Some Results on Petri Net Languages

STÉPHANE LAFORTUNE AND HYUCK YOO

Abstract—We compare various classes of Petri net languages. We present a new constructive proof of the equivalence, from the point of view of descriptive power, of “general” Petri nets and “restricted” Petri nets (no multiple arcs nor self-loops are allowed in the latter class). We also comment on the descriptive power of Petri nets versus that of finitely recursive processes.

I. INTRODUCTION

Finite-state machines (FSM), Petri nets (PN), and finitely recursive processes (FRP) are three classes of logical discrete event models that are currently being used for the modeling, analysis, and control of discrete event systems (DES). Ramadge and Wonham have proposed a control theory for DES based on controlled state machines and formal languages [8]. Petri nets have been widely used as a modeling tool for DES (see the references in [1]); Krogh et al. have discussed the control of PN in [6] and [4]. More recently, Inan and Varaiya have extended the work of Hoare [3] and proposed the FRP model for DES [5]. Each of these models has its own merits and drawbacks. From the point of view of defining the language generated by a PN and three different labelings of transitions, the class of general PN (GPN) allows such occurrences. Concerning PNL, Peterson presents four different ways of defining the language generated by a PN and three different labelings of transitions, resulting in 12 types of PNL [7]!

The purpose of this note is to clarify the relationships between some important classes of PNL. In particular, concerning languages generated by GPN and RPN, PNL have been studied in [7] and [2]. In [7], GPN and RPN are proved equivalent for languages that allow ε-labeling of transitions (ε is the empty string) but not for “ε-free” languages. Hack considered many properties of PNL in [2]. However, his proof technique for the problem that we address in Section III is not constructive but indirect, and we have found that it is incomplete concerning the elimination of “bounded closed subsets with multiple arcs” [2, p. 72]. In view of these observations, we provide in Section III a new constructive proof of the equivalence of GPN and RPN for languages that do not allow ε-labeling of transitions, thus generalizing existing results on this topic.

Finally, in Section IV, we compare PNL to the sets of traces generated by FRP. We extend the proof of Fact 3.7 in [5] to handle GPN and noninjective labelings of transitions.

II. CLASSES OF PETRI NET LANGUAGES

For the sake of brevity, we assume that the reader is familiar with the definition and operation (i.e., firing) of a PN (see, e.g., [7]). Our notation and terminology are as follows. A PN structure is a five-tuple \( C = (P, T, A, I, O) \) that describes the bipartite graph constituting the PN. \( P \) is the set of places, \( T \) the set of transitions, and \( A \) the set of directed arcs between \( T \) and \( P \). Let \( 2^A \) denote the power set of \( A \).

\( I: T \rightarrow P \times P \rightarrow 2^A \) and \( O: T \times P \rightarrow 2^A \) are functions that describe connections from places to transitions for \( I \), and from transitions to places for \( O \). It is convenient to index the elements of \( P \) and \( T \) by integers \( 1 \leq i \leq |P| \) and \( 1 \leq j \leq |T| \), respectively.

A PN structure \( C \) must satisfy the following assumptions: i) there should be no isolated node in the bipartite graph; ii) an arc connects only one \((t_i, p_j)\) pair, i.e., given any \( a \in A \), there exists a unique pair \((t_i, p_j)\) such that \( a \in I(t_i, p_j) \) or \( a \in O(t_i, p_j) \).

A “finite” PN satisfies \( |I(t_i, p_j)| \leq 1 \) and \( |O(t_i, p_j)| \leq 1 \) for all \((t_i, p_j) \in P \times T \), i.e., it has at most one arc from a given place to a given transition and one arc from a given transition to a given place (no “multiple arcs”). Let

\[
I(t_i) := \{ p_j \in P | I(t_i, p_j) \neq \emptyset \} \tag{2.1}
\]

\[
O(t_i) := \{ p_j \in P | O(t_i, p_j) \neq \emptyset \} \tag{2.2}
\]

A “self-loop free” PN satisfies \( I(t_i) \cup O(t_i) \leq \emptyset \forall t_i \in T \), i.e., there are no directed cycles of length two in the bipartite graph. RPN are self-loop free ordinary PN. The class of GPN allows both multiple arcs and self-loops and is the one usually meant when one simply says “Petri net.” It is convenient to denote by \( \Theta \) the class of GPN, and by \( \Theta^0 \) the class of self-loop free GPN, and by \( \Theta^* \) the class of RPN.

Using notation analogous to that employed for FSM, we define a PN to be a five-tuple \( N = (C, \Sigma, \alpha, \mu_0, F) \) where

- \( C \) is a PN structure \( (P, T, A, I, O) \);
- \( \Sigma \) is the alphabet of symbols for transition labeling; we require that \( \varepsilon \in \Sigma \) (this is the so-called \( \varepsilon \)-free labeling case);
- \( \alpha: T \rightarrow \Sigma \) is the transition labeling function, which need not be injective;
- \( \mu_0 \in \Theta^A \) is the initial marking vector of the places, i.e., the initial number of tokens in each place;
- \( F \subset \Theta^0 \) is the set of “final” markings that represent “completed” tasks \( F \) need not be finite.

Let \( b_\alpha: \Phi \rightarrow T \rightarrow \mathcal{P} \Sigma \) be the (partial) marking transition function corresponding to the admissible firings of \( N \). We extend this function to \( \delta b_\alpha: \Phi^* \times T^* \rightarrow \mathcal{P} \Sigma^* \) in the usual manner. Here, \( T^* \) is the Kleene closure of \( T \), also, \( \delta b_\alpha(\mu, \epsilon) = \mu \). Similarly, we extend the transition labeling function \( \alpha: T \rightarrow \Sigma^* \) thus \( \alpha(\epsilon) = \epsilon \iff s = \epsilon \).

We will consider two languages for a given PN \( N \) (corresponding to the \( P \)-type and \( L \)-type languages in [7]):

\[
L(N) = \{ \alpha(s) \in \Sigma^* | T \rightarrow s \in \delta b_\alpha(\mu_0, s) \} \text{ is defined} \tag{2.3}
\]

\[
L_\alpha(N) = \{ \alpha(s) \in \Sigma^* | T \rightarrow s \in \delta b_\alpha(\mu_0, s) \in F \} \tag{2.4}
\]

Next, define the classes of languages

\[
\mathcal{L}_S := \{ K \subset \Sigma^* | \forall N \in \Theta(K = L(N)) \} \tag{2.5}
\]

\[
\mathcal{L}_\alpha := \{ K \subset \Sigma^* | \forall N \in \Theta(K = L_\alpha(N)) \} \tag{2.6}
\]

and similarly the classes \( \mathcal{L}_{NS}, \mathcal{L}_{NS}^\alpha, \mathcal{L}_R, \mathcal{L}_R^\alpha \) by substituting \( \Theta^0 \) by \( \Theta^{NS} \) and \( \Theta^* \), respectively, in the above. Clearly, \( \mathcal{L}_{NS}^\alpha \subset \mathcal{L}_R^\alpha \subset \mathcal{L}_R \subset \mathcal{L}_\alpha \subset \mathcal{L}_S \). If \( \epsilon \in \Sigma \), i.e., if transitions can be labeled with the empty string, then Peterson’s construction [7, pp. 126–128] (for reachability analysis) actually proves the reverse inclusions. We collect this and other results from [7] in the following proposition. (Recall that a language is regular iff it is generated by an FSM.)

Proposition 2.1 [7]:

i) \( \mathcal{L}_S \subset \mathcal{L}_\alpha \)

ii) \( \mathcal{R} \subset \mathcal{L}_\alpha \) and \( \mathcal{R} \subset \mathcal{L}_S \), where \( \mathcal{R} \) denotes the class of regular languages and \( \mathcal{R} \) that of closed regular languages.

iii) If the problem formulation is changed to allow \( \epsilon \in \Sigma \), then \( \mathcal{L}_R^\alpha \subset \mathcal{L}_S \subset \mathcal{L}_\alpha \subset \mathcal{L}_R^\alpha \subset \mathcal{L}_R \).

Remark 2.1:

i) When \( \alpha \) is required to be injective (the so-called “free-labeling” case), the properties of \( \mathcal{L}_S \) and \( \mathcal{L}_\alpha \) are not as interesting as when this constraint is relaxed. Indeed, ii) in the above theorem is no longer true. This is why we do not require this assumption.

ii) It follows from definitions (2.3) and (2.5) that \( \forall L \subset \mathcal{L}_S, \epsilon \in L \). For the corresponding languages in \( \mathcal{L}_\alpha \), it simply means that \( \mu_0 \in F \). It is still the case that \( \epsilon \in \Sigma \).

The terminology is from [2].
III. A New Proof for Petri Net Languages

We now complement the results of [7] and [2] by presenting a new constructive proof of $L_n^\mathbb{N} = L_n$ and $E_n^\mathbb{N} = E$ for the case of $e$-free languages (i.e., $e\notin E$). To make the constructive procedure easier to follow, we first prove that $L_n^\mathbb{N} = L_n$ in Section III-A, and then we show how to extend that result to $L_n = L_n^\mathbb{N}$ in Section III-B (similarly for $E = E^\mathbb{N}$).

A. Self-Loop Free Case

Let $N = ((P, T, A, I, O), \Sigma, \sigma, \mu_0, F) \in \mathbb{N}^\mathbb{N}$. The objective is to build $N' = ((P', T', A', I', O'), \Sigma, \sigma', \mu_0', F') \in \mathbb{N}^\mathbb{N}$ such that $L_n(N') = L_n(N)$ and $E_n(N') = L(N)$. This will prove that $L_n^\mathbb{N} = L^\mathbb{N}$ and $E_n^\mathbb{N} = E^\mathbb{N}$. The construction procedure whose steps are described below, replaces each place $p_i \in P$ by a set of places $P'_i \subseteq P'$, and each transition $t_i \in T$ by a set of transitions $T'_i \subseteq T'$. (Recall that $\{I(t) \cap O(t) = \emptyset \; \forall t \in T\}$.

Step 1: For each $p_i \in P$, define
$$|P'_i| := \max \left\{ \max_{t \in T} |O(t, p_i)|, \max_{t \in T} |I(t, p_i)| \right\}. \tag{3.1}$$

Index each of the $|P'_i|$ places with $k, 1 \leq k \leq |P'_i|$, and form the disjoint union $P'_i := \bigcup_{p \in P'_i} p_i$. Thus, for each place $p_i \in N$, the number of corresponding places $p'_i \in N'$ is equal to the maximum number of outgoing/incoming arcs to/from $p_i$.

Before going to Step 2, we define the set $F(X, n)$, where $X$ is a set whose elements are positive integers, $1 \leq i \leq |X|$, and where $n \in \mathbb{N}$. $F(X, n) \subseteq X^n$ and it is constructed as follows. i) Build $n$-tuples from the indexed elements of $X$, with unity step, with $|X| = 1 \Rightarrow 1$ ("wrap-around"), and with $x_1$ the first element of the first $n$-tuple, $x_2$ the first element of the second $n$-tuple, etc. ii) The last $n$-tuple to build is the one whose first component is $x_1|X|$. Thus, the $n$-tuples look like $(x_1, \ldots, x_1), (x_2, \ldots, x_1), \ldots, (x_1, \ldots, x_{|X|-1})$. By convention, $F(X, 0) = \emptyset$. When two tuples of $F(X, n)$ contain the same elements, we say that they are equivalent. When $F(X, n)$ contains equivalent tuples, we delete all but one of these tuples. This arises if $|X| = n$. In all cases, each element $X$ appears in the same number of $n$-tuples in $F(X, n)$.

Example 3.1:
$$F(X = \{x_1, x_2, x_3\}, 3) = \{(x_1, x_2, x_3), (x_1, x_3, x_2), (x_2, x_1, x_3), (x_2, x_3, x_1), (x_3, x_1, x_2), (x_3, x_2, x_1)\}. \tag{3.2}$$

Let
$$|T'_i| := |P'_i| \cdot |I(t, p_i)| \cdot |O(t, p_i)|. \tag{3.3}$$

Then $T'_i \subseteq \{P'_i \times P'_i \times E_n\}$. Again, one should label each of these transitions distinctly and then form the disjoint union $\bigcup_{t \in T} T'_i = T'$. The labeling function $\sigma'$ is consistent with the original $\sigma$: if $t'_i \in T'_i$, then $\sigma'(t'_i) = \sigma(t)$. (If $t'_i \in T'_i$, we say that $t'_i$ and $t_i$ are corresponding transitions.)

Step 3: The remainder of the structure $C'$ of $N'$, namely, the connections between places and transitions, is done by examining the tuples in each $T'_i$ (3.2). Let $t_i \in T_i$. $I(t'_i)$ corresponds to the first $q$ components of $t'_i$, since, from (3.2), these components are places in subsets $P'_i$ where $p_i \in \{t_i\}$; for each $p_i$, rather than using multiple arcs, $|I(t'_i, p_i)|$ different places of $P'_i$ are connected to $t'_i$. Similarly, $O(t'_i)$ corresponds to the remaining $r$ components of $t'_i$ since these components are places in subsets $P'_i$ where this time $p_i \in O(t_i)$. It results from this construction that the original connections of $C$ are preserved in $C'$.

\begin{itemize}
  \item \[X \text{ denotes Cartesian product.}\]
\end{itemize}
ii) We now show that every string in \( L(N) \) can be generated by \( N' \) by proper choice of \( \mu' \), provided that given a firing of \( t_i \) in \( N \), one fires the "proper" \( t'_i \in T'_i \) in \( N' \).

Choose the \( \mu' \) that fills each \( P'_i \) uniformly with starting place \( p'_i \), for all \( p'_i \) such that \( \mu_0(p'_i) > 0 \). Thus, this \( \mu'_i \) is uniform and as a consequence of Lemma 3.2, \((N'_i, \mu'_i)\) is \((N, \mu_0)\)-consistent. Next, assume that \((N'_i, \mu'_i)\) is \((N, \mu)\)-consistent and that \( \mu' \) is uniform. We argue that if one fires \( t_i \) in \( N \) with \( \mu_{\text{new}} = \delta_{\nu}(\mu, t_i) \), then \( \exists t'_i \in T'_i \) such that \( \mu'_{\text{new}} = \delta_{\nu}'(\mu', t'_i) \) is uniform and \((N'_i, \mu'_{\text{new}})\) is \((N, \mu_{\text{new}})\)-consistent. This inductive argument will prove part ii) of the theorem. Its correctness is established as follows. The proper \( t'_i \) to fire in \( N' \) is the transition that, when fired, removes the tokens from the "oldest-filled" tuples for \( T'_i \) in \( F(P'_i, [I(t_i, p_i)]) \), \( p_i \in O(t_i) \). The existence of \( t'_i \) is guaranteed from the construction of \( T'_i \) in Step 2; namely, from (3.2), there exists a transition in \( T'_i \) for each possible combination of consecutive input and output places from the appropriate \( P'_i \) and \( P'_i \) subsets. By definition of "oldest-filled" and uniform marking, and since \( \mu' \in \mathcal{M}(\mu) \), \( t'_i \) is necessarily enabled in \( N' \) whenever \( t_i \) is enabled in \( N \). Firing \( t'_i \) results in marking \( \mu'_{\text{new}} \), which is uniform because if the rule oldest-filled/next-to-be-filled clearly preserves the uniformity of the markings when transitions are fired in \( N' \). Moreover, by Lemma 3.2, \((N'_i, \mu'_{\text{new}})\) will be \((N, \mu_{\text{new}})\)-consistent.

In conclusion, any firing sequence \( t_{i_1}, \ldots, t_{i_n} \) in \( N \) possesses a corresponding firing sequence \( t'_{i_1}, \ldots, t'_{i_n} \) in \( N' \), which completes the proof.

Example 3.3: Consider Fig. 1(b). If \( \mu'(P'_i) = [0 \ 1] \) then the oldest-filled tuple in \( F(P'_i, [I(t_i, p_i)]) \) for \( T'_i \) is \((1, 2, 3)\) [meaning \((p'_1, p'_2, p'_3)\)] (because \((1, 2, 3)\) is equivalent to \((2, 3, 1)\)), the oldest-filled tuple for \( T'_i \) in \( F(P'_i, [I(t_i, p_i)]) \) in \((2, 3)\), while the next-to-be-filled tuple by \( t'_i \) in \( F(P'_i, [O(t_i, p_i)]) \) is \((1, 2)\).

\[ P'_i(k, l) := \{ p'_j \in P'_i \mid p'_j \text{ is not a component of the tuple } f'_j(k) \} \]

B. General Case

Let \( S(t_i) = I(t_i) \cap O(t_i) \). Then define sets of arcs \( I'(\cdot, \cdot), O'(\cdot, \cdot), I''(\cdot, \cdot), O''(\cdot, \cdot), I'''(\cdot, \cdot), \) and \( O'''(\cdot, \cdot) \) (all initialized to \( \emptyset \)) as follows.

- If \( p_j \in S(t_i) \), then \( I'(t_i, p_j) = I(t_i, p_j) \) and \( O'(t_i, p_j) = O(t_i, p_j) \).
- If \( p_j \notin S(t_i) \), then \( I''(t_i, p_j) = I(t_i, p_j) \) and \( O''(t_i, p_j) = O(t_i, p_j) \).

The construction procedure of Section III-A is modified as follows.

**Step 1:**

\[ |P'_j| = \max \left| \max_{t_i \in T} |I'(t_i, p_j)| \right| \]
\[ \max_{t_i \in T} |O'(t_i, p_j)|, \max_{t_i \in T} |I'(t_i, p_j)| + |O'(t_i, p_j)| \] (3.5)

**Step 2:**

\[ T'_i = \{ X_{p_j \in O'(t_i)} F(P'_j, [O''(t_i, p_j)]) X \}
\[ \cap X_{p_j \in I'(t_i)} F(P'_j, [O''(t_i, p_j)]) X \]
\[ \cap X_{p_j \in I''(t_i)} F(P'_j, [O''(t_i, p_j)]) X \]
\[ X_{p_j \in O'(t_i)} \left[ \bigcup_{f'_j(k) \in F(P'_j, [O''(t_i, p_j)])} \right] \]
\[ \left. \left| I'(t_i, p_j) \right| \right] \] (3.6)

where

\[ P'_j(k, l) := \{ p'_j \in P'_j \mid p'_j \text{ is not a component of the tuple } f'_j(k) \} \]

(3.7)

Let

\[ \sum_{p_j \in O''(t_i)} |I''(t_i, p_j)| = q_i \quad \text{and} \quad \sum_{p_j \in O''(t_i)} |O''(t_i, p_j)| = r_i \]
Step 3: The connections between places and transitions in $C'$ are again inferred from the tuples in $T'$, except that this time these tuples have a more complicated form. The first $q_i$ components of $t'_i \in T'$ are input places to $t'_i$ since they correspond to $P_i(t')$ in (3.6). The next $r_i$ components are output places from $t'_i$ since they correspond to $O_i(t')$. Then output and input places alternate according to the third term in (3.6). For each $p_j \in S(t_i)$, we have $O_i(t', p_j)$ and $P_i(t', p_j)$ components that are input places to $t'_i$.

Step 4: Unchanged.

Remark 3.1: The intuition behind the third term in (3.6) is the following. Given $P'_i$ when $p_j \in S(t_i)$, each $f_{sc}(k)$ in (3.6) represents a different way of filling $(O_i(t', p_j))$ places in $P'_i$ which corresponds to the firing of $t_i$. For each $f_{sc}(k)$, we consider all the possible ways the remaining places (i.e., the set $P'_i \setminus (t', p_j)$) can be combined to provide the necessary $(P_i(t', p_j))$ tokens for the firing of $t_i$. (Note that from (3.5), there are enough remaining places.) Thus, self-loops are eliminated while the original connections in $C$ are preserved.

The above construction preserves the two important properties of the structure $C'$ of Section III-A: i) preservation of the connections of all $t'_i \in T'$; ii) presence of enough $t'_i$ to represent all the possible combinations of consecutive input and output places from the appropriate $P'_i$ subsets. Consequently, the proof of Theorem 3.1 applies mutatis mutandis to the present case.

In conclusion, we have proved the following result.

**Theorem 3.2:** $L_{\alpha}^{\mathcal{E}} = L_{\alpha}^{\mathcal{E}^*}$ and $L_{\mathcal{E}} = L$.

IV. DISCUSSION ON PETRI NETS AND FINITELY RECURSIVE PROCESSES

In [5], Inan and Vardiya introduce the FRP formalism for the modelling of DES. The language complexity of FRP is in terms of the sets of traces that these processes can generate. Let $A$ be the fixed set of events for the class of FRP, which we denote $\mathcal{F}$. We can write

$$\mathcal{L}^{\alpha} := \{K \in A^* | \exists X \in \mathcal{F}(K = \text{tr}X)\}$$

(4.1)

to represent the class of closed languages that $\mathcal{F}$ can generate (tr$X$ denotes the set of traces of process $X$). $\mathcal{L}^{\mathcal{E}}$ is compared to $\mathcal{L}$ in Fact 3.7 in [5]. However, two implicit assumptions on the structure of the PN generating $\mathcal{L}$ are made in the proof of that fact: i) there are no multiple arcs in the PN structure; and ii) the transition labeling function $f$ is injective (the free-labeling case).

In view of Theorem 3.2, Assumption i) is not restrictive. However, as was mentioned in Remark 2.1, Assumption ii) does make a difference. As one cannot relax that assumption in the proof of Fact 3.7 in [5] without affecting the synchronous composition defining the "master process" there, we suggest the following approach. We use the symbol change function $Hoare$ to define the new class of processes

$$\mathcal{L}^{\alpha}_{F} := \{Y | \exists X \in \mathcal{F} \land f_{sc} : A \to \Sigma Y = f_{sc}(X)\}$$

(4.2)

where $f_{sc}$ is as defined in [3, p. 55], i.e.,

$$\operatorname{tr}Y := \{x \in \Sigma^* | \exists \operatorname{tr}X (x = f_{sc}(t))\}$$

(4.3)

where $f_{sc}(a_1, \ldots, a_n) := f_{sc}(a_1) \cdot \cdots \cdot f_{sc}(a_n)$ for $a_i \in A, 1 \leq i \leq n$. Observe that the function $f_{sc}$ is not necessarily injective and that $\epsilon \not\in \Sigma$.

For our purposes, it is sufficient to consider FRP with constant event functions; also, the particular way in which the termination function of $Y = f_{sc}(X)$ is defined is irrelevant here.

Given a PN $N = (C, \Sigma, a, p_0, F)$, we proceed as follows to build an FRP $Z$ such that $\operatorname{tr}Z = L(N)$. We follow the proof of Fact 3.7 where $A = T$, the set of transitions of $N$, in order to build the "master process" $Z' = Y_1 | \cdots | Y_{|T|}$ in $\mathcal{F}$ (cf. [5]). Then we build $Z = f_{sc}(Z')$ where $f_{sc} = \sigma$, the transition labeling function of $N$. Thus, $\operatorname{tr}Z = L(N)$, which proves the following result.

**Proposition 4.1:** $L \subseteq L^{\alpha}$. 

V. CONCLUSION

$L_{\mathcal{E}}$ and $L$ are important classes of PNL. Because they allow non-injective transition labeling functions, they are more general and possess more interesting properties than "free-labeling" languages (cf. Remark 2.1). Moreover, they avoid $\epsilon$-labeling of transitions. Unlike the situation in finite automata theory, where both deterministic and nondeterministic automata generate the same class of languages (namely, $\epsilon$), allowing $\epsilon$-labeling may affect the properties of classes of PNL. In this context, Theorem 3.2 is an interesting result since it shows that the language complexity equivalence of GPN and RPN is also true for $\epsilon$-free languages. Theorem 3.2 also helped to establish Proposition 4.1. In general, the simplified structure of RPN as compared to GPN may prove helpful in the study of PN.

ACKNOWLEDGMENT

It is a pleasure to acknowledge many discussions with B. Krogh on the subject of Petri net languages.

REFERENCES


Simplifying $H_{\infty}$ Controller Synthesis Via Classical Feedback System Structure

HARRY S. HVOSTOV

Abstract—Recently, $H_{\infty}$-optimal controller synthesis problem has been shown to lead to the solution of two algebraic Riccati equations. This note explores the properties of this solution in the framework of the classical cascade controller feedback structure. In addition to allowing effective tradeoff between significant closed-loop transfer functions, this system structure is shown to substantially simplify controller computation in the practically important case of open-loop stable plant and weights in that at most one Riccati equation needs to be solved.

I. INTRODUCTION

The original solution to the $H_{\infty}$ norm minimization problem obtained by solving an equivalent model matching problem is summarized in [5] along with the relevant factorization theory. Complete state-space solutions for all stabilizing controllers which simultaneously minimize the $H_{\infty}$ norm of the appropriate closed-loop transfer function are given in [1]. There it is shown that under relatively mild assumptions on the plant the controller can be computed via a set of explicit state-space formulas which depend upon the solution of two associated algebraic Riccati equations.

In a typical $H_{\infty}$-optimal controller synthesis problem the plant would incorporate additional frequency dependent weights which are selected Manuscript received February 24, 1989; revised May 12, 1989. The author is with Storage Systems Advanced Development Group, Digital Equipment Corporation, Colorado Springs, CO 80919. IEEE Log Number 8933389.