Introduction to Petri Net Theory [⋆]

Hsu-Chun Yen

Dept. of Electrical Engineering, National Taiwan University
Taipei, Taiwan, R.O.C.
(yen@cc.ee.ntu.edu.tw)

Abstract. This paper gives an overview of Petri net theory from an algorithmic viewpoint. We survey a number of analytical techniques as well as decidability/complexity results for various Petri net problems.

1 Introduction

Petri nets, introduced by C. A Petri in 1962 [54], provide an elegant and useful mathematical formalism for modelling concurrent systems and their behaviors. In many applications, however, modelling by itself is of limited practical use if one cannot analyze the modelled system. As a means of gaining a better understanding of the Petri net model, the decidability and computational complexity of typical automata theoretic problems concerning Petri nets have been extensively investigated in the literature in the past four decades.

In this paper, we first give an overview of a number of analytical techniques known to be useful for reasoning about either structural or behavioral properties of Petri nets. However, due to the intricate nature of Petri nets, none of the available analytical techniques is a panacea. To understand the limitations and capabilities of analyzing Petri nets from an algorithmic viewpoint, we also summarize a variety of decidability/complexity results reported in the literature for various Petri net problems including *boundedness, reachability, containment*, *equivalence*, and more. Among them, Lipton [40] and Rackoff [56] have shown exponential space lower and upper bounds, respectively, for the boundedness problem. As for the containment and the equivalence problems, Rabin [2] and Hack [19], respectively, have shown these two problems to be undecidable. In spite of the efforts made by many researchers over the years, many analytical questions concerning Petri nets remain unanswered.

The quest for solving the general reachability problem for Petri nets has constituted perhaps the most important and challenging line of research in the Petri net community in the past. Knowing that the problem requires exponential space [40], the decidability issue of the problem was left unsolved for a long period of time until Mayr [42, 43] finally provided an answer in the affirmative (see also [38]). Before Mayr's proof, a number of attempts were made to investigate the problem for restricted classes of PNs, in hope of gaining more insights and developing new tools in order to conquer the general PN reachability problem

[⋆] This work was supported in part by NSC Grant 94-2213-E-002-087, Taiwan.

(see, e.g., [16, 21, 39, 41, 48, 61]). A common feature of those attempts is that decidability of reachabiltiy for those restricted classes of Petri nets was built upon their reachability sets being *semilinear*. As semilinear sets precisely correspond to the those characterized by Presburger Arithmetic (a decidable theory), decidability of the reachability problem follows immediately. In view of the importance of the role played by the concept of semilinearity in Petri net theory, we devote a section in this paper to surveying analytical techniques and complexity results for subclasses of Petri nets exhibiting semilinear reachability sets. As for the general reachability problem, the only known algorithm is nonprimitive recursive (see [38, 42, 43]). The exact complexity of the reachability problem remains the most challenging open problem in Petri net theory.

It is well-known that the computational power of Petri nets is strictly weaker than that of Turing machines, making them inadequate for modelling certain real-world systems such as prioritized systems [1]. To overcome this shortcoming, a number of extended Petri nets have been introduced to enhance the expressive capabilities of Petri nets. Among them are *colored Petri nets*, *Petri nets with inhibitor arcs*, *timed Petri nets*, *prioritized Petri nets*, and more. With the above extended Petri nets powerful enough to simulate Turing machines, all nontrivial problems for such Petri nets become undecidable. A natural and interesting question to ask is: are there Petri nets whose powers lie between conventional Petri nets and Turing machines? It turns out that the so-called *reset nets* and *transfer nets* are two such witnesses. The quest for such 'weaker' extensions has attracted considerable attentions in recent years.

This paper gives an overview of basic analytical techniques and decidability/complexity results for various Petri net problems. Our survey is by no means comprehensive; the interested reader is refer to [12, 33, 49] for other survey articles concerning the decidability and complexity issues of Petri nets. See also [7, 53, 58] for more about Petri nets and their related problems.

The rest of this paper is organized as follows. Section 2 gives the basic notations and terminologies of Petri nets and their equivalent models. Section 3 is devoted to the definitions of various Petri net problems that are of interest in the Petri net community. An overview of analytical techniques known to be useful for reasoning about Petri net behaviors is presented in Section 4. Decidability and complexity results concerning various Petri net problems for general Petri nets and for subclasses of Petri nets are given in Section 5 and Section 6, respectively. Finally, in Section 7 we briefly discuss the computational power of various extended Petri nets.

2 Preliminaries

Let Z (N) denote the set of (nonnegative) integers, and Z^k (N^k) the set of vectors of k (nonnegative) integers. For a k-dimensional vector v, let $v(i), 1 \leq i \leq k$, denote the ith component of v. For a $k \times m$ matrix A, let $A(i, j)$, $1 \leq i \leq k, 1 \leq j \leq m$, denote the element in the ith row and the jth column of A. We let $|S|$ be the number of elements in set S. Given a vector x, we let x^T

denote the *transpose* of x. Given an alphabet (i.e., a finite set of symbols) Σ, we write Σ^* to denote the set of all finite-length strings (including the empty string λ) using symbols from Σ.

2.1 Petri nets

A *Petri net* (PN, for short) is a 3-tuple (P, T, φ), where P is a finite set of *places*, T is a finite set of *transitions*, and φ is a *flow function* $\varphi : (P \times T) \cup (T \times P) \rightarrow N$. A *marking* is a mapping $\mu : P \rightarrow N$. (μ assigns tokens to each place of the net.) Pictorially, a PN is a directed, bipartite graph consisting of two kinds of nodes: *places* (represented by circles within which each small black dot denotes a *token*) and *transitions* (represented by bars or boxes), where each arc is either from a place to a transition or vice versa. In addition, each arc is annotated by either $\varphi(p, t)$ or $\varphi(t, p)$, where p and t are the two endpoints of the arc. See Figure 1 for an example, in which all the arc labels are one and are therefore omitted.

A transition $t \in T$ is *enabled* at a marking μ iff $\forall p \in P$, $\varphi(p, t) \leq \mu(p)$. If a transition t is enabled, it may *fire* by removing $\varphi(p, t)$ tokens from each input place p and putting $\varphi(t, p')$ tokens in each output place p'. We then write $\mu \overset{t}{\longmapsto} \mu'$, where $\mu'(p) = \mu(p) - \varphi(p, t) + \varphi(t, p)$, $\forall p \in P$.

Example 1. Figure 1 depicts a PN (P, T, φ) with $P = \{p_1, p_2, p_3, p_4, p_5\}$ and $T = \{t_1, t_2, t_3, t_4\}$, modelling a simple *producer-consumer* system. We can view a marking μ as a 5-dimensional column vector in which the ith component is $\mu(p_i)$. In Figure 1, transition t_2 is enabled at marking $\mu = (1, 0, 0, 1, 0)$ since the only input place of t_2 (i.e., p_1) satisfies $\varphi(p_1, t_2) \leq \mu(p_1)$. After firing the transition t_2, the PN reaches a new marking $\mu' = (0, 1, 1, 1, 0)$, i.e., $(1, 0, 0, 1, 0) \overset{t_1}{\longmapsto} (0, 1, 1, 1, 0)$. $\square$

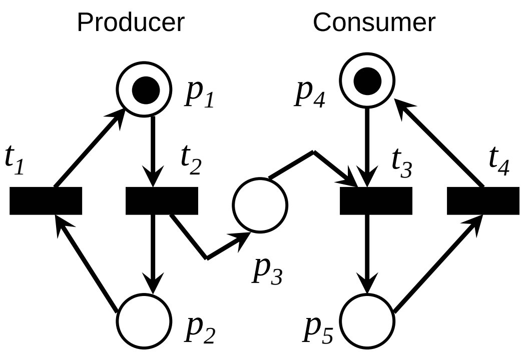


Fig. 1. A Petri net.

A sequence of transitions $\sigma = t_1...t_n$ is a *firing sequence* from μ_0 iff $\mu_0 \overset{t_1}{\longmapsto} \mu_1 \overset{t_2}{\longmapsto} \cdots \overset{t_n}{\longmapsto} \mu_n$ for some markings μ_1,...,μ_n. (We also write '$\mu_0 \overset{\sigma}{\longmapsto} \mu_n$'.) We write '$\mu_0 \overset{\sigma}{\longmapsto}$' to denote that σ is enabled and can be fired from μ_0, i.e., $\mu_0 \overset{\sigma}{\longmapsto}$ iff there exists a marking μ such that $\mu_0 \overset{\sigma}{\longmapsto} \mu$. The notation $\mu_0 \overset{*}{\longmapsto} \mu$ is used to denote the existence of a σ such that $\mu_0 \overset{\sigma}{\longmapsto} \mu$. A *marked* PN is a pair $((P, T, \varphi), \mu_0)$, where (P, T, φ) is a PN, and μ_0 is called the *initial marking*. Throughout the rest of this paper, the word 'marked' will be omitted if it is clear from the context.

By establishing an ordering on the elements of P and T (i.e., $P = \{p_1, ..., p_k\}$ and $T = \{t_1, ..., t_m\}$), we define the $k \times m$ *incidence matrix* $[T]$ of (P, T, φ) so that $[T](i, j) = \varphi(t_j, p_i) - \varphi(p_i, t_j)$. Note that $\varphi(t_j, p_i)$, $\varphi(p_i, t_j)$, and $[T](i, j)$, respectively, represent the number of tokens removed, added, and changed in place i when transition j fires once. Thus, if we view a marking μ as a k-dimensional column vector in which the ith component is $\mu(p_i)$, each column of $[T]$ is then a k-dimensional vector such that if $\mu_0 \overset{\sigma}{\longmapsto} \mu$, then the following *state equation* holds:

$$\mu_0 + [T] \cdot \#_\sigma = \mu,$$

where $\#_\sigma$ is an m-dimensional vector with its jth entry denoting the number of times transition t_j occurs in σ.

Example 2. Consider the PN in Figure 1. As marking $\mu = (0, 1, 2, 1, 0)$ is reachable from the initial marking $\mu_0 = (1, 0, 0, 1, 0)$ through the firing sequence $\sigma = t_2 t_1 t_2$, it is easy to verify (as the following equation shows) that the state equation $\mu_0 + [T] \cdot \#_\sigma = \mu$ holds.

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 2 \\ 1 \\ 0 \end{pmatrix}$$

□

As the following example shows, the existence of a solution for the state equation of a PN is necessary but not sufficient to guarantee reachability.

Example 3. Consider a PN $\mathcal{P}$=(P, T, φ) with $P = \{p_1, p_2, p_3\}$, $T = \{t_1, t_2\}$ and $\varphi(p_1, t_1) = -1$, $\varphi(t_1, p_2) = 1$, $\varphi(p_2, t_2) = -1$, $\varphi(t_2, p_1) = 1$, and $\varphi(t_2, p_3) = 1$. Let the initial marking be $(0, 0, 0)$ and the final marking be $(0, 0, 1)$. The associated state equation is

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} -1 & 1 \\ 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Clearly (1,1) is a solution to the above equation although $(0, 0, 1)$ is not reachable from $(0, 0, 0)$.

□

For ease of expression, the following notations will be used extensively throughout the rest of this paper. (Let σ, σ' be transition sequences, p be a place, and t be a transition.)

- $\#_\sigma(t)$ represents the number of occurrences of t in σ. (For convenience, we sometimes treat $\#_\sigma$ as an m-dimensional vector assuming that an ordering on T is established ($|T| = m$).)
- $\Delta(\sigma) = [T] \cdot \#_\sigma$ defines the *displacement* of σ. (Notice that if $\mu_0 \xmapsto{\sigma} \mu$, then $\Delta(\sigma) = \mu - \mu_0$.) For a place $p \in P$, we write $\Delta(\sigma)(p)$ to denote the component of $\Delta(\sigma)$ corresponding to place p.
- $Tr(\sigma) = \{t | t \in T, \#_\sigma(t) > 0\}$, denoting the set of transitions used in σ.
- $p^\bullet$=$\{t | \varphi(p,t) \geq 1, t \in T\}$ is the set of output transitions of p;
  $t^\bullet$=$\{p | \varphi(t,p) \geq 1, p \in P\}$ is the set of output places of t.

- ${}^\bullet p$=$\{t | \varphi(t,p) \geq 1, t \in T\}$ is the set of input transitions of p;
  ${}^\bullet t$=$\{p | \varphi(p,t) \geq 1, p \in P\}$ is the set of input places of t.

Given $\mu_0 \xmapsto{\sigma} \mu$, a sequence σ' is said to be a *rearrangement* of σ if $\#_\sigma = \#_{\sigma'}$ and $\mu_0 \xmapsto{\sigma'} \mu$.

Let $\mathcal{P} = ((P, T, \varphi), \mu_0)$ be a marked PN. The *reachability set* of $\mathcal{P}$ is $R(\mathcal{P}, \mu_0) = \{\mu \mid \exists \sigma \in T^*, \mu_0 \xmapsto{\sigma} \mu\}$.

2.2 Vector addition systems (with states), vector replacement systems

Vector addition systems (VAS) were introduced by Karp and Miller [36], and were later shown by Hack [18] to be equivalent to PNs. An n-dimensional VAS is a pair $G = (x, W)$, where $x \in N^n$ is called the *start point* (or *start vector*) and W is a finite set of vectors (called *addition vectors*) in Z^n. The *reachability set* of the VAS G is the set $R(G) = \{z \mid$ for some j, $z = x + v_1 + ... + v_j$, where, for all $1 \leq i \leq j$, each $v_i \in W$ and $x + v_1 + ... + v_i \geq 0\}$.

An n-dimensional *vector addition system with states* (VASS) [21] is a VAS (x, W) together with a finite set T of transitions of the form $p \to (q, v)$, where q and p are states and v is in W. The meaning is that such a transition can be applied at point y in state p and yields the point $y + v$ in state q, provided that $y + v \geq 0$. The VASS is specified by $G = (x, W, T, p_0)$, where p_0 is the starting state.

Example 4. For the PN shown in Figure 1, the corresponding VAS $\langle x, W \rangle$ is:

- $x = (1, 0, 0, 1, 0)$,
- $W = \{(1, -1, 0, 0, 0), (-1, 1, 1, 0, 0), (0, 0, -1, -1, 1), (0, 0, 0, 1, -1)\}$.

Such a VAS can also be regarded as a VASS with a single state. □

A $k \times m$ *vector replacement system (VRS)* [37] is a triple (w_0, U, W), where $w_0 \in N^k$ *(start vector)*, $U \in N^{k\times m}$ *(check matrix)*, and $W \in Z^{k\times m}$ *(addition matrix)* such that, for any i, j with $1 \leq i \leq m$ and $1 \leq j \leq k$, we have $U_i(j) + W_i(j) \geq 0$. Here U_i (respectively, W_i) is the i-th column vector of U (respectively, W). A vector $W_i \in W$ is said to be *enabled* in a vector $x \in N^k$ if and only if $x \geq U_i$; as $U_i + W_i \geq 0$, adding W_i to x yields $x + W_i \in N^k$. For a VRS $G = (w_0, U, W)$, $R(G)$ denotes the set of vectors from N^k that can be reached from w_0 by iteratively adding vectors from W enabled in the vector computed so far.

It is known that Petri net, VAS, VASS, and VRS are computationally equivalent. In fact, given an n-dimensional VASS G, we can effectively construct an $(n + 3)$-dimensional VAS G' that simulates G [21].

3 Petri net problems

What follows are problems that are of particular importance and interest in the study of PNs.

- The *reachability* problem: given a PN $\mathcal{P}$ (with initial marking μ_0) and a marking μ, deciding whether $\mu \in R(\mathcal{P}, \mu_0)$.

- The *boundedness* problem: given a PN $\mathcal{P}$ (with initial marking μ_0), deciding whether $|R(\mathcal{P}, \mu_0)|$ is finite or not.

- The *covering* problem: given a PN $\mathcal{P}$ (with initial marking μ_0) and a marking μ, deciding whether there exists a $\mu' \in R(\mathcal{P}, \mu_0)$ such that $\mu' \geq \mu$.

- The *equivalence* problem: given two PNs $\mathcal{P}_1$ (with initial marking μ_1) and $\mathcal{P}_2$ (with initial marking μ_2), deciding whether $R(\mathcal{P}_1, \mu_1) = R(\mathcal{P}_2, \mu_2)$.

- The *containment* problem: given two PNs $\mathcal{P}_1$ (with initial marking μ_1) and $\mathcal{P}_2$ (with initial marking μ_2), deciding whether $R(\mathcal{P}_1, \mu_1) \subseteq R(\mathcal{P}_2, \mu_2)$.

- The *model checking* problem: given a PN $\mathcal{P}$ (with initial marking μ_0) and a temporal formula ϕ (expressed in some temporal logic), deciding whether $\mathcal{P}, \mu_0 \models \phi$ (i.e., $(\mathcal{P}, \mu_0)$ satisfies ϕ).

- The *liveness* problem: given a PN $\mathcal{P}$ (with initial marking μ_0), deciding whether for every $t \in T, \mu \in R(\mathcal{P}, \mu_0)$, there exists a sequence of transitions σ such that $\mu \xmapsto{\sigma \cdot t}$, i.e., t is enabled after firing σ from μ.

- Others include *home-state*, *reversibility*, *self-stabilization*, *fairness*, *regularity*, *synchronic distance*, *controllability* ... and more.

Example 5. Consider the two PNs shown in Figure 2. Clearly, Figure 2(a) is bounded with respect to the given initial marking. However, if the upper-leftmost place contains two tokens, then the PN becomes unbounded. That is, being boundedness or not depends on the initial marking for the PN in Figure 2(a). The PN in Figure 2(b), on the other hand, remains bounded no matter what the initial marking of the PN is. Such a PN is called *structurally bounded*.

□

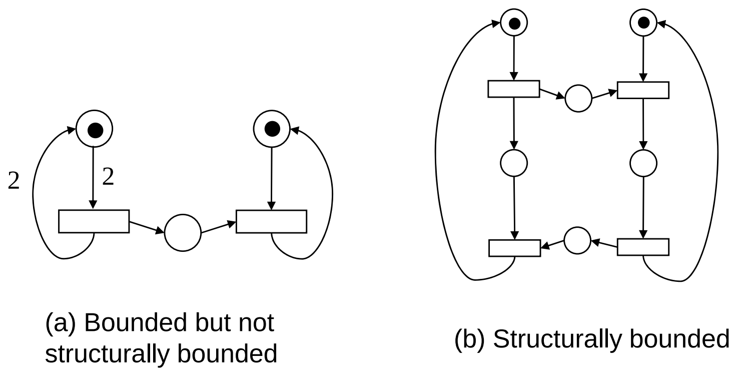


Fig. 2. Structural vs. behavioral boundedness.

To capture the essence of a transition being 'live' in various application areas, a hierarchy of liveness notions was defined in the literature (see [49]). Transition t in a PN $(\mathcal{P}, \mu_0)$ is said to be:

1. *Dead (L0-live)* if t can never be fired in any firing sequence from μ_0;
2. *L1-live (potentially firable)* if t can be fired at least once in some firing sequence from μ_0;
3. *L2-live* if, given any positive integer k, t can be fired at least k times in some firing sequence from μ_0;
4. *L3-live* if t appears infinitely often in some firing sequence from μ_0;
5. *L4-live* or *live* if t is *L1-live* for every marking μ in $R(\mathcal{P}, \mu_0)$;

A PN is said to be *L0*, *L1*, *L2*, *L3*, and *L4*-live if each of its transitions is *L0*, *L1*, *L2*, *L3*, and *L4*-live, respectively.

Example 6. Consider the PN shown in Figure 3. For any $k \in N$, $(1, 0, 0) \overset{(t_1)^k t_2 (t_3)^k}{\longmapsto}$; hence, t_3 is *L2-live*. However, it is reasonably easy to see that t_3 is not *L3-live* as there is no computation along which t_3 is fired infinitely many times. In fact, the following implications hold: L4-liveness (the strongest) $\Longrightarrow$ L3-liveness $\Longrightarrow$ L2-liveness $\Longrightarrow$ L1-liveness. □

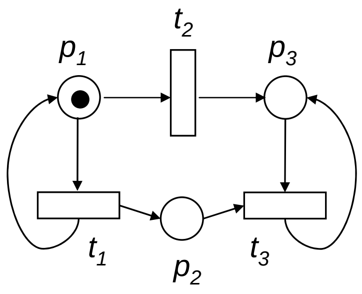


Fig. 3. L2 vs. L3 liveness.

4 Analytical techniques

In this section, we summarize various techniques useful for analyzing PN properties. Our focus is on *algebraic techniques*, *structural analysis*, and *state-space analysis*. Other techniques such as *simulation* and *synthesis/reduction* are beyond the scope of our discussion. Structural analysis is mainly designed for reasoning about properties of PNs that are independent of the initial markings. State-space analysis, on the other hand, allows us to infer properties of PNs that are sensitive to the initial markings.

4.1 Algebraic techniques

In the framework of using algebraic techniques for reasoning about PNs, solving a PN problem is reduced to finding a solution for an algebraic (in)equation associated with the PN. Due to the nature of this technique, the method is in general efficient (in most cases, polynomial in the size of the PN). Unfortunately, this technique generally provides only necessary or sufficient information for either inferring desired properties or ruling out dangerous conditions.

State equation: Figure 4 highlights the idea behind the technique based on *state equations*. If μ is reachable from μ_0 through transition sequence σ, then $\#_\sigma$ corresponds to an integer solution to the state equation $\mu_0 + [T] \cdot \#_\sigma = \mu$, where $[T]$ is the incidence matrix of the PN. The technique relies on relating the PN reachability analysis to *integer linear programming*, which is a well-established formalism. Unfortunately, a direct application of the state equation technique to general PN problems is normally not feasible, as the existence of a solution to a state equation is necessary but not sufficient for witnessing reachability. There are, however, various subclasses of PNs for which an extended state equation is sufficient and necessary to capture reachability of the underlying PN. More will be said about this in our subsequent discussion.

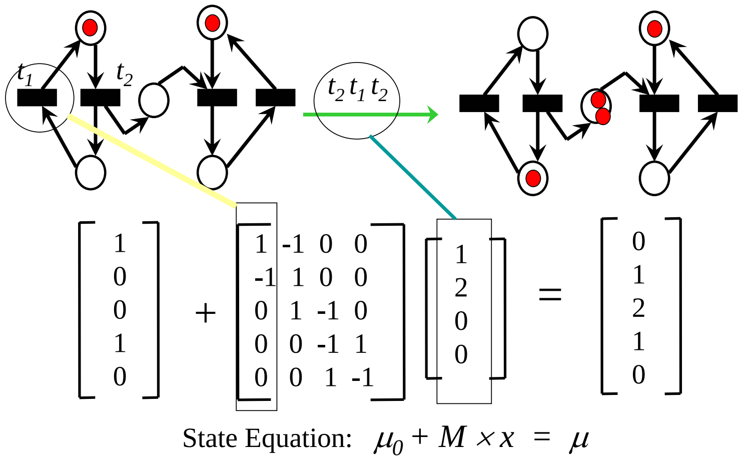


Fig. 4. State equation.

Place invariant: A *place invariant* of PN $\mathcal{P} = (P, T, \varphi)$ is a mapping Inv_P: $P \to Z$ (i.e., assigning *weights* to places) such that $\forall \mu, \mu'$ and $t \in T$, if $\mu \overset{t}{\longmapsto} \mu'$, then $\sum_{p \in P} Inv_P(p)\mu(p) = \sum_{p \in P} Inv_P(p)\mu'(p)$. In words, the firing of any transition does not change the weighted sum of tokens in the PN. Consider the PN shown in Figure 5. It is reasonably easy to observe that $(1, 2, 1)$ is a P-invariant. Other P-invariants include $(1, 1, 0), (2, 5, 3), (-2, 1, 3)$. Note that any linear combination of P-invariants is a P-invariant. Any solution of the equation $X \cdot [T] = 0$ is a P-invariant, where X is a row vector. For instance,

$$\begin{pmatrix} 1, 2, 1 \end{pmatrix} \begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

as (1,2,1) is a P-invariant. Using the so-called Farkas Algorithm, the minimal P-invariants (i.e., bases) of a PN can be calculated. Nevertheless, in the worst case the number of minimal P-invariants is exponential in the size of the PN, indicating that Farkas Algorithm may require exponential worst-case time.

Suppose μ is a reachable marking (from the initial marking μ_0) through a firing sequence σ. Clearly, $\mu_0 + [T]\#_\sigma = \mu$. (Here μ_0 and μ are column vectors.) Let X be a P-invariant. Then
$X \cdot \mu \quad = \quad X \cdot (\mu_0 + [T] \cdot \#_\sigma) \quad = \quad X \cdot \mu_0 + X \cdot [T] \cdot \#_\sigma \quad = \quad X \cdot \mu_0.$
Recall that in the example in Figure 5, $(1, 1, 0)$ is a P-invariant. For every reachable marking μ, we have $\mu(p_1) + \mu(p_2) = \mu_0(p_1) + \mu_0(p_2)$, meaning that the total number of tokens in p_1 and p_2 together remain unchanged during the course of the PN computation. Hence, if the PN starts from the initial marking $(1, 0, 1)$, then the property of *mutual exclusion* for places p_1 and p_2 can be asserted as $\mu(p_1) + \mu(p_2) = \mu_0(p_1) + \mu_0(p_2) = 1$ for all reachable marking μ. It is easy to see that if there exists a P-invariant X with $X(p) > 0$, for all $p \in P$, then the PN is

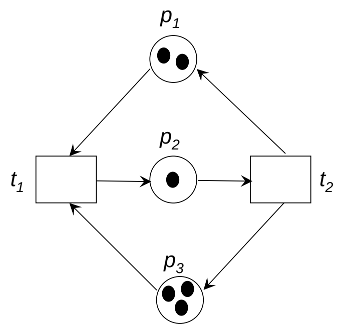


Fig. 5. A PN with a P-invariant (1,2,1).

guaranteed to be structurally bounded. Hence, *place invariants* can be used for reasoning about structural boundedness.

Transition invariant: A *transition invariant* of PN $\mathcal{P} = (P, T, \varphi)$ is a mapping $Inv_T : T \rightarrow N$ (i.e., assigning nonnegative *weights* to transitions) such that $\Sigma_{t \in T} Inv_T(t)(\Delta(t)) = 0$. In words, firing each transition the number of times specified in the T-invariant brings the PN back to its starting marking. Again consider the PN shown in Figure 5 in which $(2, 2)$ (i.e., $Inv_T(t_1) = Inv_T(t_2) = 2$) is clearly a T-invariant, so is (n, n), for arbitrary $n \geq 0$. Like P-invariants, any linear combination of T-invariants is a T-invariant. It is easy to see that T-invariants correspond to the solutions of the following equation: $[T] \cdot X^T = 0$ (where X is a row vector representing a T-invariant). The existence of a T-invariant is a necessary condition for a bounded PN to be live. To see this, suppose $\mathcal{P} = ((P, T, \varphi), \mu_0)$ is a live and bounded PN. Clearly due to $\mathcal{P}$ being live, there exists an infinite path $\mu_0 \overset{\sigma_1}{\longmapsto} \mu_1 \overset{\sigma_2}{\longmapsto} \cdots \mu_i \overset{\sigma_{i+1}}{\longmapsto} \mu_{i+1} \cdots$ such that $Tr(\sigma_i) = T$ for all $i \geq 1$ (i.e., σ_i uses all the transitions in T). Since $\mathcal{P}$ is also bounded, there exist $h > j \geq 0$ such that $\mu_j = \mu_h$; hence, $\#_{(\sigma_{j+1} \cdots \sigma_h)}$ constitutes a T-invariant.

4.2 Structural analysis

Given a PN $\mathcal{P} = (P, T, \varphi)$, a subset of places $S \subseteq P$ is called a *trap* (resp., *siphon*) if $S^\bullet \subseteq {}^\bullet S$ (resp., ${}^\bullet S \subseteq S^\bullet$). (Here ${}^\bullet S$ and $S^\bullet$ denote the sets of input and output transitions of S, respectively.) Intuitively, a trap S represents a set of places in which every transition consuming a token from S must also deposit a token back into S. In contrast, if a transition is going to deposit a token to a place in a siphon S, the transition must also remove a token from S.

Suppose S is a siphon in a live PN without isolated places (i.e., places without input/output transitions), then S must be marked in the initial marking μ_0 (i.e., $\mu_0(p) > 0$, for some place $p \in S$). Otherwise, none of the transitions in ${}^\bullet S$ or $S^\bullet$ is fireable – violating the assumption that the PN being live.

The concept of a *siphon* plays an important role in the liveness analysis for the class of *free-choice* PNs. A PN $\mathcal{P} = (P, T, \varphi)$ is called a *free-choice* PN if the following conditions hold: (1) $\forall t \in T, p \in P, \varphi(p,t) \leq 1$ and $\varphi(t,p) \leq 1$, (2) $\forall t_1, t_2 \in T, ({}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset \implies {}^\bullet t_1 = {}^\bullet t_2)$. In words, if two transitions share some input places, then they share all their input places. Known as Commoner's Theorem (see, e.g., [7]), a free-choice PN is live iff every nonempty siphon contains an initially marked trap. The reader is referred to [7] for more about free-choice PNs.

Example 7. Consider the free-choice PN shown in Figure 6. It is easy to see that $\{p_1, p_2, p_5, p_6, p_7\}$ is a siphon which does not contain any initially marked trap. Hence, the PN is not live due to Commoner's Theorem. In fact, $(1,0,0,0,0,0,0) \xmapsto{t_1} (0,0,0,0,0,1,1) \xmapsto{t_2} (0,1,0,0,0,0,1) \xmapsto{t_5} (0,1,0,0,1,0,0)$ reaches a dead marking. □

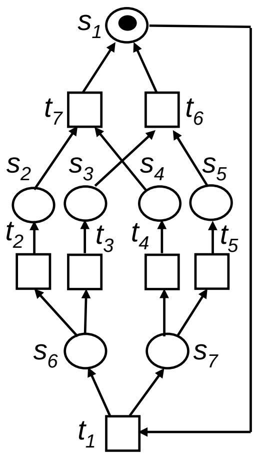


Fig. 6. A free-choice PN which is bounded but not live.

4.3 State space analysis

Reachability graph analysis: The so-called *reachability graph analysis* is perhaps the simplest and the most straightforward approach for analyzing the behavior of a PN. As its name suggests, such a technique relies on exhaustively generating all the reachable markings from a given initial marking, in hope of deducing PN properties by examining the structure of the reachability graph. Figure 7 displays a portion of the reachability graph associated with the PN in Figure 1.

In spite of its simplicity, the applicability of the technique of reachability graph analysis is rather limited; it can only be applied to bounded (i.e., finite)

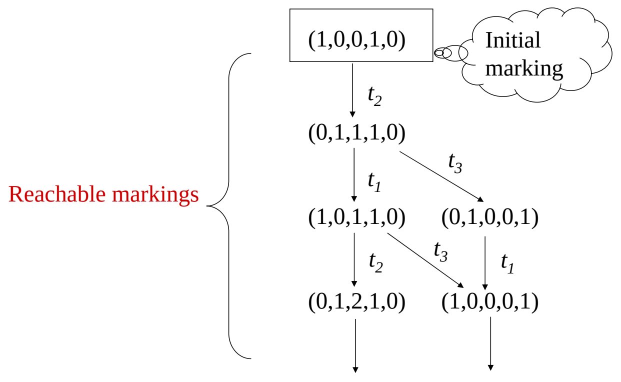


Fig. 7. Portion of the reachability graph of the Petri net in Figure 1.

PNs with small reachability sets. Even for bounded PNs which exhibit finite reachbility graphs, the technique is 'expensive' in the sense that it suffers from the *state explosion phenomenon* as the sizes of the reachability sets grow beyond any primitive recursive function even for bounded PNs in the worst case. More will be said about this later.

Coverability graph analysis: *Coverability graph analysis* offers an alternative to the technique of reachability graph analysis by abstracting out certain details to make the graph finite. To understand the intuition behind coverability graphs, consider Figure 7 which shows (part of) the reachability graph of the PN in Figure 1. Consider the path $(1,0,0,1,0) \xmapsto{t_2} (0,1,1,1,0) \xmapsto{t_1} (1,0,1,1,0)$ along which the third coordinate gains an extra token in the end (i.e., $(1,0,1,1,0) > (1,0,0,1,0)$). Clearly the third coordinate can be made arbitrarily large by repeating t_2t_1 for a sufficient number of times, as $(1,0,0,1,0) \xmapsto{t_2t_1} (1,0,1,1,0) \xmapsto{t_2t_1} (1,0,2,1,0) \xmapsto{t_2t_1} \cdots \xmapsto{t_2t_1} (1,0,n,1,0) \xmapsto{t_2t_1} \cdots$, for arbitrary n. In order to capture the notion of a place being *unbounded*, we short-circuit the above infinite sequence of computation as $(1,0,0,1,0) \xmapsto{t_2} (0,1,1,1,0) \xmapsto{t_1} (1,0,\omega,1,0)$, where ω is a symbol denoting something being arbitrarily large. One can regard ω as "infinity" having the property that $\omega > n$ for any integer n, $\omega + n = n + \omega = \omega$, $\omega - n = \omega$ and $\omega \geq \omega$. A coverability graph relates each node to a *general marking* ($\in (N \cup \{\omega\})^{|P|}$) of the original PN. The corresponding coverability graph of the PN in Figure 1 is depicted in Figure 8.

The algorithm for generating the coverability graph of a PN is shown below (see [36]):

Coverability graph aglorithm
Input: A Petri net $\mathcal{P} = (P, T, \varphi)$ with the initial marking μ_0
Output: The coverability graph $G_{\mathcal{P}}(\mu_0)$ of PN $(\mathcal{P}, \mu_0)$
(1) Create a node μ_{init} such that $\mu_{init} = \mu_0$, and mark it as "new"

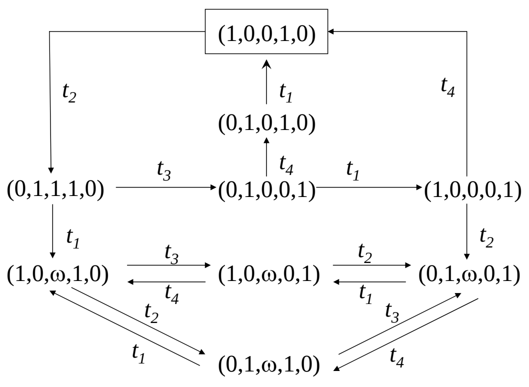


Fig. 8. Coverability graph of the Petri net in Figure 1.

```
(2)  **while** there contains some new node $\mu$ **do**
(3)      **for** each transition $t$ enabled at $\mu$ **do**
(4)          **case (i)** there is a node $\mu' = \mu + \Delta t$ in $G_{\mathcal{P}}(\mu_0)$
(5)                  add an edge $\mu \xrightarrow{t} \mu'$ to $G_{\mathcal{P}}(\mu_0)$
(6)          **case (ii)** there is a node $\mu''$ from $\mu_{init}$ to $\mu$ such that $\mu'' < \mu + \Delta t$
(7)                  add an "new" node $x$ with
(8)                      $x(p) = \omega$ if $\mu''(p) < (\mu + \Delta t)(p)$
(9)                      $x(p) = \mu''(p)$, otherwise
(10)                 add an edge $\mu \xrightarrow{t} x$
(11)         **case (iii)** otherwise
(12)                 add an "new" node $x$ with $x = \mu + \Delta t$ and an edge $\mu \xrightarrow{t} x$
(13)     **end for**
(14)     mark $\mu$ with "old"
(15) **end while**
*end algorithm*
```

As it turns out, the coverability graph of a PN is always finite ([36]). In addition, a PN is unbounded iff an ω occurs in the corresponding coverability graph, which, in turn, yields a decision procedure for deciding the boundedness property. It should be noted that such a technique does not answer the reachability problem as ω abstracts out the exact number of tokens that a place can accumulate, should the place be potentially unbounded.

A reachability graph (possibly of infinite size) captures the exact information about the set of reachable markings of a PN, whereas a coverability graph (always of finite size) provides an over-approximation of the reachability set, should it be infinite. Again the coverability graph analysis suffers from *state explosion.*

5 Complexity analysis of various Petri net problems

5.1 Boundedness and covering

The boundedness problem was first considered by Karp and Miller in [36], where it was shown to be decidable using the technique of *coverability graph analysis*. The algorithm presented there was basically an unbounded search and consequently no complexity analysis was shown. Subsequently, a lower bound of $O(2^{c \times m})$ space was shown by Lipton in [40], where m represents the dimension of the problem instance and c is some constant. Finally, an upper bound of $O(2^{c \times n \times logn})$ space was given by Rackoff in [56]. Here, however, n represents the size or number of bits in the problem instance and c is a constant.

Lipton's exponential space lower bound was established by constructing a VAS to maintain and store a number, whose value ranges between 0 and 2^{2^k}. This number could then be incremented by 1 (as long as the current value was below the upper limit), decremented by 1 (as long as the current value exceeded 0), and tested for zero. The zero-test was the hard part in Lipton's construction. See [59] for a refinement of Lipton's lower bound in the framework of multiparameter analysis.

Rackoff's exponential space upper bound was established using induction (on the dimension of the VAS instance). Such a technique has found additional applications in [14, 17, 59]. In particular, Rosier and Yen [59] refined Rackoff's strategy to derive a *multiparameter analysis* of the boundedness problem for VASSs, yielding an upper bound of $O(2^{c \times k \times logk}(l + logn))$ space, where k is the dimension, n is the number of states, and l is the length of the binary representation of the largest number mentioned in the VASS. A direct consequence of the above is that when the dimension of a VASS (VAS, or PN) is a fixed constant, then the boundedness problem is solvable in polynomial space. As the strategy behind Rackoff's proof is interesting and important in its own right (as was witnessed by its additional applications in [14, 17, 59]), in what follows we briefly describe the key steps of the proof.

Let (v, A) be a VAS of size n. It is well known that (v, A) is unbounded iff there is a computation $v \overset{*}{\longmapsto} v' \overset{*}{\longmapsto} v''$ such that $v'' > v'$. Rackoff's strategy relies on showing that if such a path exhibiting unboundedness exists, then there is 'short' witness.

A $w \in Z^k$ is called *i-bounded* (resp., *i-r bounded*) if $0 \leq w(j)$, $\forall 1 \leq j \leq i$ (resp. $0 \leq w(j) \leq r$, $\forall 1 \leq j \leq i$). Let $p = w_1 w_2 \cdots w_m$ be a sequence of vectors in Z^k. Sequence p is said to be

- *i-bounded* (resp., *i-r bounded*) if every member of p is *i-bounded* (resp., *i-r bounded*).
- *self-covering* if there is a $1 \leq j \leq m$ such that $w_m > w_j$.
- an *i-loop* if $w_m(j) = w_1(j), \forall 1 \leq j \leq i$.

Let $m(i, v)$ be the length of the shortest i-bounded self-covering path in (v, A); =0 if no such path exists. Also let $g(i) = max\{m(i, v) : v \in Z^k\}$. Note that $g(i)$ represents the length of the longest i-bounded self-covering path from any starting vector in Z^k. The key in Rackoff's proof relies on finding a bound for $g(i)$ inductively, where $i = 1, 2, ..., k$.

To derive $g(i)$, it was shown in [56] that if there is an $i - r$ bounded self-covering path in (v, A), then there is a 'short witness' of length bounded by r^{n^c}, where c is a constant independent of (v, A). The proof of this result relies on rearranging as well as chopping off unnecessary i-loops along an $i - r$ bounded self-covering path, using a result in [3] concerning bounds of integer solutions of linear (in)equations. Using the above result, it could be shown that $g(0) \leq 2^{n^c}$ and $g(i+1) \leq (2^n g(i))^{n^c}$, $0 \leq i \leq k-1$. To see this, let $p : v_1 \cdots v_m$ be any $(i+1)$-bounded self-covering path. Consider two cases:

- Case 1: Path p is $(i+1) - (2^n g(i))$-bounded. Then the length of p is $\leq (2^n g(i))^{n^c}$, which follows immediately from the earlier result concerning short witnessing paths.
- Case 2: Otherwise, let v_h be the first vector along p that is not $(2^n g(i))$ bounded. By removing $(i+1)$-loops, the prefix $v_1 ... v_h$ can be shortened (if necessary) to make the length $\leq (2^n g(i))^{i+1}$. With no loss of generality, we assume the $(i+1)$st position to be the coordinate whose value exceeds $2^n g(i)$ at v_h. Recalling the definition of $g(i)$, there is a self-covering path, say l, of length $\leq g(i)$ from v_h. By appending l to $v_1 ... v_h$ (i.e., replacing the original suffix path $v_h ... v_m$ by l), the new path is an $(i+1)$-bounded self-covering path, because the value of the (i+1)st coordinate exceeds $2^n g(i)$ and the path l (of length bounded by $\leq g(i)$) can at most subtract $(2^n g(i))$ from coordinate $i+1$. (Note that the application of an addition vector can subtract at most 2^n from a given coordinate.)

By solving the recurrence relation $g(0) \leq 2^{n^c}$ and $g(i+1) \leq (2^n g(i))^{n^c}$, $0 \leq i \leq k-1$, the length of the shortest path witnessing unboudedness is $\leq 2^{2^{c \times n \times logn}}$. A nondeterministic search immediately yields a $O(2^{c \times n \times logn})$ space complexity for the boundedness problem.

The complexity (both upper and lower bounds) of the covering problem can be derived along a similar line of that of the boundedness problem. See [56] for details.

5.2 Reachability

Of various problems of interest in the study of PNs, the reachability problem is perhaps the one that has attracted the most attention in the PN community in the past four decades. One reason is that the problem has many real-world applications; furthermore, it is the key to the solutions of several other PN problems (such as liveness).

Before the decidability question of the reachability problem for general PNs was answered in the affirmative by Mayr [42, 43] in the early 1980's (see also [38]), a number of attempts were made to investigate the problem for restricted classes of PNs, in hope of gaining more insights and developing new tools in order to conquer the general PN reachability problem. Before Mayr's proof, Sacerdote and Tenney [60] claimed the reachability problem to be decidable; yet they failed to provide a convincing proof. What follows are notable milestones along this line of research.

In 1974, van Leeuwen [61] first showed the reachability problem to be decidable for 3-dimensional PNs. Hopcroft and Pansiot [21] later extended van Leeuwen's finding to 5-dimensional PNs in 1979. About the same time Landweber and Robertson [39] as well as Grabowski [16], Mayr [41] and Muller [48] considered PNs on which either structural or behavioral constraints are imposed, and showed the reachability problem to be decidable for the classes of *conflict-free* and *persistent* PNs. A important common feature of the above attempts is that the decidability result was built upon showing the reachability set to be *semilinear*. As it turns out, semilinearity is also preserved for the so-called *normal* [62], *sinkless* [62], and *communication-free* PNs (also known as *BPP nets*) [28, 10].

Although the reachability problem for general PNs is known to be decidable, no complexity analysis was given in [42, 43], (nor in [38]). The best known lower bound for the problem is exponential space hard, which is identical to that of the boundedness problem. (In fact, Lipton's lower bound proof works for both the reachability and the boundedness problems.) In view of the importance of the reachability problem, finding the exact complexity of the problem remains one of the most important open problems in Petri net theory.

5.3 Containment and equivalence

In the late 1960's, Rabin first showed the containment problem for PNs to be undecidable. Even though the original work of Rabin was never published, a new proof was presented at a talk at MIT in 1972 [2]. In 1975, Hack [19] extended Rabin's result by showing the equivalence problem of PNs to be undecidable as well. Both undecidability proofs were based on *Hilbert's Tenth Problem* [6], a famous undecidable problem.

Hilbert's Tenth Problem is the problem of, given a polynomial $P(x_1, ..., x_n)$ over n variables with integer coefficients, deciding whether $P(x_1, ..., x_n) = 0$ has integer solutions. Reducing from Hilbert's Tenth Problem, it is not hard to see that the so-called *polynomial graph inclusion problem*, i.e., given two polynomials P and Q, deciding whether $\{(x_1, ..., x_n, y) \mid y \leq P(x_1, ..., x_n),\ \text{with}\ x_1, ..., x_n, y \in N\} \subseteq \{(x_1, ..., x_n, y) \mid y \leq Q(x_1, ..., x_n),\ \text{with}\ x_1, ..., x_n, y \in N\}$, is undecidable. The key behind Rabin's and Hack's proofs relies on showing PNs to be capable of *weakly computing* polynomials. By weakly computing a polynomial $P(x_1, ..., x_n)$, we mean a PN $\mathcal{P}$ with n designated places $p_1, ..., p_n$ (holding the n input values of P) and a designated 'output' place q can be constructed in such a way that for arbitrary input values $v_1, ..., v_n \in N$, starting from $v_1, ..., v_n$ tokens in places $p_1, ..., p_n$, respectively, $\mathcal{P}$ has the ability to deposit y tokens in place q for some $y \leq P(v_1, ..., v_n)$ when halting. With such capabilities of weakly computing polynomials, two PNs $\mathcal{P}$ and $\mathcal{Q}$ can be constructed from two given polynomials P and Q such that $R(\mathcal{P}) \subseteq R(\mathcal{Q})$ (or $R(\mathcal{P}) = R(\mathcal{Q})$) iff the answer to the polynomial graph inclusion problem regarding polynomials P and Q is positive.

It turns out that the equivalence problem remains undecidable with respect to several other notions of *equivalence*, including *trace equivalence*, *language equivalence* as well as *bisimulation equivalence* [32] for labelled PNs. In fact, it follows from [27] that all the equivalences under the interleaving semantics are undecidable.

5.4 Liveness

In [18], several variants of the reachability problem (including the general one) were shown to be recursively equivalent. Among them is the *single-place zero reachability problem*, i.e., the problem of determining whether a marking with no tokens in a designated place can be reached. Hack [18] also showed the *single-place zero reachability problem* to be recursively equivalent to the liveness problem. As a result, deciding whether a PN is live or not is decidable. Like the general reachability problem, the exact computational complexity of the liveness problem remains open.

5.5 Model checking

For some time, *temporal logic* has been considered a useful formalism for reasoning about systems of concurrent programs. A typical problem involving temporal logic is the *model checking* problem, i.e., the problem of determining whether a given structure defines a model of a correctness specification expressed in the temporal logic.

Before getting into the study of model checking for PNs, we require the following basic definitions of *linear-time temporal logic* (LTL) and a *branching-time temporal logic* called *computation tree logic* (CTL).

An *LTL well-formed formula* is defined as

$$F \mid \neg f \mid f \wedge g \mid \bigcirc f \mid f \, \mathcal{U} \, g$$

where F is a predicate, and f and g are well-formed formulas. For convenience, we also write $f \vee g \equiv \neg(\neg f \wedge \neg g)$, $\Diamond f \equiv \mathit{true} \, \mathcal{U} \, f$ and $\Box f \equiv \neg \Diamond \neg f$. Intuitively, $\bigcirc$, $\mathcal{U}$, $\Diamond$, and $\Box$ are temporal operators denoting *next-time*, *until*, *eventually*, and *always*, respectively. LTL formulas are interpreted on *computation paths.* With respect to a computation path $s_0 \to s_1 \to \cdots$, the intuitive meanings of the the above formulas are:

1. F: the atomic predicate F holds at s_0,
2. $\neg f$: formula f does not hold at s_0,
3. $f \wedge g$: both f and g hold at s_0,
4. $\bigcirc f$: f holds at s_1 (i.e., the immediate successor of s_0),
5. $f \, \mathcal{U} \, g$: there exists an $i \geq 0$ such that f holds at $s_0, ..., s_{i-1}$ and g holds at s_i.

A *CTL formula* is defined as

$$F \ \mid\ \neg f \ \mid\ f \wedge g \ \mid\ \exists \bigcirc f \ \mid\ \forall \bigcirc f \ \mid\ \exists(f\ \mathcal{U}\ g) \mid \forall(f\ \mathcal{U}\ g)$$

where F is a predicate, f and g are CTL formulas, and $\exists$ and $\forall$ are path quantifiers denoting 'there exists a path' and 'for all paths', respectively. Among others, useful abbreviations include: $\exists\Diamond f \equiv \exists(true\ \mathcal{U} f)$; $\forall\Diamond f \equiv \forall(true\ \mathcal{U} f)$; $\forall\Box f \equiv \neg\exists\Diamond(\neg f)$; $\exists\Box f \equiv \neg\forall\Diamond(\neg f)$.

CTL formulas are interpreted on *computation trees*. With respect to a tree rooted at s_0, the intuitive meanings of the formulas mentioned above are:

1. F , $\neg f$ and $f \wedge g$ are the same as those in the LTL case,
2. $\exists \bigcirc f$: there exists a *child* s of s_0 such that f holds at s,
3. $\forall \bigcirc f$: for every *child* s of s_0, f holds at s,
4. $\exists(f\ \mathcal{U}\ g)$: there exists a computation path l from s_0 such that $f\ \mathcal{U}\ g$ holds with respect to l,
5. $\forall(f\ \mathcal{U}\ g)$: for every computation path l emanating from s_0, $f\ \mathcal{U}\ g$ holds with respect to l.

The *model checking problem* for PNs is the problem of, given a PN $\mathcal{P}$ (with initial marking μ_0) and a temporal formula ϕ (expressed in some temporal logic), deciding whether the computation of $\mathcal{P}$ from μ_0 satisfies ϕ.

Model checking Petri nets was first investigated by Howell, Rosier, and Yen in [24, 25]. As was shown in [24], the model checking problem for a fairly simple temporal logic is undecidable, even for the class of conflict-free PNs. Following a number of subsequent work on the study of model checking for PNs (see, e.g., [31, 44, 11, 17]) from a decidability/complexity viewpoint, we now have a reasonably clear picture. Consider two types of atomic predicates: *state-based* and *action-based* predicates. As their names suggest, a state-based predicate applies to the 'markings' of a PN computation, whereas the value of an action-based predicate depends only the 'actions' taken along the computation.

With respect to *state-based* predicates, the model checking problem is undecidable for both linear-time and branching-time temporal logics. The undecidability result holds for branching-time, action-based temporal logics as well. Interestingly, however, model checking for linear-time, action-based temporal logics turns out to be decidable [17].

To show the model checking problem for linear-time, state-based temporal logic to be undecidable, we reduce from the containment problem of PNs. Given two PNs N_1 and N_2, we construct a PN N shown in Figure 9, in which r_1 and r_2 serve as 'control' places for controlling the firings of transitions in N_1 and N_2, respectively. For each place p_i in N_1, let p'_i be the corresponding place in N_2. A transition t_i is introduced to remove an equal amount of tokens from p_i and p'_i. It is not hard to see that

$$R(N_1) \subseteq R(N_2)$$

$$iff$$

$$\Box(\ ((r_1 = 1 \wedge r_2 = 0) \wedge \bigcirc(r_1 = 0 \wedge r_2 = 1)) \Rightarrow \Diamond(\bigwedge_i (p_i = 0 \wedge p'_i = 0))\ )$$

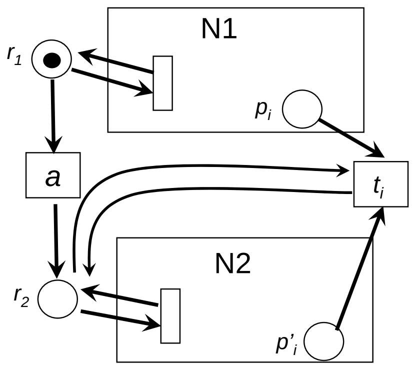


Fig. 9. Undecidability proof.

Note that $((r_1 = 1 \wedge r_2 = 0) \wedge \bigcirc(r_1 = 0 \wedge r_2 = 1))$ holds only at a marking at which transition a fires. Using a similar construction, the undecidability result for either state-based or action-based branching-time temporal logic can be shown.

The idea behind model checking an action-based linear-time temporal formula ϕ for PN $\mathcal{P}$ is the following. (See [17] for details.) Construct a *Buchi automaton* $M_{\neg\phi}$ to capture all the computations satisfying the negation of ϕ. Then it can be shown that $\mathcal{P}$ satisfies ϕ iff the intersection between the sets of computations of $\mathcal{P}$ and $M_{\neg\phi}$ is empty. By using a VASS to capture the 'Cartesian product' of $\mathcal{P}$ (equivalently, a VAS) and $M_{\neg\phi}$ (a finite automaton), the model checking problem is then reduced to finding certain infinite computations in a VASS, which turns out to be decidable.

5.6 Self-stabilization

Before the end of this section, let us elaborate a bit about the *self-stabilization* issue of PNs. The notion of *self-stabilization* was introduced by Dijkstra [8] to describe a system having the behavior that regardless of its starting configuration, the system would return to a 'legitimate' configuration eventually. (By a legitimate configuration, we mean a configuration which is reachable from the initial configuration of the system.) The motivation behind self-stabilization is that a self-stabilizing system has the ability to 'correct' itself even in the presence of certain unpredictable errors that force the system to reach an 'illegitimate' configuration during the course of its operations. In this sense, self-stabilizing systems exhibit fault-tolerant behaviors to a certain degree.

Let $\mathcal{S}$ be a (finite or infinite) *system* with c_0 as its *initial configuration.* Also let $R(\mathcal{S}, c_0)$ $(=\{c \mid c_0 \xrightarrow{*} c\})$ denote the set of *reachable configurations* from c_0. A computation σ from configuration c_1 is said to be *non-self-stabilizing* iff one of the following holds:

1. σ is finite ($\sigma : c_1 \xrightarrow{t_1} c_2 \xrightarrow{t_2} \cdots c_{m-1} \xrightarrow{t_{m-1}} c_m$, for some m) such that c_m is a dead configuration and $c_m \notin R(\mathcal{S}, c_0)$, <u>or</u>

2. σ is infinite ($\sigma : c_1 \xrightarrow{t_1} c_2 \xrightarrow{t_2} \cdots c_i \xrightarrow{t_i} c_{i+1} \cdots$) such that $\forall i \geq 1, c_i \notin R(\mathcal{S}, c_0)$.

See Figure 10. A system is said to be *self-stabilizing* if for each configuration c, none of the computations emanating from c is non-self-stabilizing. The *self-stabilization problem* is to determine, for a given (finite or infinite) system, whether the system is self-stabilizing.

The self-stabilization problem has only been scarcely studied in the Petri net community. It is known [4] that for bounded ordinary PNs (i.e. PNs without multiple arcs), the problem is PTIME-complete, whereas for bounded general PNs (i.e. PNs with multiple arcs), the problem becomes PSPACE-complete. For general unbounded PNs, the analysis of the self-stabilization problem remains open.

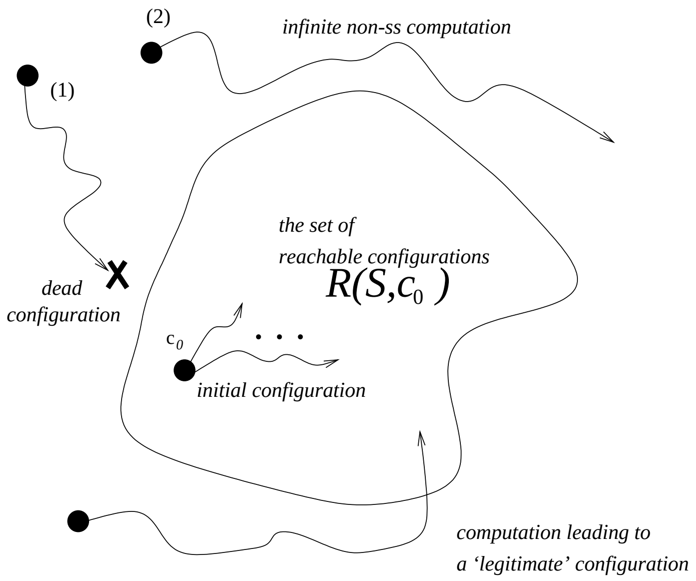
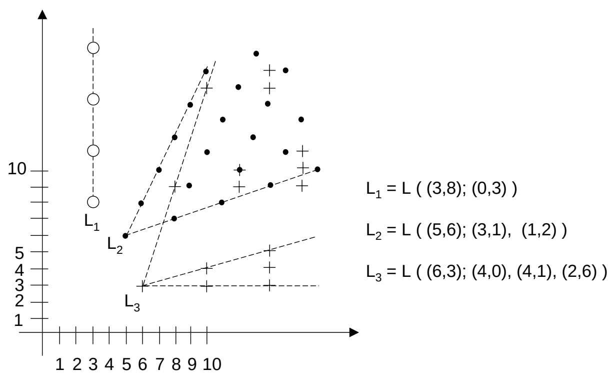


Fig. 10. Non-self-stabilizing computations.

6 Petri nets with semilinear reachability sets

The concept of *semilinearity* plays a key role not only in conventional automata theory and formal languages (see, e.g., [29]), but also in the analysis of PNs. A subset L of N^k is a *linear set* if there exist vectors $v_0, v_1, \ldots, v_t$ in N^k such that $L = \{v \mid v = v_0 + m_1 v_1 + \cdots + m_t v_t,\ m_i \in N\}$. The vectors v_0 (referred to as the *constant vector*) and $v_1, v_2, \ldots, v_t$ (referred to as the *periods*) are called the *generators* of the linear set L. For convenience, such a linear set is written as $\mathcal{L}(v_0; v_1, \ldots, v_t)$. A set $SL \subseteq N^k$ is *semilinear* [15] if it is a finite union of linear sets, i.e., $SL = \bigcup_{1 \le i \le m} \mathcal{L}_i$, where $\mathcal{L}_i$ ($\subseteq N^k$) is a linear set. The empty set is a trivial semilinear set. Every finite subset of N^k is semilinear – it is a finite union of linear sets whose generators are constant vectors. Figure 11 shows an example of a semilinear set, which consists of three linear sets L_1, L_2 and L_3. Clearly semilinear sets are closed under (finite) union. It is also known that they are closed under complementation and intersection. It is worthy of noting that semilinear sets are exactly those that can be expressed by *Presburger Arithmetic* (i.e., first order theory over natural numbers with addition) [55], which is a decidable theory.



10
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10
L_1
L_2
L_3
L_1 = L ((3,8); (0,3))
L_2 = L ((5,6); (3,1), (1,2))
L_3 = L ((6,3); (4,0), (4,1), (2,6))

Fig. 11. A semilinear set.

It is known [21] that for PNs of dimension 6 (equivalently, 6-dimensional vector addition systems, or 3-dimensional vector addition systems with states) or beyond, their reachability sets may not be semilinear in general.

For subclasses of PNs with semilinear reachability sets, a natural question to ask is: *What is the size of its semilinear representation?* An answer to the above question is key to the complexity analysis of various PN problems. In what follows, we survey several notable examples along the line of research of analyzing the sizes of semilinear representations for PNs with semilinear reachability sets.

Finite PNs: The reachability sets of finite (i.e., bounded) PNs are trivially semilinear. Mayr and Meyer [45] showed that the containment and equivalence problems for finite VASs are not primitive recursive. Subsequently, McAloon [46] showed that the problems are primitive recursive in the Ackermann function, and Clote [5], using Ramsey theory, showed the finite containment problem to be DTIME (Ackermann) complete. Using a different approach, Howell, Huynh, Rosier and Yen [22] showed an improvement of two levels in the primitive recursive hierarchy over results previously obtained by McAloon, thus answering a question posed by Clote.

2-dimensional VASS or 5-dimensional VAS (PN): It was first known by Hopcroft and Pansiot [21] that PNs of dimension 5 always have semilinear reachability sets. However, Hopcroft-Pnasiot algorithm does not reveal any upper bound on the size of the semilinear set representation, nor does it tell how quickly the set can be generated. In a subsequent study, Howell, Huynh, Rosier and Yen [22] gave a detailed analysis of the semilinear reachability sets of 2-dimensional VASSs, yielding the following result:

Given a 2-dimensional, n-state VASS V in which the largest integer mentioned can be expressed in l bits, we can construct in $DTIME(2^{2^{c\times l\times n}})$ (for some constant c) a semilinear reachability set representation $SL = \bigcup_{1\leq i\leq k} \mathcal{L}_i(x_i; P_i)$ such that, for some constants d_1, d_2, d_3,

1. $k = O(2^{2^{d_1\times l\times n}})$,
2. $\forall 1 \leq i \leq k$, $||x_i|| = O(2^{2^{d_2\times l\times n}})$
3. $\forall 1 \leq i \leq k$, $|P_i| = O(2^n)$
4. $\forall v \in P_i$, $\forall 1 \leq i \leq k$, $||v|| = O(2^{d_3\times l\times n})$

(Note that $||x||$ denotes the *1-norm* of vector x.)

Using the above result, the reachability, containment and equivalence problems for such VASSs were shown to be solvable in $DTIME(2^{2^{d\times l\times n}})$, for some constant d [22]. A matching lower bound was also established in [22].

Conflict-free, normal, sinkless, communication-free PNs: In this section, we employ a *decompositional approach* to serve as a unified framework for analyzing a wide variety of subclasses of PNs. Under this framework, answering the reachability question is equated with solving an instance of *integer linear programming*, which is relatively well-studied. To a certain extent, the decompositional approach can be thought of as a generalization of the *state equation* approach mentioned in our earlier discussion.

Before going into the details, the definitions of those subclasses of PNs for which the decompositional approach works are given first.

A *circuit* of a PN is simply a closed path (i.e., a cycle) in the PN graph. The presence of complex circuits is troublesome in PN analysis. In fact, strong evidence has suggested that circuits constitute the major stumbling block in the analysis of PNs. To get a feel for why this is the case, recall that in a PN $\mathcal{P}$ with initial marking μ_0, a marking μ is reachable (from μ_0) in $\mathcal{P}$ *only if* there exists a column vector $x \in \mathrm{N}^k$ satisfying the state equation $\mu_0 + [T]\cdot\ x = \mu$. The converse, however, does not necessarily hold. In fact, lacking a necessary and sufficient condition for reachability in general has been blamed for the high degree of complexity in the analysis of PNs. (Otherwise, one could have tied the reachability analysis of PNs to the *integer linear programming* problem, which is relatively well understood.) There are restricted classes of PNs for which necessary and sufficient conditions for reachability are available. Most notable, of course, is the class of *circuit-free* PNs (i.e., PNs without circuits) for which the equation $\mu_0 + [A]\cdot\ x = \mu$ is sufficient and necessary to capture reachability.

A slight relaxation of the circuit-freedom constraint yields the same necessary and sufficient condition for the class of PNs without *token-free* circuits in every reachable marking [62].

Formally, a *circuit* c of a PN is a sequence $p_1 t_1 p_2 t_2 \cdots p_n t_n p_1$ ($p_i \in P$, $t_i \in T$, $p_i \in {}^\bullet t_i$, $t_i \in {}^\bullet p_{i+1}$), such that $p_i \neq p_j, \forall i \neq j$ (i.e., all nodes except the first and the last are distinct along the closed path). We write $P_c = \{p_1, p_2, \cdots, p_n\}$ (resp., $T_c = \{t_1, t_2, \cdots, t_n\}$) to denote the set of places (resp., transitions) in c, and $tr(c)$ to represent the sequence $t_1 t_2 \cdots t_n$. We define the token count of circuit c in marking μ to be $\mu(c) = \sum_{p \in P_c} \mu(p)$. A circuit c is said to be *token-free* in μ iff $\mu(c) = 0$. Given two circuits c and c', c is said to be *included* (resp., *properly included*) in c' iff $P_c \subseteq P_{c'}$ (resp., $P_c \subset P_{c'}$). We say c is *minimal* iff it does not properly include any other circuit. Circuit c is said to be a

- $\oplus$-circuit iff $\forall i, 1 \leq i \leq n, {}^\bullet t_i = \{p_i\}$ (i.e., t_i has p_i as its unique input place), or
- $\odot$-circuit iff $\forall t \in T, \sum_{p \in P_c} (\varphi(t,p) - \varphi(p,t)) \geq 0$ (i.e., no transition can decrease the token count of c).

In this section, we elaborate on a useful technique, called *decomposition*, using which the reachability relation for various subclasses of PNs is linked with integer linear programming [26, 50, 63, 64]. Among those for which our decompositional approach is applicable include:

- *Conflict-free Petri nets*: A PN $\mathcal{P} = (P, T, \varphi)$ is *conflict-free* [39] iff for every place p, either
  1. $|p^\bullet| \leq 1$, or
  2. $\forall t \in p^\bullet$, t and p are on a self-loop.

  In words, a PN is conflict-free if every place which is an input of more than one transition is on a self-loop with each such transition.
- *Normal Petri nets*: A PN $\mathcal{P} = (P, T, \varphi)$ is *normal* [62] iff for every minimal circuit c and transition $t_j \in T$, $\sum\limits_{p_i \in P_c} a_{i,j} \geq 0$. Intuitively, a PN is normal iff no transition can decrease the token count of a minimal circuit by firing at any marking, Alternatively, every minimal circuit of a normal PN is a $\odot$-circuit.
- *Sinkless Petri nets*: A PN $\mathcal{P} = (P, T, \varphi)$ is *sinkless* [62] iff each minimal circuit of $\mathcal{P}$ is sinkless.
- *BPP nets*: A PN $\mathcal{P} = (P, T, \varphi)$ is a *BPP net* [10, 28] iff $\forall$ $t \in$T, $|{}^\bullet t| = 1$, i.e., every transition has exactly one input place, implying that every circuit of a BPP net is a $\oplus$-circuit. BPP nets are also known as *communication-free PNs*.
- *Trap-circuit Petri nets*: A PN $\mathcal{P} = (P, T, \varphi)$ is a *trap-circuit* PN [30] iff for every circuit c in $\mathcal{P}$, P_c is a trap.
- *Extended trap-circuit Petri nets*: A PN $\mathcal{P} = (P, T, \varphi)$ is an *extended trap-circuit* PN [63] iff for every circuit c in $\mathcal{P}$, either P_c is a trap or c is a $\oplus$-circuit.

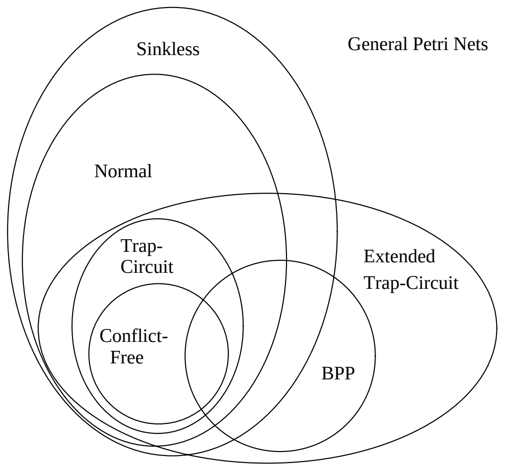


Fig. 12. Containment relationships among various Petri net classes.

The containment relationship among the above subclasses of PNs is depicted in Figure 12.

The idea behind the decompositional technique relies on the ability to decompose a PN $\mathcal{P}=(P, T, \varphi)$ (possibly in a nondeterministic fashion) into sub-PNs $\mathcal{P}_i = (P, T_i, \varphi_i)$ ($1 \leq i \leq n$, $T_i \subseteq T$, and φ_i is the restriction of φ to $(P \times T_i) \cup (T_i \times P)$) such that for an arbitrary computation $\mu_0 \xmapsto{\sigma} \mu$ of PN $\mathcal{P}$, σ can be rearranged into a canonical form $\sigma_1\sigma_2 \cdots \sigma_n$ with $\mu_0 \xmapsto{\sigma_1} \mu_1 \xmapsto{\sigma_2} \mu_2 \cdots \mu_{n-1} \xmapsto{\sigma_n} \mu_n = \mu$, and for each i, a system of linear inequalities $ILP_i(x, y, z)$ can be set up (based upon sub-PN $\mathcal{P}_i$, where x, y, z are vector variables) in such a way that $ILP_i(\mu_{i-1}, \mu_i, z)$ has a solution for z iff there exists a σ_i in T_i^* such that $\mu_{i-1} \xmapsto{\sigma_i} \mu_i$ and $z = \#_{\sigma_i}$. See Figure 13.

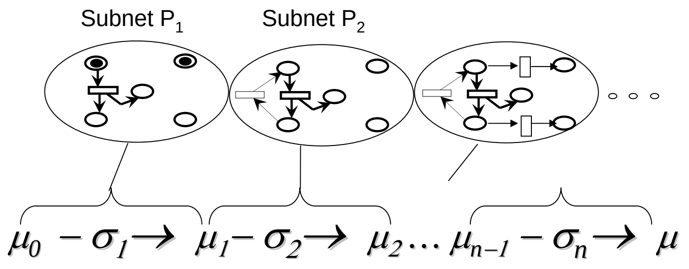

Subnet P_1
Subnet P_2
$\mu_0 - \sigma_1 \rightarrow \mu_1 - \sigma_2 \rightarrow \mu_2 \ldots \mu_{n-1} - \sigma_n \rightarrow \mu$

Fig. 13. Decompositional approach.

Each of the subclasses of PNs depicted in Figure 12 enjoys the merit of having a 'nice' decomposition. In what follows, we elaborate on two notable examples, namely, the classes of *normal* and *sinkless* PNs. For other subclsses of PNs, the reader is referred to [50, 63, 64].

Let $\mu_0 \xmapsto{\sigma_1} \mu_1 \xmapsto{\sigma_2} \mu_2 \cdots \mu_{n-1} \xmapsto{\sigma_n} \mu_n = \mu$ be a computation in a normal (or sinkless) PN $\mathcal{P}$ reaching μ. The rearrangement $\sigma_1\sigma_2\cdots\sigma_n$ of σ is such that if $\sigma = t_1\sigma'_1 t_2\sigma'_2 \cdots t_n\sigma'_n$ where $t_1, t_2, ..., t_n$ mark the first occurrences of the respective transitions in σ (i.e., $t_i, 1 < i \leq n$, is not in the prefix $t_1\sigma'_1 \cdots t_{i-1}\sigma'_{i-1}$) then σ_i is a permutation of $t_i\sigma'_i$. (In words, the appearance of a new transition triggers the beginning of a new segment.) Furthermore, by letting

- $T_0 = \emptyset$,
- $\forall 1 \leq i \leq n$, $T_i = T_{i-1} \cup \{t_i\}$, for some $t_i \notin T_{i-1}$ enabled at μ_{i-1}, and
- φ_i is the restriction of φ to $(P \times T_i) \cup (T_i \times P)$,

it can be shown [26] that reachability in sub-PN $\mathcal{P}_i$ can be captured by an instance $ILP_i(\mu_{i-1}, \mu_i, z_i)$. That is, marking μ_i is reachable from μ_{i-1} in sub-PN $\mathcal{P}_i$ iff there is a solution with respect to variable z_i in $ILP_i(\mu_{i-1}, \mu_i, z_i)$. As a result, if μ is reachable, then (1) there must exist a canonical computation reaching μ such that the computation can be decomposed into a sequence of sub-computations, say $\sigma_1, \sigma_2, \cdots, \sigma_n$, each coincides with the respective member in the PN $\mathcal{P}$ decomposition; namely $\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_n$, and (2) checking the reachability of PN $\mathcal{P}$ is equivalent to solving the a collection of systems of linear equalities made up of $ILP_i(\mu_{i-1}, \mu_i, z_i)$, for $1 \leq i \leq n$. See [26] for more details.

Based on the decompositional approach, the reachability problem for each of the subclasses of PNs depicted in Figure 12 is solvable in NP. (In fact, the problem is NP-complete as the NP-hardness lower bound is easy to show.)

In addition to the aforementioned subclasses of PNs, *single-path PNs* are another class whose semilinear reachability sets have been characterized in detail. See [23] for more.

7 Extended Petri nets

As mentioned in our earlier discussion, the power of conventional PNs is strictly weaker than that of Turing machines. Those using PNs to model real-world systems have often found the expressive power of PNs to be too simple and limited. In many real-time applications, it is often desirable to give certain jobs higher priorities over others, so that critical actions can be finished within their time constraints. One way to do so, for example, is to assign each transition of a process a priority which indicates the degree of importance or urgency. As [1] indicates, the conventional PN model is unable to model prioritized systems. From a theoretical viewpoint, the limitation of PNs is precisely due to the lack of abilities to test potentially unbounded places for zero and then act accordingly. With zero-testing capabilities, it is fairly easy to show the 'extended' PNs to be equivalent to *two-counter machines* (which are Turing equivalent).

To remedy the weakness in expressive power, a number of extended PN models have been proposed in the literature. Among them include *colored PNs* [34], *timed PNs* [47, 57], *PNs with inhibitor arcs* [51], *Prioritized PNs* [20], *PNs under the maximal firing strategy* ... and more. Each of the above extensions allows the PN to faithfully simulate *test-for-zero*, rendering the above extended PNs Turing-equivalent. In what follows, we consider the so-called *PNs under the maximal firing strategy*, which are of interest due to their close connection with the model of *P systems* [52], a model abstracting from the way living cells process their chemical compounds in their compartmental structures.

Under the maximal firing strategy, all the fireable rules are applied in a nondeterministic and maximally parallel fashion at any point in time. Now we show how PNs operating under this new semantics are capable of simulating two-counter machines, which are finite automata augmented with two counters on which the following operations can be applied to each counter: (1) add one to a counter, (2) subtract one from a counter, provided that the counter is not zero, and (3) test a counter for zero and then move accordingly. It is well-known that two-counter machines and Turing machines are computationally equivalent.

The simulations of types (1) and (2) operations of a two-counter machine are straightforward. Now we see how (3) can be faithfully simulated by a PN operating under the maximal firing mode. Consider the PN structure shown in Figure 14, in which place C simulates one of the two counters, and the current state of the two-counter machine is recorded by placing a token in the corresponding place, for example, p_1. Consider the following two cases, depending on whether C is empty or not:

1. $(C = 0)$: the computation involves $(p_1 = 1) \xmapsto{t_1} (p_2 = p_3 = 1) \xmapsto{t_3} (p_3 = p_4 = 1) \xmapsto{t_4} (p = 1)$
2. $(C = k > 0)$: the computation involves $(p_1 = 1; C = k) \xmapsto{t_1} (p_2 = p_3 = 1; C = k) \xmapsto{\{t_3,t_2\}} (p_4 = p_5 = 1; C = k - 1) \xmapsto{t_5} (p' = 1; C = k - 1)$

It is then reasonably easy to see that a token is deposited into p provided that the counter is zero to begin with; otherwise, a token is moved into p' while C is being decremented by one.

With the above extended PNs powerful enough to simulate Turing machines, all nontrivial problems for such PNs become undecidable. A natural and interesting question to ask is: are there PNs whose powers lie between conventional PNs and Turing machines? As it turns out, the quest for such 'weaker' extensions has attracted considerable attentions in recent years. Such PN extensions include *reset nets* [9], and *transfer nets* [13]. A reset net is a PN (P, T, φ) equipped with a set F_R $(\subseteq T \times P)$ of *reset arcs*. When a transition t with $(t, p) \in F_R$ is fired, place p is reset to zero. In a transfer net, a set F_T $(\subseteq P \times T \times P)$ of *transfer arcs* is associated with a PN (P, T, φ) such that when t is fired with $(p, t, q) \in F_T$, then the following actions are taken in the given order: (1) removing the enabling tokens, (2) transferring all tokens from p to q, and (3) adding the usual output tokens.

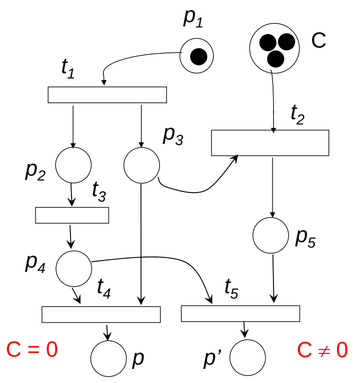


Fig. 14. Simulating test-for-zero of a two-counter machine.

Interestingly and somewhat surprisingly, the boundedness problem is decidable for transfer nets but undecidable for reset nets. The *termination problem* is decidable for both classes, whereas *structural termination* turns out to be undecidable for both. The interested reader is referred to [9, 13] for more about reset and transfer nets. It is of interest to seek additional decidability/complexity results for problems related to such PN extensions as well as characterizing additional PN extensions which are weaker than Turing machines but more powerful than conventional PNs.

References

1. Agerwala, T. and Flynn, M. Comments on capabilities, limitations and 'correctness' of Petri nets, *Proc. of 1st Annual Symposium on Computer Architecture*, pp. 81-86, 1973.
2. Baker, H. Rabin's proof of the undecidability of the reachability set inclusion problem of vector addition systems, Computation Structures Group Memo 79, Project MAC, MIT, July 1973.
3. Borosh, I. and Treybig L. Bounds on positive integral solutions of linear Diophantine equations, *Proc. Amer. Math. Soc.* **55**:299-304, 1976.
4. Cherkasova, L., Howell, R. and Rosier, L. Bounded self-stabilizing Petri nets, *Acta Informatica* **32**:189-207, 1995.
5. Clote, P. On the fFinite containment problem for Petri nets, *Theoretical Computer Science*, **43**:99–105, 1986.
6. Davis, M. Hilbert's tenth problem is unsolvable, *American Mathematical Monthly* **80**:233-269, 1973.
7. Desel, J. and Esparza, J. *Free choice Petri nets*, volume 40 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1995.
8. Dijkstra, E. Self-stabilizing systems in spite of distributed control, *C. ACM* **17**:643-644, 1974.
9. Dufourd, C., Jancar, P. and Schnoebelen, P. Boundedness of reset P/T nets, *Int'l Colloquium on Automata, Languages and Programming*, LNCS 1644, pp. 301-310, Springer-Verlag, 1999.

10. Esparza, J. Petri nets, commutative context-free grammars and basic parallel processes, *Fundamenta Informaticae* **30**:24-41, 1997.
11. Esparza, J. Decidability of model checking for infinite-state concurrent systems, *Acta Informatica* **34**:85-107, 1997.
12. Esparza, J. Decidability and complexity of Petri net problems V an introduction, *Lectures on Petri Nets I: Basic Models. Advances in Petri Nets*, LNCS 1491, In G. Rozenberg and W. Reisig, editors, pp. 374V428, 1998.
13. Finkel, A. and Schnoebelen, Ph. Fundamental structures in well-structured infinite transition systems, *Proc. of Latin American Theoretical INformatics (LATIN)*, LNCS 1380, pp. 102-118, 1998.
14. German, S. and Sistla, A. Reasoning about systems with many processes, *J. ACM* **39**(3):675 - 735, 1992.
15. Ginsburg, S. The Mathematical Theory of Context-Free Languages, New York: McGraw-Hill, 1966.
16. Grabowski, J. The decidability of persistence for vector addition systems, *Inf. Process. Lett.* **11**(1):20-23,1980.
17. Habermehl, P. On the complexity of the linear time mu-calculus for Petri nets, *18th International Conference on Application and Theory of Petri Nets*, LNCS 1248, pp. 102–116, 1997. .
18. Hack, M. The recursive equivalence of the reachability problem and the liveness problem for Petri nets and vector addition systems, *FOCS*, pp. 156-164, 1974.
19. Hack, M. The equality problem for vector addition systems is undecidable, In *C.S.C. Memo 121, Project MAC, MIT*, 1975.
20. Hack, M. Decidability questions for Petri nets, PhD dissertation, Dept. of Electrical Engineering, MIT, 1975.
21. Hopcroft, J. and Pansiot, J. On the reachability problem for 5-dimensional vector addition systems, *Theoretical Computer Science*, 8(2):135–159, 1979.
22. Howell, R., Huynh, D., Rosier, L. and Yen, H. Some complexity bounds for problems concerning finite and 2-dimensional vector addition systems with states, *Theoretical Computer Science*, **46**(2-3):107-140, 1986.
23. Howell, R., Jancar, P., and Rosier, L. Completeness results for single path Petri nets, *Information and Computation* **106**:253-265, 1993.
24. Howell, R. and Rosier, L. Problems concerning fairness and temporal logic for conflict-free Petri nets, *Theoretical Computer Science* **64**(3):305-329, 1989.
25. Howell, R., Rosier, L. and Yen, H. A taxonomy of fairness and temporal logic problems for Petri nets, *Theoretical Computer Science*, **82**:341-372, 1991.
26. Howell, R., Rosier, L. and Yen, H. Normal and sinkless Petri nets, *Journal of Computer and System Sciences* **46**:1-26, 1993.
27. Huttel, H. Undecidable equivalences for basic parallel processes, *TACS 94*, LNCS 789, pp. 454-464, 1994.
28. Huynh, D. Commutative grammars: the complexity of uniform word problems, *Information and Control* **57**:21-39, 1983.
29. Ibarra, O., Reversal-bounded multicounter machines and their decision problems, *JACM* **25**(1):116 - 133, 1978.
30. Ichikawa, A. and Hiraishi, K. Analysis and control of discrete event systems represented by Petri nets, LNCIS 103, pp. 115-134, 1987.
31. Jancar, P. Decidability of a temporal logic problem for Petri nets, *Theoretical Computer Science* **74**:71-93, 1990.
32. Jancar, P. Non-primitive recursive complexity and undecidability for Petri net equivalences, *Theoretical Computer Science* **256**:23-30, 2001.

33. Jantzen, M. Complexity of place/transition nets, *Advances in Petri nets* **86**, LNCS 254, 413-435, 1986.
34. Jensen, K. Coloured Petri Nets and the Invariant-Method, *Theor. Comp. Science* **14**, 317-336. 1981.
35. Jones, N., Landweber, L. and Lien, Y. Complexity of some problems in Petri nets, *Theorettical Computer Science* **4**:277-299, 1977.
36. Karp, R. and Miller, R. Parallel program schemata, *Journal of Computer and System Sciences* **3**:167–195, 1969.
37. Keller, R. Vector replacement systems: a formalism for modelling asynchronous systems, Tech. Rept. 117, Computer Science Lab., Princeton Univ. 1972.
38. Kosaraju, R. Decidability of reachability in vector addition systems, *Proc. the 14th Annual ACM Symposium on Theory of Computing*, pp. 267-280, 1982.
39. Landweber, L. and Robertson, E. Properties of conflict-free and persistent Petri nets, *JACM* **25**(3):352-364, 1978.
40. Lipton, R. *The reachability problem requires exponential space*, Technical Report 62, Yale University, Dept. of CS., Jan. 1976.
41. Mayr, E. Persistence of vector replacement systems is decidable, *Acta Informattica* **15**:309-318, 1981.
42. Mayr, E. An algorithm for the general Petri net reachability problem, *STOC*, pp. 238-246, 1981.
43. Mayr, E. An algorithm for the general Petri net reachability problem, *SIAM J. Comput.* **13**:441-460, 1984.
44. Mayr, R., Decidability and complexity of model checking problems for infinite-state systems, PhD thesis, Computer Science Dept., TU-Munich, Germany, April 1998.
45. Mayr, E. and Meyer, A. The complexity of the finite containment problem for Petri nets, *J. ACM*, **28**(3):561-576, 1981.
46. McAloon, K. Petri nets and large finite sets, *Theoretical Computer Science* **32**:173V183, 1984.
47. Merlin, P. A study of the recoverability of computing systems, PhD thesis, Dept. of Information and COmputer Science, Univ. of California at Irvine, 1974.
48. Muller, H. Decidability of reachability in persistent vector replacement systems, *9th Symp. on Math. Found. of Computer Science*, LNCS 88, pp. 426-438. 1980.
49. Murata, T. Petri nets: properties, analysis and applications, *Proc. of the IEEE*, 77(4):541–580, 1989.
50. Olsén, H. Automatic verification of Petri nets in a CLP framework, Ph.D. Thesis, Dept. of Computer and Information Science, IDA, Linköping Univ., 1997.
51. Patil, S. Coordination of asynchronous events, PhD thesis, Dept. of Elec. Eng., MIT, Cambridge, Mass., May. 1970.
52. Paun, Gh. Computing with membranes, *Journal of Computer and System Sciences*, **61**(1):108–143, 2000.
53. Peterson, J. *Petri Net Theory and the Modeling of Systems*, Prentice Hall, Englewood Cliffs, NJ, 1981.
54. Petri, C. Kommunikation mit Automaten, Dissertation, Rheinisch-Westfalisches Institut fur. Intrumentelle Mathematik an der Universitat Bonn, Bonn. 1962.
55. Presburger, M. Uber die Vollstandigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt, in Comptes Rendus du I congres de Mathematiciens des Pays Slaves, Warszawa, pp. 92-101, 1929.
56. Rackoff, C. The covering and boundedness problems for vector addition systems, *Theoretical Computer Science* **6**:223-231, 1978.

57. Ramchandani, C. Analysis of asynchronous concurrent systems by timed Petri nets, PhD thesis, MIT, Boston, 1974
58. Reisig, W., *Petri Nets: An Introduction*, Springer-Verlag New York, Inc., New York, NY, 1985.
59. Rosier, L. and Yen, H. A multiparameter analysis of the boundedness problem for vector addition systems, *Journal of Computer and System Sciences* **32**(1):105-135, 1986.
60. Sacerdote, G. and Tenney, R. The decidability of the reachability problem for vector addition systems, *STOC*, pp. 61-76. 1977.
61. van Leeuwen, J. A partial solution to the reachability problem for vector addition systems, *STOC*, pp. 303–309, 1974.
62. Yamasaki, H. Normal Petri nets, *Theoretical Computer Science* **31**:307-315, 1984.
63. Yen, H. On the regularity of Petri net languages, *Information and Computation* **124**(2):168-181, 1996.
64. Yen, H. On reachability equivalence for BPP-nets, *Theoretical Computer Science* **179**:301-317, 1997.