking $\mathbf{m}^0$ in the PN N.

A supervisory policy $\mathcal{P} : \mathcal{N}^n \times T \rightarrow \{0, 1\}$, is a function that returns a 0 or 1 for each reachable marking and each transition. The supervisory policy $\mathcal{P}$ permits the firing of transition t_j at marking $\mathbf{m}^i$, if and only if $\mathcal{P}(\mathbf{m}^i, t_j) = 1$. If $t_j \in T_e(N, \mathbf{m}^i)$ ($\mathcal{P}(\mathbf{m}^i, t_j) = 1$) for some marking $\mathbf{m}^i$, we say the transition t_j is state-enabled (control-enabled) at $\mathbf{m}^i$. A transition has to be state- and control-enabled before it can fire. To reflect the fact that the supervisory policy does not control-disable any uncontrollable transition, we assume that $\forall \mathbf{m}^i \in \mathcal{N}^n, \mathcal{P}(\mathbf{m}^i, t_j) = 1$, if $t_j \in T_u$.

A transition t_k is *live* under the supervision $\mathcal{P}$ if $\forall \mathbf{m}^i \in \Re(N, \mathbf{m}^0, \mathcal{P}), \exists \mathbf{m}^j \in \Re(N, \mathbf{m}^i, \mathcal{P})$ such that $t_k \in T_e(N, \mathbf{m}^j)$ and $\mathcal{P}(\mathbf{m}^j, t_k) = 1$. A policy $\mathcal{P}$ is a *liveness enforcing supervisory policy* (LESP) for $N(\mathbf{m}^0)$ if all transitions in $N(\mathbf{m}^0)$ are live under $\mathcal{P}$. The set $\Delta(N) = \{\mathbf{m}^0 : \exists$ an LESP for $N(\mathbf{m}^0)\}$ represents the set of initial markings for which there is an LESP for a PN structure N.

Next we briefly introduce some concepts in computability-theory. A decision-problem is a problem that is posed as a ”yes” or “no” question for the input values. An undecidable problem is a decision problem for which there does not exist a single algorithm that correctly answers "yes" or "no" to all possible inputs. This also means that there do not exist finite number of algorithms that can correctly solve the decision problem for input-sets that are unique to themselves; because if this were true, then the finitely many algorithms could be combined to create a single algorithm (based on the inputs for which they work), which can be used to correctly answer "yes" or "no" for all possible inputs. The *Halting Problem* is the most fundamental of all (undecidable) decision problems. The halting problem is the problem of determining, from a description of an arbitrary computer program and an input, whether the program will halt or continue to run forever. Informally, the undecidability of the halting problem stems from the fact that there is no way to ascertain if a program that has not yet halted is in a state of suspended-animation for perpetuity, from an instance where the same program is progressing towards completion. For additional details we refer the reader to Cutland's text [3].

III. Some Computability-theoretic Results

An integral-set of markings $\mathcal{M} \subseteq \mathcal{N}^n$ is said to be *control-invariant* with respect to a PN structure N, if the firing of any enabled uncontrollable transition at a marking in $\mathcal{M}$ in N results in a new marking that is also in $\mathcal{M}$.

Theorem 5.3 of [4] proved the necessary and sufficient condition for the existence of an LESP, which we state without proof.

Theorem 1: There exists an LESP for $N(\mathbf{m}^0)$ if and only if $\mathbf{m}^0 \in \Delta(N)$, where $\Delta(N)$ is the set of initial markings that satisfy the following properties:

1) $\Delta(N)$ is control-invariant.
2) $\forall \mathbf{m}^1 \in \Delta(N), \exists \mathbf{m}^2, \mathbf{m}^3 \in \Delta(N), \exists$ a valid firing sequence $\sigma = \sigma_1\sigma_2$ in N such that $\mathbf{m}^1 \xrightarrow{\sigma_1} \mathbf{m}^2 \xrightarrow{\sigma_2} \mathbf{m}^3 \geq \mathbf{m}^2$ such that all transitions appear in σ_2 at least once and $\forall \sigma_3 \in pr(\sigma_1\sigma_2)$, $\mathbf{m}^1 \xrightarrow{\sigma_3} \mathbf{m}^4 \Rightarrow \mathbf{m}^4 \in \Delta(N)$. Here $pr(\bullet)$ denotes the prefix.

However, the above conditions were shown to be untestable. Consider the net $\widetilde{N}$ in Fig. 6. Subnets N_1 and N_2 have $\mathbf{m}_1^0$ and $\mathbf{m}_2^0$ as their initial markings respectively. Besides these subnets, place π_1 has one token. All other places have zero tokens.

Theorem 2: $(\widetilde{N}(\mathbf{m}^0)$ is live$) \Leftrightarrow (\Re(N_1, \mathbf{m}_1^0) \subseteq \Re(N_2, \mathbf{m}_2^0))$.

Proof: (*Idea*) The liveness of τ_1 can be guaranteed iff the token load of π_1 is repeatedly replenished. But since the initial marking is such that $\widetilde{\mathbf{m}}^0(\pi_1) = 1$, $^\bullet\pi_1 = \tau_{n+4}$ and $\widetilde{\mathbf{m}}^0(\pi_{n+4}) = 0$, the token load of π_1 can be repeatedly replenished iff the token at place π_1 is safely passed on to π_{n+4}. In fact, the whole net is live once a token is placed in π_{n+4} as $\tau_{n+4}^\bullet = \widetilde{\Pi}$. Therefore, the presence of a token in π_{n+4} is a necessary and sufficient condition for liveness of $\widetilde{N}$. Now the token can be passed to π_{n+4} if and only if it is not lost at π_{n+3} by firing of the transitions $\widetilde{\tau}_j^i$, ($i = 1, 2$ and $j = 1, 2, \ldots n$). This can be prevented iff all the tokens of N_1 and N_2 are sinked through transitions $\hat{\tau}_j^i$, ($i = 1, 2$ and $j = 1, 2, \ldots n$). This is possible if and only if $N_1(\mathbf{m}_1^0)$ and $N_2(\mathbf{m}_2^0)$ reach the exact same marking, which is true iff $\Re(N_1, \mathbf{m}_1^0) \subseteq \Re(N_2, \mathbf{m}_2^0)$. ■

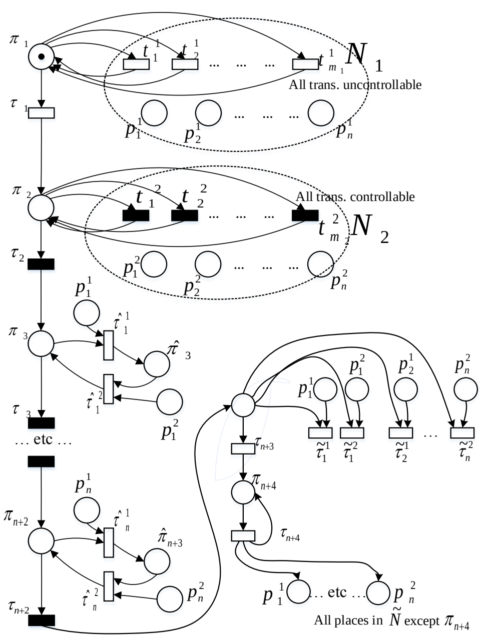

Fig. 6: $\widetilde{N}$ constructed from N_1 and N_2

Theorem 5.3 of [4] presents a rigorous proof of Theorem 2. In what follows, we briefly discuss the proof that determining if $\Re(N_1, \mathbf{m}_1^0) \subseteq \Re(N_2, \mathbf{m}_2^0)$ is not semi-decidable. The reduction argument is based on the fact that *Hilbert's tenth problem* is unsolvable.

Definition 1: Hilbert's tenth problem: Given a polynomial, P, over n variables with integer coefficients, does there exist a vector of integers $(x_1, x_2, \ldots, x_n)$ such that $P(x_1, x_2, \ldots, x_n) = 0$?

The equation $P(x_1, x_2, \ldots, x_n) = 0$ is called the *Diophantine Equation*. Matiyasevich [5] proved that there is no general algorithm to determine if an arbitrary Diophantine equation has a root or not. We will use this result as the basis of our proof. The idea is to construct a net that can compute, in some sense, all solutions of an arbitrary Diophantine polynomial. We do it in two steps— first by reducing Hilbert's tenth problem to the polynomial graph inclusion problem and then by reducing the polynomial graph inclusion problem to the PN reachability inclusion problem. In the next few paragraphs, we discuss the first of the two reductions in reasonable detail, and present the main idea of the second reduction. The complete proof is given in [2]. We use x to denote the vector $(x_1, \ldots, x_n)$.

Definition 2: The graph $G(P)$ of a Diophantine polynomial $P(x)$ with nonnegative coefficients is the set

$$G(P) = \{(x, y) \mid P(x) \geq y \text{ with } x, y \geq 0\} \quad (1)$$

Definition 3: The Polynomial Graph Inclusion Problem is to determine for two Diophantine polynomials A and B if $G(A) \subseteq G(B)$.

Without loss of generality, we only consider non-negative polynomials with non-negative solutions. We split $P(x)$ into two polynomials, $A(x)$ and $B(x)$ such that $P(x) = A(x) - B(x)$, by putting all the terms with positive coefficients in $A(x)$ and all terms with negative coefficients in $B(x)$. Since $P(x) \geq 0$, we have $A(x) \geq B(x)$. Consider the following two polynomial graphs:

$$G(A) = \{(x, y) \mid A(x) \geq y\}$$
$$G(1 + B) = \{(x, y) \mid 1 + B(x) \geq y\}$$

Now, $G(1 + B) \subseteq G(A)$ if and only if for all nonnegative x and y, $1 + B(x) \geq y \Rightarrow A(x) \geq y$. This is true if and only if there *does not exist* x and y such that:

$$A(x) < y \leq 1 + B(x)$$

But we also have $A(x) \geq B(x)$.

$$A(x) < y \leq 1 + B(x) \leq 1 + A(x)$$

Since all quantities are integers:

$$y = 1 + B(x) = 1 + A(x)$$

which is true if and only if $A(x) = B(x)$, that is $P(x) = 0$. Therefore, $G(1 + B) \subseteq G(A)$ if and only if there does not exist x such that $A(x) = B(x)$. To determine that the equation $P(x) = 0$ has a solution, we need only to show that it is not the case that $G(1 + B) \subseteq G(A)$. This completes the first reduction.

The second step in the proof is reducing the polynomial graph inclusion problem to the PN reachability inclusion problem. Recall the weak multiplier in Fig. 2. Observe that a weak multiplier is nothing but a polynomial graph where the actual value in place $p_{x \cdot y}$ is equivalent to the y of Equation 1. The main idea of the reduction-proof is that a chain of weak multipliers can be used to simulate a graph of an arbitrary polynomial. Then after some technical adjustments (like equating the number of places in both nets etc.) $G(A)$ and $G(1+B)$ become equivalent to $\Re(N_1, \mathbf{m}_1^0)$ and $\Re(N_2, \mathbf{m}_2^0)$ respectively. The reader is referred to [2] for the complete rigorous proof.

Then from Theorem 2 we have the following result [6], which is a refinement of the result in [4]:

Theorem 3: "Is $\mathbf{m}^0 \in \Delta(N)$?" and "Is $\mathbf{m}^0 \notin \Delta(N)$?" are not semi-decidable.

Theorem 3 proves that testing if there is an LESP for a *particular marking* is not semi-decidable. The next question is to investigate if there is any marking at all for which an LESP exists. The following theorem proves that it is not semi-decidable [7]:

Theorem 4: "Is $\Delta(N) = \emptyset$?" and "Is $\Delta(N) \neq \emptyset$?" are not semi-decidable.

Philosophically, the caveat in testing the necessary and sufficient conditions given in Theorem 1 is that the conditions are self referential— to determine if a marking is in $\Delta(N)$, we need information about $\Delta(N)$. That is, we do not have a starting-point for algorithmically constructing $\Delta(N)$. The next natural direction of research is to identify nets for which we can characterize the set $\Delta(N)$, and thus enabling us to test the conditions in 1. [6], [8] proved that the existence of an LESP is decidable for PN structures that belong to certain (testable) classes of PNs. These observations hold for the family of PN structures known as $\mathcal{H}$-class in the literature, which is identified by the following structural properties: (1) for each place, the weights associated with the outgoing arcs that terminate on uncontrollable transitions must have the smallest of all outgoing arc-weights; (2) the set of input places to each uncontrollable transition is no larger than the set of input places of any transition which shares a common input place with it. The set $\Delta(N)$ for these classes of PNs is right-closed. A set of markings $\mathcal{M} \in \mathcal{N}^n$ is said to be right-closed if $((\mathbf{m}^1 \in \mathcal{M}) \wedge (\mathbf{m}^2 \geq \mathbf{m}^1) \Rightarrow (\mathbf{m}^2 \in \mathcal{M}))$, and is uniquely defined by its finite set of minimal elements. However, there are nets that have a right-closed $\Delta(N)$ and do not belong to the $\mathcal{H}$-class of PN structures. As an illustration, consider the net N_1 in Fig. 7. The outgoing arcs of place p_1 violate the $\mathcal{H}$-class restriction. It can be verified that $\Delta(N_1)$ is right-closed ([9]). Is it the particular structure of $\Delta(N)$ that makes the conditions in Theorem 1 testable? If yes, the next question is to determine if $\Delta(N)$ is right-closed or not for arbitrary nets. Irrespective of the answer to the above question, the following two theorems prove that it is not semi-decidable to determine if $\Delta(N)$ is right-closed or to even determine if $\Delta(N)$ has a right-closed subset [7].

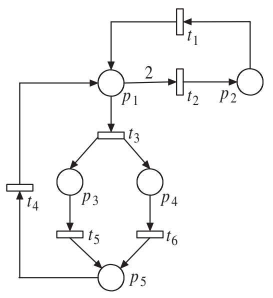

Fig. 7: A PN structure $N_1 \notin \mathcal{H}$ for which $\Delta(N_1)$ is right-closed.

Theorem 5: "Is $\Delta(N)$ right-closed" and "Is $\Delta(N)$ not right-closed" are not semi-decidable.

Theorem 6: "Is there a right-closed subset of $\Delta(N)$?" and "Is there no right-closed subset of $\Delta(N)$?" are not semi-decidable.

However, if we constrain the problem even further and assume a particular nature of the LESP, then the problem becomes decidable. An LESP $\mathcal{P}$ for a PN $N(\mathbf{m}^0)$ is said to be a *marking-monotone LESP* if it is also an LESP for $N(\widehat{\mathbf{m}}^0)$ for any $\widehat{\mathbf{m}}^0 \geq \mathbf{m}^0$, as well.

Theorem 7: The existence of a Marking-Monotone LESP is decidable for an arbitrary PN [7].

It is to be noted that for nets belonging to $\mathcal{H}$-class, a marking-monotone LESP is the maximally permissive (or minimally restrictive) LESP for that net. If a minimally restrictive LESP prevents the firing of a state-enabled transition at a marking that is reachable under its supervision, then *all* LESPs would do the same.

IV. Other Results in Literature

We present a brief review of the literature pertinent to the synthesis of *liveness enforcing supervisory policies* (LESPs) for different classes of PNs. Giua [10] introduced *monitors* to supervisory control of PNs. Moody and Antsaklis [11] used monitors to enforce liveness in certain classes of PNs. This work was extended by Iordache and Antsaklis [12] to include a sufficient condition for the existence of policies that enforce liveness in a class of PNs called *Asymmetric Choice Petri nets*. Reveliotis et al. used the *theory of regions* to identify policies that enforce liveness in *Resource Allocation Systems* [13]. Ghaffari, Rezg and Xie [14] used the theory of regions to obtain a *minimally restrictive* supervisory policy that enforces liveness for a class of PNs. Marchetti and Munier-Kordon [15] presented a sufficient condition for liveness, that can be tested in polynomial time, for a class of general PNs known as *Unitary Weighted Event Graphs*. Basile et al. [16] presented sufficient conditions for minimally-restrictive, closed-loop liveness of a class of *Marked Graph* PNs supervised by monitors that enforce *Generalized Mutual Exclusion Constraints* (GMECs). [17] presented a necessary and sufficient condition for the existence of GMECs that enforces, among other things, liveness, in a bounded PN. [18] investigated simplified *linear and nonlinear classifiers* that are maximally permissive for deadlock avoidance in resource allocation systems. Chen and Li [19] used the vector covering approach to reduce the sets of legal markings and *first met bad* markings, resulting in a maximally permissive control policy for flexible manufacturing systems.

V. Discussion and Open Problems

The paper laid out, in a tutorial manner, the landscape of literature in the area of synthesis of LESPs for arbitrary PNs. The first half of the paper was devoted to getting a feel of PNs and the liveness problem with the help of examples. The second half of the paper underlined some of the fundamental computability-theoretic results from literature in the area of LESP synthesis. Some proofs were also presented in this tutorial with an objective of developing an intuition for the solution. Rigorous versions of the proofs could be found in the respective references. We end the paper by listing some new areas of research:

1) Identifying further subclasses of nets and/or characterizing LESPs for which the existence of an LESP is decidable.
2) The liveness property of a system can be roughly interpreted as the *excitability* of all modes of a system. An investigation into continuous-time approximations of PNs ([20]) and switched linear models might potentially build a bridge between the DEDS theory and that of hybrid systems.
3) Robust LESPs: The LESPs synthesized in the literature assume perfect information about the state of the system. How will the policies change if the supervisor does not have perfect information about the system?
4) Connection with Combinatorial Game Theory: The uncontrollable transitions can be interpreted as adversaries which try to take the system to (appropriately defined) unsafe states, while the controllable transitions have to fire in such a way so as to remain in the (appropriately defined) safe states. Can a PN (or any DEDS model for that matter) be interpreted as a board game? Such a bridge, apart from giving insights, would enable the use of tools from Combinatorial Game Theory in DEDS theory.

References

[1] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[2] J. L. Peterson, "Petri net theory and the modeling of systems," 1981.

[3] N. Cutland, *Computability: An Introduction to Recursive Function Theory*. Cambridge, UK: Cambridge University Press, 1980.

[4] R. S. Sreenivas, "On the existence of supervisory policies that enforce liveness in discrete-event dynamic systems modeled by controlled petri nets," *IEEE Transactions on Automatic Control*, vol. 42, no. 7, pp. 928–945, 1997.

[5] Y. Matiyasevich, *Hilbert's 10th Problem*. Cambridge, MA: MIT Press, 1993.

[6] R. Sreenivas, "On the existence of supervisory policies that enforce liveness in partially controlled free-choice petri nets," *IEEE Transactions on Automatic Control*, vol. 57, no. 2, pp. 435–449, 2012.

[7] C. Chen, A. Raman, H. Hu, and R. S. Sreenivas, "On liveness enforcing supervisory policies for arbitrary petri nets," *submitted, IEEE Transactions on Automatic Control*.

[8] N. Somnath, *On computing a liveness enforcing supervisory policy for a class of general Petri nets*. University of Illinois at Urbana-Champaign, 2015.

[9] S. Chandrasekaran, N. Somnath, and R. Sreenivas, "A software tool for the automatic synthesis of minimally restrictive liveness enforcing supervisory policies for a class of general petri net models of manufacturing-and service-systems," *Journal of Intelligent Manufacturing*, vol. 26, no. 5, pp. 945–958, 2015.

[10] A. Giua, "Petri nets as discrete event models for supervisory control," *Rensselaer Polytechnic Institute, Troy, NY*, 1992.

[11] J. O. Moody and P. J. Antsaklis, *Supervisory control of discrete event systems using Petri nets*. Springer Science & Business Media, 2012, vol. 8.

[12] M. Iordache and P. J. Antsaklis, *Supervisory control of concurrent systems: a Petri net structural approach*. Springer Science & Business Media, 2007.

[13] S. A. Reveliotis, E. Roszkowska, and J. Y. Choi, "Generalized algebraic deadlock avoidance policies for sequential resource allocation systems," *IEEE Transactions on Automatic Control*, vol. 52, no. 12, pp. 2345–2350, 2007.

[14] A. Ghaffari, N. Rezg, and X. Xie, "Design of a live and maximally permissive petri net controller using the theory of regions," *IEEE transactions on robotics and Automation*, vol. 19, no. 1, pp. 137–141, 2003.

[15] O. Marchetti and A. Munier-Kordon, "A sufficient condition for the liveness of weighted event graphs," *European Journal of Operational Research*, vol. 197, no. 2, pp. 532–540, 2009.

[16] F. Basile, L. Recalde, P. Chiacchio, and M. Silva, "Closed-loop live marked graphs under generalized mutual exclusion constraint enforcement," *Discrete Event Dynamic Systems*, vol. 19, no. 1, pp. 1–30, 2009.

[17] F. Basile, R. Cordone, and L. Piroddi, "Integrated design of optimal supervisors for the enforcement of static and behavioral specifications in petri net models," *Automatica*, vol. 49, no. 11, pp. 3432–3439, 2013.

[18] R. Cordone, A. Nazeem, L. Piroddi, and S. Reveliotis, "Designing optimal deadlock avoidance policies for sequential resource allocation systems through classification theory: existence results and customized algorithms," *IEEE Transactions on Automatic Control*, vol. 58, no. 11, pp. 2772–2787, 2013.

[19] Y. Chen and Z. Li, "Design of a maximally permissive liveness-enforcing supervisor with a compressed supervisory structure for flexible manufacturing systems," *Automatica*, vol. 47, no. 5, pp. 1028–1034, 2011.

[20] R. David and H. Alla, "Timed continuous petri nets," in *Discrete, Continuous, and Hybrid Petri Nets*. Springer, 2010, pp. 159–229.