UNIVERSITY OF MICHIGAN
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
LECTURE NOTES FOR EECS 661
CHAPTER 4: PETRI NETS

Stéphane Lafortune

November 2004

4.1: MODELING DES USING PETRI NETS

Motivation:

- Petri nets are a popular modeling formalism. Their graphical representation is intuitive and convenient for small systems.
- The modeling power of Petri nets is larger than that of finite-state automata, if the number of tokens is not bounded.
- *Sequential Function Charts* (SFC), a widely-used programming language for Programmable Logic Controllers (PLC), is inspired by Petri nets.
- Petri nets are amenable to analysis and to a certain extent to synthesis as well.
- There is a well-developed theory for a certain class of timed Petri nets (leading to the max-plus algebra - cf. Chapter 5).

Historical Remarks: Introduced by Carl Petri in his 1962 Ph.D. thesis (Germany). Mostly studied in Europe. Some work at MIT in the early 70's and more recently on their control (CMU, Kentucky, Notre-Dame) and use in manufacturing (RPI, NJIT, UM).

4.1: Modeling DES Using Petri Nets

Basic Definitions

- A *Petri net structure* (or *graph*), denoted by C, is a *weighted bipartite graph*

$$C = (P, T, A, w)$$

where

  - P is the finite set of *places* (one type of node in the graph)
  - T is the finite set of *transitions* (the other type of node in the graph)
  - $A \subseteq (P \times T) \cup (T \times P)$ is the set of arcs from places to transitions and from transitions to places in the graph
  - $w : A \rightarrow \{1, 2, 3, \ldots\}$ is the weight function on the arcs.

We assume that C has no isolated places or transitions.

- A *labeled Petri net* (or simply *Petri net*) is an eight-tuple

$$N = (P, T, A, w, E, \ell, x_0, X_m)$$

where

  - $C = (P, T, A, w)$ is a Petri net graph
  - E is the event set (or alphabet) for transition labeling;
  - $\ell : T \to E$ is the transition labeling function; in general, ℓ is not injective
  - $x_0 \in \mathbb{N}^{|P|}$ is the initial state or marking of the net, i.e., the initial number of tokens in each place
  - $X_m \subseteq \mathbb{N}^{|P|}$ is the set of "marked states" or "final markings" of the net.

The above definition does not explictly specify the state space and state transition function as these are implictly defined by C.

- The *state* of a net, denoted x, is a vector in $\mathbb{N}^{|P|}$ that corresponds to the number of tokens in each place; the number of tokens in place p is denoted by $x(p)$.

  The literature uses the term *marking* instead of state.

- The state space, X, is thus a subset of $\mathbb{N}^{|P|}$.

- The *state transition function* of the net,

$$f : \mathbb{N}^{|P|} \times T \rightarrow \mathbb{N}^{|P|} ,$$

  corresponds to the admissible *firings* of the net; "firing" is the term used in the literature for execution of an event.

- In order to describe f, it is convenient to define the four following functions.

  1. $I(t) := \{p : (p,t) \in A\}$
  2. $O(t) := \{p : (t,p) \in A\}$
  3. $Inp(p) := \{t : (t,p) \in A\}$
  4. $Outp(p) := \{t : (p,t) \in A\}$.

- The dynamics of the net are now described as follows:

$$f(x,t) \text{ is defined if } x(p) \geq w(p,t)\ \forall p \in I(t)$$

in which case $f(x,t) = x'$ where $x'(p) =$

$$\begin{aligned} x(p) \quad & \text{if} \quad p \notin I(t) \wedge p \notin O(t) \\ x(p) - w(p,t) \quad & \text{if} \quad p \in I(t) \wedge p \notin O(t) \\ x(p) - w(p,t) + w(t,p) \quad & \text{if} \quad p \in I(t) \wedge p \in O(t) \\ x(p) + w(t,p) \quad & \text{if} \quad p \notin I(t) \wedge p \in O(t) \end{aligned}$$

The above can be written simply as

$$x'(p) = x(p) - w(p,t) + w(t,p)$$

if we assign $w(p,t)$ $[w(t,p)]$ to zero when $(p,t) \notin A$ $[(t,p) \notin A]$.

- When $f(x,t)$ is defined, we say that t is *enabled* at state x.

- The state transition function is extended to $f : \mathbb{N}^{|P|} \times T^* \to \mathbb{N}^{|P|}$ in the usual manner.
- Observe that the number of tokens in the net need not be constant; in fact, it can grow unboundedly.
- The set of *reachable* states of a net is:

$$R(N) := \{y \in \mathbb{N}^{|P|} : \exists s \in T^*(f(x_0, s) = y)\} \ .$$

- With the above definitions, we can characterize the *languages* generated and marked by a net N.
  - We start by extending the labeling function to $\ell : T^* \to E^*$ in the usual manner.
  - Then we define:

$$\begin{aligned} \mathcal{L}(N) &= \{\ell(s) \in E^* : (s \in T^*) \wedge (f(x_0, s) \text{ is defined })\} \\ \mathcal{L}_m(N) &= \{\ell(s) \in \mathcal{L}(N) : (s \in T^*) \wedge (f(x_0, s) \in X_m)\} \ . \end{aligned}$$

- The class of languages that can be represented by Petri nets is

$$\mathcal{PNL} := \{K \subseteq E^* : \exists N(K = \mathcal{L}_m(N))\} \ .$$

  - This is a general definition and the properties of $\mathcal{PNL}$ depend heavily on the specific properties of ℓ and X_m (more later).
  - In any case, $\mathcal{R} \subset \mathcal{PNL}$: inclusion is easily seen as any finite-state automaton can be "redrawn" as a net; strict inclusion is illustrated by an example.

Some Remarks on Modeling

- Various situations that can be modeled by Petri nets are: conflict (as for automata), concurrency (not "directly" modeled by automata - cf. *interleaving*), and confusion.

  (Note that in our discussion of $\mathcal{PNL}$ above, we assumed no concurrent events. Doing so would require defining so-called "concurrent Petri net languages".)

- The fact that places can contain an arbitrarily large number of tokens does not prevent modeling situations where places represent entities that have *bounded* capacities (e.g., buffers).

  – In these cases, the transition rule needs to be changed to account for these bounds.

  – However, we can avoid changing the transition rule if we use in the construction of the net so-called *complementary places*. Note that this transformation does not affect the language the net in the sense that it does not introduce new transitions (events). (See Problem 4.14...)

  – In conclusion, we can restrict our discussion to infinite-capacity places WLOG (without loss of generality).

Classification of Nets: Some Terminology

- *Ordinary nets*: when $w : A \rightarrow \{1\}$, i.e., all arcs have weight one.
- *State graphs:* nets that
  1. are ordinary;
  2. $|I(t)| = |O(t)| = 1 \ \forall t \in T$; and
  3. $||x_0||_1 = 1$ (only one token in initial state).

  Thus the number of tokens remains constant and equal to one.

  Sometimes these are called "finite-state-machine nets"; this is bad terminology in the sense that these are not the only kinds of nets that can be represented as FSM (or automaton); any net with a *finite state space* can be transformed to an equivalent FSM (automaton), i.e., a FSM that generates and marks the same languages. Take $X = R(N)$, etc. (Of course, the representation is unlikely to be as compact.)

- *Marked graphs:* nets that

  1. are ordinary, and
  2. $|Inp(p)| = |Outp(p)| = 1 \; \forall p \in P$.

  Thus there is never any conflict.

  This is dual to the preceding subclass.

  Marked graphs are also called *event graphs* and *decision-free nets*.

  This class of nets has nice properties and has been extensively studied. In particular, for timed nets, it leads to dynamics that can be represented linearly in the (max,+) algebra.

- Nets with *inhibitor* arcs: the transition can only fire if the input place coresponding to the inhibitor arc is *empty*.

- *Colored nets:* when there are multiple classes of tokens in places and corresponding multiple firing rules for a transition.

4.2: ANALYSIS OF PETRI NETS

4.2: Analysis of Petri Nets

Behavioral Properties of Interest

Behavioral properties are those properties that depend on all of N, i.e, not only on its structure but also on its initial state.

- **Reachability:** Concerns $R(N)$, $\mathcal{L}(N)$ and $\mathcal{L}_m(N)$.

  The related property of *coverability* is also very useful. The coverability problem is: given N and x, does there exist $x' \in R(N)$ such that $x' \geq x$ (where $\geq$ is component-wise)?

- **Boundedness:** A net is said to be *k-bounded* if $x(p) \leq k$ for all $p \in P$ and all $x \in R(N)$.

  If this is true for $k = 1$, the net is said to be *safe*. For example, a state graph is necessarily safe.

  If a net is k-bounded, then its state space is contained in $\{0, 1, \ldots, k\}^{|P|}$ and thus the net is language-equivalent to a finite-state automaton.

- **Liveness:**
  - A transition t is said to be *potentially firable* in state x if there exists x' reachable from x (i.e., there exists $s \in T^*$ such that $f(x, s) = x'$) such that t is enabled in x'; if not, then t is said to be *dead* in x.
  - A *transition t is live* in state x if it is potentially firable in every state reachable from x.
  - A *net is live* if every transition is live in x_0. Thus no matter what state x has been reached from x_0, it is always possible to ultimately fire any transition in the future. This is a very strong condition. In particular, it means that the net does not deadlock. Many weaker notions of liveness have been studied in the literature.

- **Non-interruptedness (or persistence):** For any two enabled transitions, the firing of one does not disable the other. This is also called *persistence*.
- **Conservation:** Petri net N with initial state x_0 is said to be *conservative with respect to* $\boldsymbol{\gamma} = [\gamma_1, \gamma_2, \ldots, \gamma_n]$ if

$$\sum_{i=1}^{n} \gamma_i x(p_i) = \text{ constant} \tag{1}$$

  for all states $x \in R(N)$.

  The motivation for this definition is that often, tokens model resources that are being allocated, in which case their number should remain constant.

  If a net is conservative with all $\gamma_i > 0$, then that net is equivalent to a finite-state automaton since its state space is necessarily finite.

The Coverability Tree Approach

- The *coverability tree* of a net N is a finite tree that is built to represent $R(N)$.

  This tree is denoted $CT(N)$ and it can used to study some of the behavioral properties of N.

- Since $CT(N)$ is finite by definition, the use of a special symbol is necessary when N is not bounded, i.e., when $R(N)$ has infinite cardinality.

  – We will write ω when the number of tokens in a place can become arbitrarily large.

  – The symbol ω is to be treated as ∞, i.e., for any nonnegative integer k, we have:

$$\omega \pm k = \omega, \quad k < \omega, \quad \omega \leq \omega \ .$$

Algorithm to Construct CT:

Step 1: Initialize $\mathbf{x} = \mathbf{x}_0$ (initial state).

Step 2: For each new node, $\mathbf{x}$, evaluate the transition function $f(\mathbf{x}, t_j)$ for all $t_j \in T$:

Step 2.1: If $f(\mathbf{x}, t_j)$ is undefined for all $t_j \in T$ (i.e., no transition is enabled at state x), then $\mathbf{x}$ is a terminal node.

Step 2.2: If $f(\mathbf{x}, t_j)$ is defined for some $t_j \in T$, create a new node $\mathbf{x}' = f(\mathbf{x}, t_j)$. If necessary, adjust the marking of that node as follows:

Step 2.2.1: If $x(p_i) = \omega$ for some p_i, set $x'(p_i) = \omega$.

Step 2.2.2: If there exists a node $\mathbf{y}$ in the path from the root node $\mathbf{x}_0$ (included) to $\mathbf{x}$ such that $\mathbf{x}' >_d \mathbf{y}$, set $x'(p_i) = \omega$ for all p_i such that $x'(p_i) > y(p_i)$.

Step 2.2.3: Otherwise, $x'(p_i)$ is as obtained in $f(\mathbf{x}, t_j)$.

Step 3: If all new nodes are either terminal or duplicate nodes, stop.

It is not difficult to prove that by construction, CT is guaranteed to be finite (this is due to the use of ω).

Using CT:

- Clearly, N is bounded iff ω does not appear in the nodes of $CT(N)$.

  In this case the *coverability* tree becomes the *reachability* tree.

- N is safe iff only 0's and 1's appear in the nodes of $CT(N)$.

- $CT(N)$ allows to check if N is conservative, since there is only finite number of nodes (states) to test.

  – This results in having to solve a system of linear equations.

  – If ω is present, its weight must be 0.

- Due to the "loss of information" that the use of ω entails, reachability, liveness, and language issues cannot in general be completely answered from $CT(N)$.

  For instance, if $x \in R(N)$, then there exists node $y \in CT(N)$ such that $x \leq y$, but *the converse need not be true.*

The Linear-Algebraic Approach

Linear Discrete-Time State Equation

- Some analytical results can be derived by taking advantage of the numerical structure of the state space of a net.
- For this purpose, define the $|T| \times |P|$ matrix $\mathbf{A}$ where each entry is given by

$$a_{j,i} = w(t_j, p_i) - w(p_i, t_j)$$

i.e., $a_{j,i}$ is the net change in the number of tokens in place p_i when transition t_j fires.

Clearly,

$$t_j \text{ is enabled iff } x(p_i) \geq w(p_i, t_j) \text{ for all } p_i \in I(t_j).$$

- If we denote by x_k the state of the net immediately after the k-th firing of an enabled transition, with x_0 as the initial state, and by u_k a $|T| \times 1$ column vector with zero entries except for component l if t_l is the k-th transition to fire, then the following linear equation describes the evolution of the state:

$$x_k = x_{k-1} + \mathbf{A}^T u_k, \ k = 1, 2, \ldots$$

Clearly, we should always have that $x_k \geq 0$, since only enabled transitions can fire.

- If we consider a firing sequence of k transitions and form the sum

$$u_{1,k} = \sum_{l=1}^{k} u_l$$

then we can write

$$x_k = x_0 + \mathbf{A}^T u_{1,k} \ .$$

$u_{1,k}$ is called the *firing count vector*.

- Is is important to observe that the sequencing information on the order of firing of the transitions from x_0 to x_k is lost in $u_{1,k}$.

Thus, in general, only necessary conditions can be obtained from the state equation.

There are basically two reasons why this happens:

1. The state x_k is required to be a nonnegative integer.
2. The equation $\mathbf{A}^T u = (x - x_0)$ might have an integer solution u for given x and x_0, yet none of the transitions fired by u could be enabled in x_0.

Thus if u is found, one may have to exhaustively try all possible orderings of transition firings in order to make sure a given x is indeed reachable.

P- and T-invariants

- A $|P| \times 1$ non-zero integer solution y of $\mathbf{A}y = 0$ is called a *P-invariant* of the net structure.
- A $|T| \times 1$ non-zero integer solution u of $\mathbf{A}^T u = 0$ is called a *T-invariant* of the net structure.
- Invariants are used for the study of *structural* properties of Petri nets, i.e., properties that depend only on the Petri net structure and therefore must hold for any initial state.

  We state some results.

- **Definition:** A net *structure* is said to be *conservative* if there exists $y \in \mathbb{N}^{|P|}$, $y > 0$, such that for all choices of x_0 in N and for all $x \in R(N)$,

$$y^T x = y^T x_0 \ .$$

  If the above does not hold for $y > 0$ but does hold for some $y \geq 0$, then C is said to be *partially conservative*.

  Result: A net structure is (partially) conservative iff there exists a P-invariant y of positive (nonnegative) integers.

  A net structure is strictly conservative iff $y = [1 \cdots 1]^T$ is a P-invariant.

- **Definition:** A net *structure* is said to be *bounded* if, for any choice of x_0, the resulting net is bounded.

  Result: A net structure is bounded iff there exists a $|P| \times 1$ vector y of positive integers such that $\mathbf{A}y \leq 0$.

- **Definition:** A net structure is *consistent* if there exists x_0 and a firing sequence $s \in T^*$ such that:

  1. $f(x_0, s) = x_0$; and
  2. every transition in T occurs at least once in s.

  Result: A net structure is consistent iff there exists a T-invariant u of positive integers.

Analysis of Marked Graphs

Because of their special structure, several analytical resutls are available about marked graphs. We briefly discuss some frequently cited results.

- By the defining property of a marked graph, it can be redrawn as a directed graph where nodes correspond to transitions, arcs to places, and tokens are placed on arcs.

  In this new digraph, a node fires by removing a token from each incoming arc and placing a token on each outgoing arc (recall that a marked graph is an ordinary net, i.e., all its arc weights are equal to one).

- A *cycle* (or circuit) in a marked graph is a closed path from a transition back to the same transition.

  Thus a cycle in a marked graph is a directed cycle in the digraph representation of the marked graph. But if a node (transition) of the digraph is on a directed cycle, then it has exactly one incoming arc and one outgoing arc that also belong to the cycle.

  These observations lead to the following results.

1. In a marked graph N, the number of tokens in a cycle is invariant under any firing, i.e.,

$$x(Cycle) = x_0(Cycle)\ \forall x \in R(N)$$

   where $x(Cycle) := \sum_{p \in Cycle} x(p)$.

2. An *autonomous* marked graph is live iff x_0 places at least one token in each cycle. (Autonomous means no source or sink transitions.)

3. A live marked graph is safe iff every place belongs to a cycle with $x_0(Cycle) = 1$.

Comments about Decidability

While all the problems of interest for finite-state automaton models are decidable (e.g., reachability, language equivalence, etc.), the same is not true for Petri net models of discrete event systems.

[A problem is *undecidable* if there does not exist a general algorithm for its solution, i.e., there does not exist a computer program to solve this problem for any input.]

- The reachability problem (Is $x \in R(N)$?) and the liveness problem (Is N live?) are equivalent (i.e, each one can be reduced to the other) and decidable (albeit with exponential complexity).
- The equality problem (Is $R(N_1) = R(N_2)$?, or equivalently, Is $\mathcal{L}_m(N_1) = \mathcal{L}_m(N_2)$?) is undecidable!

4.3: PETRI NET LANGUAGES

4.3: Petri Net Languages

- The main class of Petri net languages of interest is the class $\mathcal{PNL}$ defined in section 4.1.
- Recall that $\mathcal{PNL}$ is strictly larger than the class of regular languages.

  $\mathcal{PNL}$ is closed under union, intersection, shuffle, and concatenation. However, it is not closed under Kleene-closure (*).

  The proof of this fact involves manipulations of Petri net structures.

  We discuss two useful manipulations.

Nets with Starting Places

- This manipulation consists of transforming any net N to a language-equivalent net N', where the structure of N' has a specific place identified as *starting place* and denoted p_S, such that:
  1. p_S has no input transitions, i.e., $Inp(p_S) = \emptyset$;
  2. the initial state x'_0 has one token in p_S and no tokens elsewhere.
- The construction procedure of N' from N is as follows:
  1. Start with $C' = C$, then add new place p_S to P'.
  2. $T' \leftarrow T' \cup \{t''_j : tj \text{ is enabled at } x_0 \text{ in } N\}$.
  3. For each t''_j:
     $\ell(t''_j) = \ell(t_j)$
     $I(t''_j) = \{p_S\}$ and $w'(p_S, t''_j) = 1$.
     Now define $w'(t''_j, p) = x_0(p) - w(p, t_j) + w(t_j, p)$ and set
     $O(t''_j) = \{p : w'(t''_j, p) > 0\}$.
     The purpose here is that the firing of t''_j should result in the same state as if t_j had been fired in N.

- This transformation clearly preserves language-equivalence.

  Moreover, it makes it easy to "merge" two nets:

  - in the case of union, overlap the two starting places;
  - in the case of shuffle, places the two nets "side-by-side".

Product of Two Nets

- The product of two Petri nets N_1 and N_2 is defined to be the net $N = N_1 \times N_2$ that generates $\mathcal{L}(N_1) \cap \mathcal{L}(N_2)$ and marks $\mathcal{L}_m(N_1) \cap \mathcal{L}_m(N_2)$.
- To contruct $N_1 \times N_2$, proceed as follows:
  1. WLOG, start with N_1 and N_2 with unique starting places p_{S1} and p_{S2}.
     Set $P = P_1 \cup P_2$.
  2. For each pair of transitions $t_j \in T_1$ and $t_k \in T_2$ for which $\ell(t_j) = \ell(t_k)$, create a new transition t_{jk} which fires iff t_j and t_k fire:
     $I(t_{jk}) = I(t_j) \cup I(t_k)$ and $w(p, t_{jk}) = w(p, t_j) + w(p, t_k)$;
     $O(t_{jk}) = O(t_j) \cup O(t_k)$ and $w(t_{jk}, p) = w(t_j, p) + w(t_k, p)$.
     T of N is the set of all such t_{jk}.
  3. Remove all isolated places from P.
  4. Set $x_0(p_{S1}) = x_0(p_{S2}) = 1$ and $x(p) = 0$ for all other places in P.
  5. Set X_m to be the set of states of N for which the components from P_1 correspond to a marked state in N_1 and the components from P_2 correspond to a marked state in N_2.
- $\mathcal{PNL}$ is closed under parallel composition; modify the above procedure by including (in addition to step 2) the transitions (and the places attached to them) of N_1 and N_2 whose event labels are in $E_1 \backslash E_2$ (for N_1) and $E_2 \backslash E_1$ (for N_2)

The Problem with Trimming in $\mathcal{PNL}$

- Trimming a finite-state automaton to render it *nonblocking* is essentially "trivial": do a reachability search to identify the states that are not coaccessible and delete them.
- Trimming a blocking Petri net is not "trivial."
  - First, the trimming of a net may require the introduction of new places and transitions to its structure.
  - Second, the trimming of a net may lead to a more fundamental problem: the prefix-closure of a Petri net language *need not be a Petri net language*!

    To see this, consider the Petri net language

    $$\mathcal{L}_m(N) = \{a^m b a^m b : m \geq 0\}$$

    marked by (blocking) net N. It can be verified that $\overline{\mathcal{L}_m(N)} \notin \mathcal{PNL}$.

4.4: CONTROL OF PETRI NETS

4.4: Control of Petri Nets

Supervisory Control and Petri Nets

- The theory of supervisory control that we presented in Chapter 3 is event- and language-based, thus it is (to some extent) independent of the particular formalism used to represent the languages of interest.

  Automata models and the corresponding class of regular languages were discussed extensively because of the nice closure properties of the class of regular languages with respect to the operations that arise in solving supervisory control problems and also because of the amenability of automaton models for performing the required synthesis operations.

- Clearly, one could also consider supervisory control problems where Petri nets are used to model the uncontrolled system behavior, i.e., G is a Petri net, and/or the controller S is realized by a Petri net N (as opposed to being realized by an automaton R). The feedback control loop would proceed exactly as before.

- An important motivation for using Petri net models is to *extend the applicability of the synthesis results of supervisory control theory beyond the class of regular languages.*

  It turns out that several difficulties arise in this endeavor. For this reason, no comprehensive set of results has been developed yet. Some of these difficulties are highlighted in the textbook.

- If $\mathcal{L}_m(G)$ and L_{am} are both regular languages, then it is not necessary to use Petri net models, even if these languages are specified in terms of Petri net recognizers.

  These Petri nets will necessarily be bounded and thus we can always build equivalent automata and then apply the algorithms of Chapter 3 to solve the supervisory control problems.

  However, it may still be useful to work with the Petri net representations for computational complexity reasons.

  This is the motivation for the large amount of work on *state feedback* control of Petri nets.

Remark on Infinite-State Systems or Specifications

- It should be noted that if $\mathcal{L}_m(G)$ is not regular (say, G is an unbounded Petri net) but L_{am} is *regular and controllable* (w.r.t. $\mathcal{L}(G)$ and E_{uc}), then the non-regularity of $\mathcal{L}_m(G)$ is of no consequence since the desired controller S can be realized by an automaton (as was done in Chapter 3).
  - The "catch" here is the verification of the controllability properties of L_{am}, which in general requires manipulating G.
  - However, it should be observed that if it can be verified that L_{am} is controllable w.r.t a *superset* of $\mathcal{L}(G)$, then it is necessarily controllable w.r.t. $\mathcal{L}(G)$. Thus if one "suspects" that L_{am} is controllable, then one can attempt to formally verify that conjecture by proving the controllability of L_{am} w.r.t. a *regular* superlanguage of $\mathcal{L}(G)$.
- On the other hand, if $\mathcal{L}_m(G)$ is regular but L_{am} is *not regular but controllable*, then S cannot be realized by an automaton. (The regularity of $\mathcal{L}_m(G)$ is of no help.) But one may be able to realize S by a Petri net!

State Feedback Control of Petri Nets

- In this approach, the control of Petri nets is considered from a "state viewpoint" as opposed to a "language viewpoint".
  - The uncontrolled system model G is a Petri net whose transition labeling function is injective. (Equivalently, $E = T$.)
  - The set of controllable events (i.e., transitions) of G is dynamically controlled by control policy
    $$\varphi : R(G) \rightarrow 2^E$$
    i.e., the control action $\varphi(x)$ is a function of the current state x of the net and not of the whole past behavior.
  - Let us denote the controlled system by φ/G.
- The goal is to synthesize a control policy φ such that the set of reachable states under the control of φ, $R(\varphi/G)$, is disjoint from a given set of *forbidden states* (or forbidden markings) and is *maximally permissive* (in some sense).

  This is called the *forbidden state problem of controlled Petri nets*, denoted FSPN hereafter.

- Clearly, FSPN is equivalent to BSCP if the legal language L_a is defined to be $\mathcal{L}(G)$ minus all traces that go through illegal states and if we assume that no two enabled transitions can fire simultaneoulsly.
- The point here though is to exploit the *structure* of the net G in order to synthesize the control policy φ.

  In particular, in the case of bounded nets, the structure should be exploited in order to solve FSPN in a more computationally efficient manner than if the net had been transformed to an automaton and FSPN had been solved using the synthesis algorithms of Chapter 3.
- For further details, see the survey paper by Holloway, Krogh, & Giua in *Discrete Event Systems: Theory and Applications*, 1997.

Control of Petri Nets Using P-Invariants

- Another approach for posing the control problem for Petri nets is to look only at the Petri net graph and consider specifications given in terms of linear inequalities on the state (marking) of the net that must hold for all initial states:

$$l^T x = x^T l \leq b_1$$

  for all $x \in R(N)$, and for all x_0.

- The key here is to connect such specifications with the property of conservation and P-invariants:

$$\begin{aligned} x^T y &= x_0^T y + u^T \mathbf{A} y \\ \mathbf{A} y &= 0 \\ \Rightarrow x^T y &= x_0^T y = constant(x_0) \end{aligned}$$

- Rewrite the above specification as an equality by introducing a "slack place", denoted by p_c:

$$x^T l + x(p_c) = b_1$$

The introduction of this new place means that the new incidence matrix for the Petri net graph is of the form:

$$\mathbf{A_{new}} = [\mathbf{A} \;\; A_c]$$

where A_c is a row describing the connectivity of new place p_c.

- We can see now that if we can find a P-invariant for $\mathbf{A_{new}}$, we will get:

$$\begin{aligned} \mathbf{A_{new}} y_{des} &= 0 \;\text{ where }\; y_{des}^T = [l^T \;\; 1] \\ \Rightarrow x_{new}^T y_{des} &= constant \\ \Rightarrow x^T l + x(p_c) &= constant \end{aligned}$$

- The initial marking in place p_c is chosen to force the "constant" to be equal to b_1:

$$x_0(p_c) = b_1 - x_0^T l$$

If this results in negative entries for $x_0(p_c)$, then no solution exists to the original problem.

- So the essence of the problem is to solve the matrix equality:

$$A_c = -\mathbf{A} l$$

- For further details, see the book by Moody & Antsaklis on "Supervisory Control of Petri Nets".