Structural Traps of Nested Petri nets

Leonid W. Dworzanski

Structural Traps of Nested Petri nets

L. W. Dworzanski[a,*]

[a]National Research University Higher School of Economics, Myasnitskaya ul. 20, Moscow, Russia 101000

Abstract

The nested Petri nets (NP-nets) are a nets-within-nets formalism convenient for modelling systems that consist of distributed mobile agents with individual behaviour. The expressiveness of the NP-nets formalism is greater than that of classical place/transition nets; and, the formalism allows to model open multiagent systems with agents being introduced, eliminated, and cloned. Several verification methods based on structural analysis and model checking techniques were developed for the formalism. But one of the basic structural methods for Petri nets analysis - traps - was not studied for NP-nets yet. In this paper we introduce NP-nets traps, study how to find them, and illustrate the analysis technique with a system of mobile agents.

Keywords: Petri nets, nets-within-nets formalisms, traps, behaviour analysis

1. Introduction

The ubiquitous propagation of mobile computational devices into the human environment stimulated the rapid spread of distributed systems with mobile agents-components. Not only software components become more distributed and autonomous (web services, mobile agents, cloud applications, ubiquitous computing), but computational devices become more powerful and more autonomous due to intensive development of computing, data storing, and energy accumulating technologies. From safety critical to everyday "internet-of-things" systems — wireless sensor networks, system for coordinating search and rescue operations, corporate and social networks of personal computational devices, automated urban metro subway systems [1] — consist of mobile software and hardware agents moving in informational or physical spaces. The correctness of such systems is becoming more and more crucial to safety and quality of life of individuals. To model and rigorously check the safety of these systems, formal methods techniques should be utilized.

Petri nets evolution resembles the evolution of software systems. From flat unstructured nets, Petri nets have evolved into high-level nets with hierarchical structure and tokens attributed with complex data and/or individual behaviour. "Nets-within-nets" is a modern approach based on the object-oriented paradigm that introduces individual behaviour to tokens by assigning marked Petri nets to them [2]. The application of the approach for modelling active objects, mobility and dynamics in distributed systems is extensively studied [3–5]. Nested Petri nets (NP-nets)

*Corresponding author

Email address: leo@mathtech.ru (L. W. Dworzanski)

[6] are an extension of high-level Petri nets according to the nets-within-nets approach. NP-nets formalism combines value semantics with dynamical hierarchical structure.

An NP-net consists of a high-level system net that coordinates a number of net tokens. Each token has its own behaviour determined by its internal marked Petri net. The levels in an NP-net are synchronized via synchronized transitions (simultaneous firing of transitions in adjacent levels of the net). Because of a loosely-coupled multilevel structure, NP-nets can be used for effective modelling of adaptive distributed systems [7], systems of mobile robots [8], sensor networks of mobile agents [9], innovative space systems architectures [10].

The analysis methods for NP-nets are under active development. In the work [11] the approach to checking properties of NP-nets by translating them into coloured Petri nets was developed. The practical value of the translation is determined by the comprehensive tool support for analysis of coloured Petri nets. In [12] a verification method based on translating recursive NP-nets into PROMELA language and applying SPIN model checker is provided. The compositional approach to inferring liveness and boundedness of NP-nets from liveness and boundedness of NP-nets separate components was introduced in [13] and received software support in [14].

Two basic structural methods for Petri nets analysis are (co-)invariants and (co-)traps. Structural place invariants method for NP-nets was suggested in [15]. However, traps, while defined for many Petri net formalisms, are not yet studied for nets-within-nets formalisms. Traps play crucial role in reachability and liveness analysis of TCC-, EFC-, EAC- and other (sub)classes of Petri nets. In [16], it was shown that liveness of TCC-, EFC- depends on the initial marking of a Petri net traps.

Section 2 contains preliminaries and NP-nets definition. Section 3 introduces traps definition for NP-nets. Section 4 provides an algorithmic procedure to find such traps by reducing the search to satisfiability problem.

2. Preliminaries

By $\mathbb{N}$ we denote the sets of non-negative natural numbers. For a set S, a *bag (multiset) m* over S is a mapping$m : S \to \mathbb{N}$. The set of all bags over S is denoted by $\mathbb{N}^S$. We denote addition and subtraction of two bags by $+$ and $-$, the number of all elements in m taking into account the multiplicity by $\|m\|$, and comparisons of bags by $=, <, >, \leq, \geq$. that are defined as $m_1 \; R \; m_2 \equiv \forall s \in S : m_1(s) \; R \; m_2(s)$ where R is one of $=, <, >, \leq, \geq$. We overload the set notation writing $\emptyset$ for the empty bag and $\in$ for the element inclusion.

Petri nets is a well-known formalism for concurrent systems modelling. In this section, we give the definition of coloured Petri nets with labelled transitions [17] (CP-nets) parameterized with a value universe U. A coloured function for places is defined using the notion of types, and a coloured function for arcs is defined using expressions over the simple additive language $Expr$. Each place is mapped to a type, which is a subset of U. We assume a language $Expr$ for arcs expressions over a set Var of variables and a set Con of constants with some fixed interpretation $\mathcal{I}$, such that for any type-consistent evaluation $\nu : Var \to U$ the value $\mathcal{I}(e, \nu) \in \mathbb{N}^U$ of an expression $e \in Expr$ is defined. We also assume a set Lab of labels for transitions such that $\tau \notin Lab$. The label τ is the special 'silent' label, while labels from Lab mark externally observable firings. The τ labels are usually omitted on transitions.

Definition 1 (Coloured Petri net). A *coloured net* over the universe U is a 6-tuple $\langle P, T, F, \upsilon, \gamma, \Lambda \rangle$, where

- P and T are disjoint finite sets of *places*, respectively, *transitions*;
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs*;
- $\upsilon : P \to 2^U$ is a *place typing* function mapping P to the subsets of U;
- $\gamma : F \to Expr$ is an *arc labelling* function;
- $\Lambda : T \to Lab \cup \{\tau\}$ is a *transition labelling* function.

For an element $x \in P \cup T$, an arc $\langle y, x\rangle$ is called an *input arc*, and an arc $\langle x, y\rangle$ — an *output arc*. A *preset* ${}^\bullet x$ and a *postset* $x^\bullet$ are subsets of $P \cup T$ such that ${}^\bullet x = \{y | \langle y, x\rangle \in F\}$ and $x^\bullet = \{y | \langle x, y\rangle \in F\}$. Given a CP-net $N = \langle P, T, F, \upsilon, \gamma, \Lambda\rangle$ over the universe U, a *marking* in N is a function $m : P \to \mathbb{N}^U$ such that $m(p)$ has $\upsilon(p)$ as a support set. By $\mathfrak{M}_N$, we denote all valid markings of N. A pair $\langle N, m\rangle$ of a CP-net and a marking is called a marked net.

Let $N = \langle P, T, F, \upsilon, \gamma, \Lambda\rangle$ be a CP-net. A transition $t \in T$ is *enabled* in a marking m iff $\exists \nu \forall p \in P : \langle p, t\rangle \in F \Rightarrow m(p) \geq \mathcal{I}(\gamma\langle p, t\rangle, \nu)$. Here $\nu : Var \to U$ is a variable evaluation called a *binding*. An enabled transition t can *fire* yielding a new marking $m'(p) = m(p) - \mathcal{I}(\gamma\langle p, t\rangle, \nu) + \mathcal{I}(\gamma\langle t, p\rangle, \nu)$ for each $p \in P$ (denoted $m \xrightarrow{t} m'$). The set of all markings reachable from a marking m (via a sequence of firings) is denoted by $\mathfrak{R}(m)$.

As usual, a marked net defines a labelled transition system (LTS) which represents the observable behaviour of the net; the transition system states correspond to net markings, its transitions correspond to firings, and such transitions are labelled with corresponding synchronization labels. Also, note that PT-nets are a particular case of CP-nets with a singleton universe.

Let $\langle N, m_0\rangle$ be a marked net with transitions labelled by labels from $Lab \cup \{\tau\}$. By abuse of notation we write $m \xrightarrow{\lambda} m'$, if firing of a transition t, labelled with $\lambda \in Lab \cup \{\tau\}$, transfers a marking m to a marking m'. For $\lambda \neq \tau$, we write $m \overset{\lambda}{\Rightarrow} m'$ if there is a sequence of firings $m = m_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} m_k \xrightarrow{\lambda} m^1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} m^n = m'$, where $k, n \geq 1$. We write $m \overset{\tau}{\Rightarrow} m'$ if there is a sequence of firings $m = m_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} m_k = m'$, where $k \geq 1$.

Definition 2. Let $\left\langle N_1, m_0^1\right\rangle$ and $\left\langle N_2, m_0^2\right\rangle$ be two marked Petri nets with transitions labelled with alphabet $Lab \cup \{\tau\}$. Net N_1 m-simulates N_2 (denoted $N_1 \sqsubseteq_m N_2$) iff there is a relation $R \subseteq \mathcal{R}(N_1) \times \mathcal{R}(N_2)$ such that

1. $\langle m_0^1, m_0^2\rangle \in R$;
2. for any $\langle m_1, m_2\rangle \in R$, if $m_1 \xrightarrow{\lambda} m_1'$ in N_1, then
   (a) there exists m_2' such that $m_2 \overset{\lambda}{\Rightarrow} m_2'$ in N_2 and $\langle m_1', m_2'\rangle \in R$
   (b) for each m_2' such that $m_2 \overset{\lambda}{\Rightarrow} m_2'$ in N_2 it should be $\langle m_1', m_2'\rangle \in R$

Definition 3. Let $\left\langle N_1, m_0^1\right\rangle$ and $\left\langle N_2, m_0^2\right\rangle$ be two marked Petri nets with transitions labelled with alphabet $Lab \cup \{\tau\}$. Nets N_1 and N_2 are m-bisimilar (denoted $N_1 \approx_m N_2$) iff there is a relation $R \subseteq \mathcal{R}(N_1) \times \mathcal{R}(N_2)$ such that

1. $\langle m_0^1, m_0^2\rangle \in R$;

2. N_1 m-simulates N_2 with R, and, simultaneously, N_2 m-simulates N_1 with R^{-1}.

Note that the m-bisimilarity relation is not reflexive. The intrinsic reason for this is that the composition of components even with the same behaviour may be not live.

Nested Petri nets (*NP-nets*) are coloured Petri nets over a special universe [18]. This universe consists of elements of some finite set S (called atomic tokens) and marked Petri nets (called net tokens). We consider here only two-level NP-nets, where net tokens are classical place-transition nets.

Let S be a finite set of atomic objects. For a CP-net N by $\mathcal{M}(N, S)$ we denote the set of all marked nets, obtained from N by adding markings over the universe S. Let then $N_1, \ldots, N_k$ be CP-nets over the universe S. Define a universe $\mathcal{U}(N_1, \ldots, N_k) = S \cup \mathcal{M}(N_1, S) \cup \cdots \cup \mathcal{M}(N_k, S)$ with types $S, \mathcal{M}(N_1, S), \ldots, \mathcal{M}(N_k, S)$. We denote $\Omega(N_1, \ldots, N_k) = \{S, \mathcal{M}(N_1, S), \ldots, \mathcal{M}(N_k, S)\}$. By abuse of notation, we say that a place p with a type $\mathcal{M}(N, S)$ is typed by N.

Definition 4 (Nested Petri net). Let *Lab* be a set of transition labels and let $N_1, \ldots, N_k$ be CP-nets over the universe S, where all transitions are labelled with labels from $Lab \cup \{\tau\}$.

An NP-net is a tuple $NP = \langle N_1, \ldots, N_k, SN \rangle$, where $N_1, \ldots, N_k$ are called element nets, and SN is called a system net. A system net $SN = \langle P_{SN}, T_{SN}, F_{SN}, \upsilon, \gamma, \Lambda \rangle$ is a CP-net over the universe $\mathcal{U} = \mathcal{U}(N_1, \ldots, N_k)$, where places are typed by elements of $\Omega = \Omega(N_1, \ldots, N_k)$, transition labels are from $Lab \cup \{\tau\}$, and an arc expression language *Expr* is defined as follows.

Let *Con* be a set of constants interpreted over $\mathcal{U}$, and *Var* – a set of variables, typed with Ω-types. Then an expression in *Expr* is a multiset of elements over $Con \cup Var$ of the same type with two additional restrictions for each transition $t \in T_{SN}$:

1. constants or multiple instances of the same variable are not allowed in input arc expressions of t;
2. each variable in an output arc expression for t occurs in one of the input arc expressions of t.

Note that removing the first restriction on system net arc expressions makes NP-nets Turing-complete [6], since without this restriction there would be a possibility to check, whether inner markings of two tokens in a current marking are equal, and hence to make a zero-test. The second restriction excludes infinite branching in a transition system representing the behaviour of an NP-net.

The interpretation of constants from *Con* is extended to the interpretation $\mathcal{I}$ of expressions under a given binding of variables in the standard way.

We call a marked element net a net token, and an element from S — an atomic token. A marking in an NP-net is defined as a marking in its system net. So, a marking $m : P_{SN} \to \mathbb{N}^{\mathcal{U}}$ in an NP-net maps each place in its system net to a multiset of atomic tokens or net tokens of appropriate type.

A behaviour of an NP-net is composed of three kinds of steps (firings). An *element-autonomous step* is the firing of a transition t labelled with τ (τ-transition) in one of the net tokens of the current marking according to the usual firing rule for coloured Petri nets. Formally, let m be a marking in an NP-net NP, $\alpha = \langle N, \mu \rangle \in m(p)$ — a net token residing in the place

$p \in P_{SN}$ in m. Let also t_α be enabled in α and $\mu \xrightarrow{t_\alpha} \mu'$ in α. Then the element-autonomous step $s = \{t_\alpha\}$ is enabled in m, and the result of s-firing is the new marking m' such that for all $p' \in P_{SN} \setminus p$: $m'(p') = m(p')$, and $m'(p) = m(p) - \alpha + \langle N, \mu' \rangle$. Note that such a step changes only the inner marking in one of the net tokens.

A *system-autonomous step* is the firing of a transition $t \in T_{SN}$, labelled with τ, in the system net according to the firing rule for coloured Petri nets, as if net tokens were just coloured tokens without an inner marking. Formally, the system-autonomous step $s = \{t\}$ is *enabled* in a marking m iff there exists a binding $\nu : Var \to \mathcal{U}$ such that $\forall p \in P_{SN} : \langle p, t \rangle \in F_{SN} \Rightarrow m(p) \geq \mathcal{I}(\gamma \langle p, t \rangle, \nu)$. The result of s-firing is the new marking $m'(p) = m(p) - \mathcal{I}(\gamma \langle p, t \rangle, \nu) + \mathcal{I}(\gamma \langle t, p \rangle, \nu)$ for each $p \in P_{SN}$ (denoted $m \xrightarrow{s} m'$). An autonomous step in a system net can move, copy, generate, or eliminate tokens involved in the step but does not change their inner markings.

A *(vertical) synchronization step* is the simultaneous firing of a transition $t \in T_{SN}$ labelled with some $\lambda \in Lab$ (λ-transition) in the system net together with firings of transitions $t_1, \ldots, t_q$ ($q \geq 1$), also labelled with λ, in all net tokens involved in (i.e. consumed by) this system net transition firing.

Formally, let m be a marking in an NP-net NP, a transition $t \in T_{SN}$ be labelled with λ and enabled in m via binding ν as a system-autonomous step. We say that a net token α is involved in t-firing via binding ν iff $\alpha \in \mathcal{I}(\gamma \langle p, t \rangle, \nu)$ for some $p \in {}^\bullet t$. Let then $\alpha_1 = \langle N_{i1}, \mu_1 \rangle, \ldots, \alpha_q = \langle N_{iq}, \mu_q \rangle$ be all net tokens involved in the firing of t via binding ν, and, for each $1 \leq j \leq q$, there is a transition t_j, labelled with λ in N_{ij}, such that t_j is enabled in μ_j, and $\mu_j \xrightarrow{t_j} \mu'_j$ in N_{ij}. Then the synchronization step $s = \{t, t_1, \ldots, t_q\}$ is enabled in m for NP, and the result of s-firing is the new marking m' defined as follows. For each $p \in P_{SN}$: $m'(p) = m(p) - \mathcal{I}(\gamma \langle p, t \rangle, \nu) + \mathcal{I}(\gamma(t, p), \nu')$, where for a variable x: $\nu(x) = \langle N, \mu \rangle$ implies $\nu'(x) = \langle N, \mu' \rangle$.

Figure 1 gives an example of a synchronization step. The left part of the picture shows a marked fragment of a system net. The transition t has two input places p_1 and p_2, and two output places p_3 and p_4. In the current marking, the place p_1 contains three net tokens; two of them, α_1 and α_3, are explicitly depicted. The place p_2 contains two net tokens; the structure and the marking of one of them are shown in the picture. Only the synchronization step is allowed here since all transitions are labelled with the synchronization label λ. A possible binding of variables x, y, z in the input arc expressions is $x = \alpha_1, y = \alpha_2$ and $z = \alpha_3$. Then the transitions t in the system net, net token transitions $\alpha_1 : t_1$, $\alpha_3 : t_1$, and $\alpha_2 : t_2$ fire simultaneously. The resulting marking m' is shown on the right side of the picture. According to the output arc expressions, after t-firing, two copies of α_1 appear in p_3, the net token α_3 disappears, α_2 with a new marking is transported to the place p_4, and a new net token α_c appears in p_4 being a value of the net constant c.

A transition labelled with $\lambda \in Lab$ in a system net consumes net tokens with enabled transitions labelled with λ. To exclude obviously dead transitions, we add to our definition of NP-nets the following syntactical restriction: for each system net transition t labelled with $\lambda \neq \tau$, and for each $p \in {}^\bullet t$, p is typed by an element net with at least one transition labelled with λ.

We denote by $E : \{p_1, p_2, \ldots\}$ ($E : \{t_1, t_2, \ldots\}$) a set of places (transitions) $p_1, p_2, \ldots$ of an element net E

Thus, a step is a set of transitions (a one-element set in the case of an autonomous step). We write $m \xrightarrow{s} m'$ for a step s converting the marking m into the marking m'. By $Steps(NP)$ we denote the set of all (potential) steps in NP.

A run in an NP-net NP is a sequence $\rho = m_0 \xrightarrow{s_1} m_1 \xrightarrow{s_2} \ldots$, where $m_0, m_1, \ldots$ are markings,

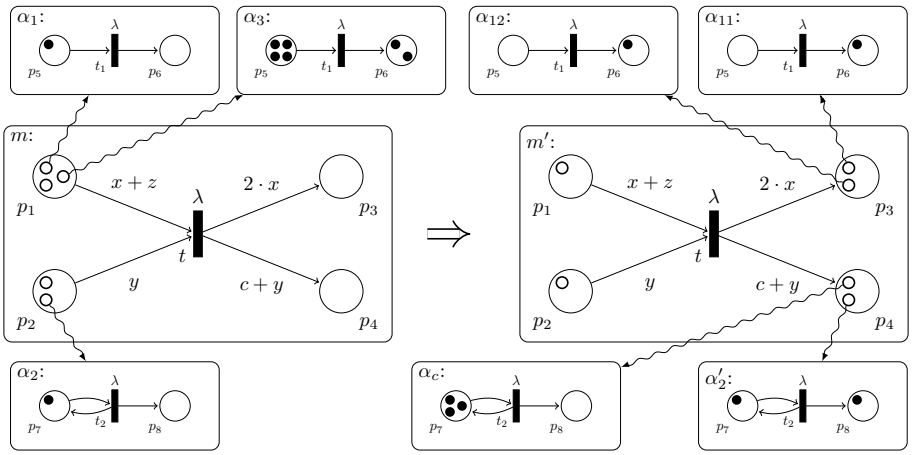

Figure 1: A firing of a synchronization step in the NP-net

m_0 is an initial marking, and $s_1, s_2, \ldots$ are steps in *NP*. By $\sigma|_T = s'_1, \ldots s'_n$, we denote the projection of a run ρ on a set of transitions T such that $s'_i = t$ if $t \in s_i$, and $s'_i = \tau$, otherwise. For a sequence of steps $\sigma = s_1, \ldots s_n$, we write $m \xrightarrow{\sigma} m'$ and say that m' is reachable from m, if $m = m_0 \xrightarrow{s_1} m_1 \ldots \xrightarrow{s_n} m_n = m'$. By $\mathfrak{R}(NP, m)$ we denote the set of all markings reachable from m in *NP*, and by abuse of notations we write $\mathfrak{R}(NP)$ for the set of all markings reachable in *NP* from its initial marking.

When a net token has an enabled transition with a synchronization label λ, i.e. it can participate in a synchronization step that includes transitions labelled with λ, we hereinafter say that the net token supports (or provides) a λ-synchronization. For brevity, a transition labelled with synchronization label λ is called λ-transition. In opposite direction, we say that the system net supports (or provides) a λ-synchronization for a net token α if the system net can fire a λ-transition involving α. Runs consisting of only autonomous steps are called τ-runs.

A transition t in an NP-net *NP* is live if, for every reachable marking m, there is a run $\rho = m \xrightarrow{s_1} m_1 \xrightarrow{s_2} \ldots m_k \xrightarrow{s_k}$ such that $t \in s_k$ where $k \geq 1$. The *sub-liveness* problem is the problem of deciding whether a given transition t is live in a given net with a marking m [19].

In what follows, when we speak of separate components of a NP-net, we consider a system net and element nets as usual coloured Petri disregarding synchronization. For a system net *SN* in a NP-net *NP*, by $\widehat{SN}$ we denote the system net component, and, for an element net N, by $\widehat{N}$ we denote the element net component. Net tokens in a system net component marking are treated as plain tokens without inner markings, and synchronization labels are not taken into account during execution. Operational semantics for $\widehat{SN}$ and $\widehat{N}$ is defined according to the firing rules for coloured Petri nets.

Let then m be a marking in a NP-net *NP*, i.e. a mapping $m : P_{SN} \to \mathbb{N}^{\mathcal{U}}$, where $\mathcal{U} = S \cup \mathcal{M}(N_1, S) \cup \cdots \cup \mathcal{M}(N_k, S)$. By $\widehat{m}$, we denote a marking in $\widehat{SN}$ s.t. for each $p \in P_{SN}$ we have $\widehat{m}(p) = m(p)$, if p is typed by S, and, for p typed with N_i, we have $\widehat{m}(p) = c_p \cdot \langle N_i, \mathbb{0} \rangle$, where $\langle N_i, \mathbb{0} \rangle$ is N_i with empty marking, and c_p is the number of tokens in $m(p)$.

Note that net tokens of the same type (i.e., with the same net structure) are not distinguished

in a system net autonomous firing. This follows from the first input arc expressions restriction for NP-nets, which eliminates comparing inner markings of net tokens. Moreover, since all tokens in a system net place are of the same type, enabledness of an autonomous transition in a system net depends only on the numbers of tokens in its input places, and a system net considered as a separate component is actually similar to a PT-net.

For further details on NP-nets see [6, 20]. Note, however, that here we consider a typed variant of NP-nets, where a type is instantiated to each place.

NP-traps and co-traps *) are they dual *) how to deduce liveness w/ (co)traps *) are they compositional *) commoner for NP-nets

3. Nested Petri nets traps definition

Transferring a notion from one structure to another highly depends on what we expect from the result: to be structurally as close to the original definition as possible, to play the same role for theorems similar to thus of the original structure, or to be identified/constructed by similar algorithms (or by algorithms with not much worse complexity). All these expectations can lead to different interpretations of the original notion in a new context. To justify a suggested notion, we show that the new notion is applicable in behavioural analysis of NP-nets and comply to axioms similar to that which are complied by the original notion of traps.

Traps play at least two important roles in Petri net analysis. Firstly, they can be used directly to prove some important properties of a Petri net [21]. Secondly, they play crucial role in liveness analysis of Petri nets [22] and their important subclasses (EFC-, EAC-, TCC-nets) [16, 23].

Structurally, a trap for a Petri net is a set of places S such that ${}^\bullet S \subseteq S^\bullet$, i.e. if any transition consumes some tokens from S by firing then it produces at least a token back to S. The *fundamental property* of a trap — once it is marked, it stays marked forever.

We start by introducing the following notation for different kinds of sets of net places

- $E_i.S \subseteq P_{E_i}$ — place set S of elementary net E_i;
- $p_{SN} : E_i.S \subseteq P_{E_i}$ — place set S of elementary net E_i within system net place p_{SN};
- $\alpha_i.S \subseteq P_{E_i}$ — place set S of net token α_i;
- $p_{SN} : \alpha_i.S \subseteq P_{E_i} \times P_{SN}$ — place set S of net token α_i within system net place p_{SN}.

We call a set of places S of an NP-net component N a *local* trap — whether N is a system net or an element net — if S is a trap for N considered as a separate component, i.e. forgetting synchronization labelling.

3.1. NP-net traps containing system net places

Let us start with traps containing system net places. The system in Figure 2 consists of computational resources — threads of execution — and coordinating agents; the agents can allocate threads and assign jobs to them. The system net *SN*, which is shown on the right part of the figure, models the overall interaction of agents and net tokens. Available threads are located in threads pools p_7 and p_8. Net tokens α_1 and α_2, on the left part of the figure, represent computational agents, that flow through the system; places p_1–p_6 are the possible locations of such net tokens. Computation phases, which require available threads, are modelled with places $SN : \{p_2, p_3, p_5, p_6\}$. Before starting computation, an agent-token α allocates ($SN : t_1$ or

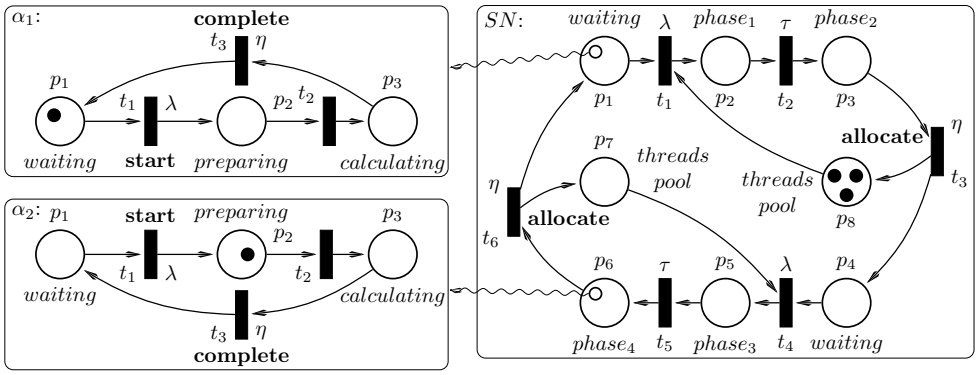

Figure 2: System with two agents

$SN:t_4$) a thread from the corresponding threads pool and, simultaneously, moves ($\alpha:t_1$) into state $\alpha:preparing$. Then, the agent moves ($\alpha:t_2$) into state $\alpha:calculating$. As soon as the agent completes its computation, it gets back ($\alpha:t_3$) to state $\alpha:waiting$ simultaneously with releasing ($SN:t_3$ or $SN:t_6$) the allocated thread.

Places $SN:\{p_2,p_3,p_8\}$ form a trap as any adjacent transition $SN:\{t_2,t_3,t_1\}$, that consumes a token from these places, returns some tokens back to these places as well. Places $SN:\{p_2,p_3,p_8\}$ form a trap due to the same reason. Note that we do not require that the tokens, produced during a step execution, should have the same type as the consumed tokens. For example, during a firing of transition $SN:t_3$, a net token is consumed from trap's place $SN:p_3$, and a black token is produced to trap's place $SN:p_8$. Both traps are local — they correspond to the traps in the system net considered as a separate component.

3.2. NP-net traps containing element net places

Now, let us consider another agent-based queueing system illustrated in Figure 3. It consists of a private storage (DB), agents of two types, and a queue of incoming queries. New queries appear ($SN:t_{10}$) in place $SN:p_7$. The access to the private storage is modelled with places $SN:p_3$ and $SN:p_4$; place $SN:p_2$ models a mutex used to limit the access to DB. A computational agent can handle queries if it has a key to the private DB. Computational agents are located in *agents pool* $SN:p_2$ and new agents may enter ($SN:t_3$). A supervisor agent, located in pool $SN:p_1$, possesses a private key for the private storage and can handle queries; in addition, the supervisor can allocate ($SN:t_6$) a computational agent for handling the queries and share ($\alpha_2:t_4$) the private key with the agent. The supervisor can also release ($SN:t_7$) the computational agent from handling queries. We want to prove a property "there is always at least one agent that can handle queries". The supervisor agent can leave the system; in absence of the supervisor, there is always should be at least one computational agent with a private key that is able to handle queries. System net traps enable us to capture only the presence of net tokens in $SN:p_2$ not their internal marking. To prove that a net token that models computation agent possesses a token in place $\alpha_c:p_0$, we need to extend the notion of traps to the element nets level.

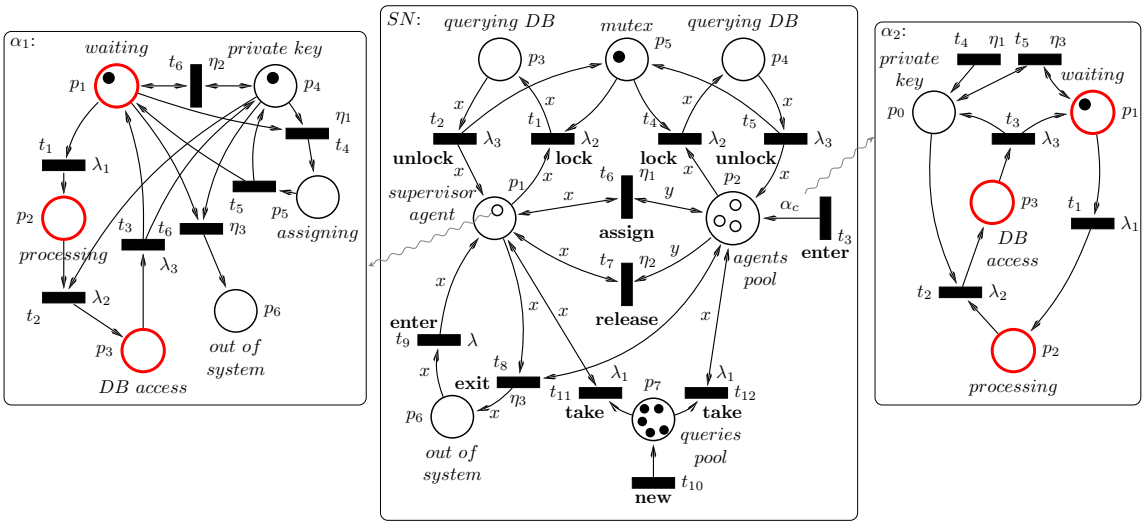

Figure 3: System with two agents

For brevity, we shortcut a set, that consists of the same element net place set S for several different system net places $p_1^{SN}, \ldots, p_n^{SN}$, as $\{p_1^{SN}, \ldots, p_n^{SN}\} : S$, i.e. $\{p_1^{SN}, \ldots, p_n^{SN}\} : S = p_1^{SN} : S \cup \ldots \cup p_n^{SN} : S$

Now consider set $S_1 = \{SN : p_1, p_6\} : \{\alpha_1 : p_4, p_5\} \cup \{SN : p_2\} : \{\alpha_2 : p_0, p_3\}$, which forms an element net trap. Note that transition $\alpha_1 : t_6$, while consuming a token from place $\alpha_1 : p_4$, shall be executed simultaneously with transition $\alpha_2 : t_5$, which guarantees a token in place $\alpha_2 : p_0$. Due to the fundamental property of traps, as trap S is marked in the initial marking, it stays marked in all further states. Consequently, either a supervisor with a marked state p_4 or p_5 or a computational agent with a marked state p_0 or p_3 is in the system. Thus always at least one agent — a supervisor or a computational agent — has access to the private storage. A trap $S_2 = \{SN : p_1, p_6\} : \{\alpha_1 : p_1, p_2, p_3\} \cup \{SN : p_2\} : \{\alpha_2 : p_1, p_2, p_3\}$ guarantees that there always be an active agent — a supervisor or a computational agent — with a token in one of its control states: *waiting*, *processing*, or *DB access*.

A single NP-net step may involve several net tokens located in the same system net place and, consequently, typed with the same element net. Consumption of internal tokens in one net token can be compensated with production of internal tokens in another net token. An element net trap does not distinguish net tokens (element net instances). Therefore, new net tokens, that are produced by system net transitions into element net trap places, are not distinguished from the already existing ones.

3.3. NP-net traps containing net token places

Let us consider a queueing system with concurrent processing depicted in Figure 4. Each agent is represented with a net token in place $SN : p_3$; an agent may have access to one or both processing servers — *server* 1 and *server* 2. Place $SN : p_3$ models a queries queue. When an agent take a query from the queue, it sends requests to both servers and, then, returns a response

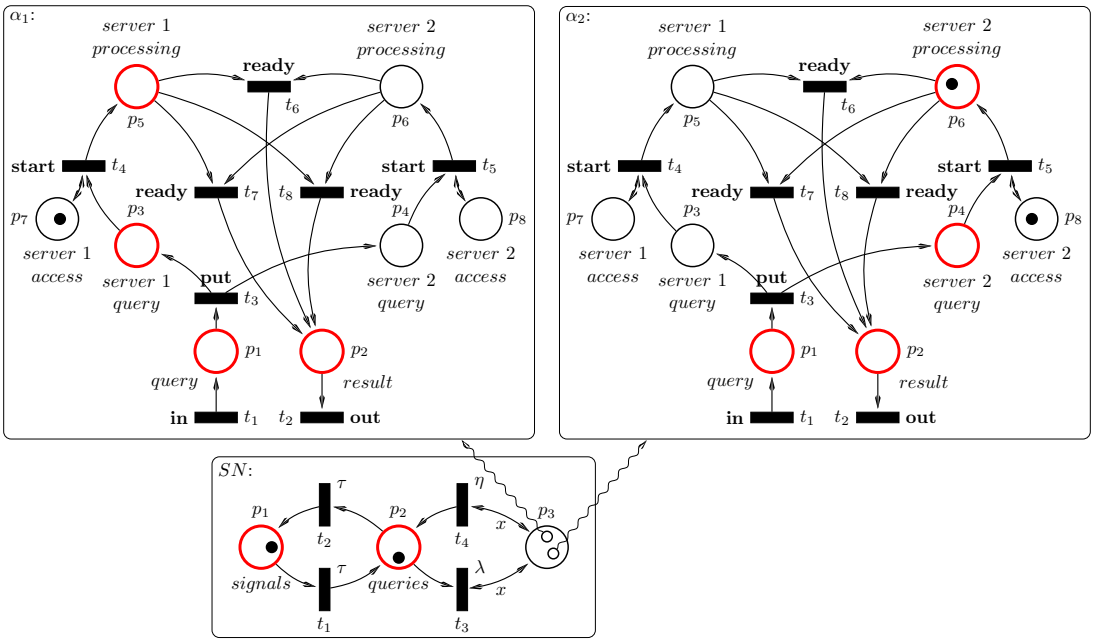

Figure 4: Queueing system with concurrent processing

from the server that responded first. A set $S_3 = \{p_1^{SN}, p_2^{SN}\} \cup \{SN : p_3\} : \{\alpha_1 : p_1, p_3, p_5, p_2\} \cup \{SN : p_3\} : \{\alpha_2 : p_1, p_2, p_4, p_6\}$ forms a trap. For example, when a step $\{SN : t_3, \alpha_1 : t_1\}$ fires, it consumes a token in place $SN : p_2$; this token is compensated with a token produced by $\alpha_1 : t_1$ into place $\alpha_1 : p_1$. The same happens when a token is consumed by net token α_2. As it can be observed, the trap constituents for α_1 and α_2 are different. If we need to have different traps for different net tokens; it is necessary to extend the trap definition to encompass net-token places.

A caveat here is that the existence of such a trap depends on the dynamical properties of the system net; i.e. that no new net tokens are generated/appear in the place $SN : p_3$. As a variable can be bound to such a new token, a firing of transition $SN : t_3$ can consume tokens from $SN : p_2$ but not put tokens into place p_1 of α_1 or α_2, or any other place of the trap. It worth to note that it is possible to consider different possible to handle new net tokens; however, we will not consider them here.

3.4. NP-net traps formal definition

Now, we shall give a formal definition of NP-net traps. As it was in the last example, an NP-net trap may include system net places and net token places simultaneously. Therefore, we put the definition of the general case with all kind of place sets.

Definition 5 (NP-net traps). Let N be an NP-net. Let S be a union of places sets

- system net places — $\{p_1^{SN} \dots p_k^{SN}\}$;

- element net places — $p_1^{SN}:S_1,\ldots,p_l^{SN}:S_l$ such that $\forall i\in\overline{1,l}:\upsilon(p_i^{SN})=E_i\wedge S_i\subseteq P_{E_i}$, where p_i^{SN} is a system net place, and E_i — the type of p_i^{SN},
- net token places — $p_1^{SN}:\alpha_1^1.S_1^1, p_1^{SN}:\alpha_2^1.S_2^1,\ldots,p_n^{SN}:\alpha_{m_1}^n.S_{m_1}^n\cdots,p_n^{SN}:\alpha_1^n.S_1^n, p_n^{SN}:\alpha_2^n.S_2^n\ldots,p_n^{SN}:\alpha_{m_n}^n.S_{m_n}^n$ such that $\forall i\in\overline{1,n}:\upsilon(p_i^{SN})=E_i\wedge\left(\forall j:S_j^i\subseteq P_{E_i}\right)\}$ where α_j^i is a j-th net token within place p_i^{SN}, S_j^i is a set of places of α_j^i, and m_i is a number of net tokens within place p_i^{SN}.

Then S forms a trap iff, for any marking of NP-net, any step s that consumes a token from a place in S, also, puts at least a token back to S, i.e.
$\forall m\in\mathfrak{M}_N:\forall s\in m\xrightarrow{*}:\Big((\exists t\in s:(\exists i\in\overline{1,k}:{}^\bullet t\cap p_i^{SN}\neq\emptyset)\vee(\exists i\in\overline{1,l}:{}^\bullet t\cap S_i\neq\emptyset)\vee$
$(\exists i\in\overline{1,n}:\exists j\in\overline{1,m_i}:{}^\bullet t\cap S_j^i\neq\emptyset))\implies(\exists t\in s:(\exists i\in\overline{1,k}:t^\bullet\cap p_i^{SN}\neq\emptyset)\vee$
$(\exists i\in\overline{1,l}:t^\bullet\cap S_i\neq\emptyset)\vee(\exists i\in\overline{1,n}:\exists j\in\overline{1,m_n}:t^\bullet\cap S_j^i\neq\emptyset))\Big)$ □

A subtle issue to address is net token elimination. When a net token is eliminated, its internal tokens are eliminated as well. What if these internal tokens belonged to the places of a trap? Would the trap property be violated? It is possible to interpret such elimination as that the internal tokens are still in the places of the trap, even if the net token has 'left' the system. However, in this paper, we stick to the following interpretation — if a net token may vanish by a step s execution, then s should produce a token to a trap place of a net token that is still in the system or system net trap place. For example, in the system in Figure 3, even if we remove place $SN:p_6$, i.e. transition $SN:t_8$ would only consume a token from $SN:p_1$ but not produce a token into $SN:p_6$, then the transition $\alpha_2:t_5$ would still guarantee the consistency of the highlighted trap by putting a token into $\alpha_2:p_1$.

4. Finding traps in nested Petri nets

There are several algorithms to find traps of Petri nets [24–26]. In [25], authors suggested a branching algorithm based on places eliminating procedure and set-based decomposition to find a set of minimal siphons for a given Petri net. The algorithm demonstrates performance improvement, comparing to other known algorithms, on randomly generated Petri nets test set. In [26], the authors suggested an iterative

Recall that the fundamental property of traps — if a trap is marked, it stays marked any further. To preserve the fundamental property, each transition that consumes from a place belonging to a trap, have to put at least a token back to the trap. The idea of an algorithm in [16] is based on this restriction. We will following the place encoding in [16]: a property 'p_i *does not belong to the trap*' is encoded with y_{p_i}; also, 'p_i *does belong to the trap*' is encoded with $\overline{y}_{p_i}$. Using such correspondence, it is possible to encode the above restriction for an output place p of a transition t with $\overline{y}_p\implies\bigvee\limits_{p'\in t^\bullet}\overline{y}_{p'}$, or, equivalently, $y_p\vee\left(\bigvee\limits_{p'\in t^\bullet}\overline{y}_{p'}\right)$.

To be able to select system net places and net tokens for which we want to find traps in an NP-net $NP=\langle N_1,\ldots,N_k,SN,m_0\rangle$, we introduce a selection function $Ctx:P_{SN}\to A\cup\{N_1,\ldots,N_k\}$; set A is a set of net tokens in the initial marking m_0. $Ctx(p)=\alpha_i$ denotes that the internal places of net token α_i located in system net place p can be included into a NP trap. If we do not need to distinguish individual net tokens and just interested in element net traps, then we put $\forall p\in P_{SN}:Ctx(p)=\upsilon(p)$. For example, if in Figure 3 there would be only one supervisor agent

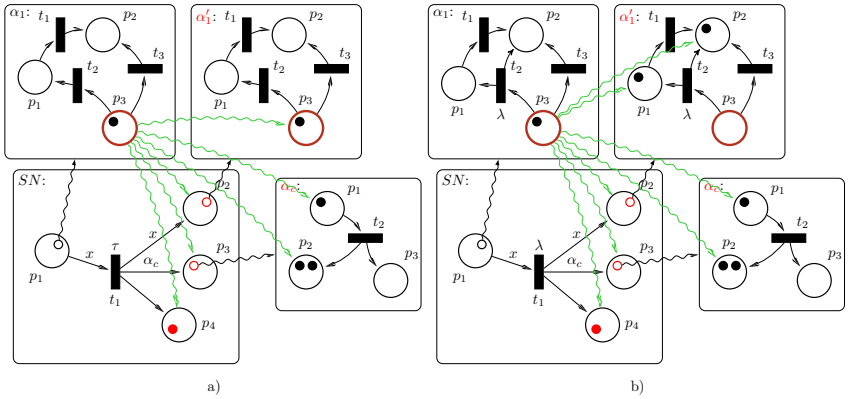

Figure 5: Examples of *a*) a system autonomous and *b*) a synchronization step executions. Green arrows illustrate an implication clause induced by a step.

α_1 that can be located in places $SN\colon p_1, p_3, p_6$, then we may put $\forall p \in \{p_1, p_3, p_6\} : Ctx(p) = \alpha_1$. Let E_2 be the element net of agent net tokens; as there can be many agent tokens in $SN\colon p_2, p_4, p_7$, we put $\forall p \in \{p_2, p_4, p_7\} : Ctx(p) = E_2$. The traps corresponding to such defined function Ctx include element net trap places in $SN\colon p_2, p_4, p_7$, net token trap places in $SN\colon p_1, p_3, p_6$, and system net places.

Now we consider what clauses are induced by element-autonomous steps. Let $NP = \langle N_1, \ldots, N_k, SN, m_0\rangle$ be an NP-net. Recall that an element-autonomous step $\{t\}$ execution consists of one τ-transition t firing in a net token α. If a system net place p_{SN} is typed with $E = \upsilon(p_{SN})$, and E includes a τ-transition t, then an execution of step $\{t\}$ consumes tokens from ${}^\bullet t$ and produces tokens into $t^\bullet$. Therefore, if a place $p \in {}^\bullet t$ is included into a trap T, then one of places $t^\bullet$ must be included into T:

$$\overline{y}_{p_{SN}:E.p} \rightarrow \bigvee_{q \in t^\bullet} \overline{y}_{p_{SN}:E.q} \tag{1}$$

If a trap may include net token trap places of a specific net token α, i.e. $\alpha \in Ctx(p_{SN})$, then we include net token trap places; namely, we add a clause with α as the context of places, i.e.

$$\overline{y}_{p_{SN}:\alpha.p} \rightarrow \bigvee_{q \in t^\bullet} \overline{y}_{p_{SN}:\alpha.q} \tag{2}$$

For example, the transition t_7 of net token α_1 in Figure 4 is an autonomous transition (labelled with τ). If $\alpha_1 \in Ctx(SN\colon p_3)$, then step $\{t_7\}$ induces clause $\overline{y}_{p_{SN}:\alpha.p_2} \rightarrow \overline{y}_{p_{SN}:\alpha.p_5} \vee \overline{y}_{p_{SN}:\alpha.p_6}$.

Now let us consider system autonomous steps. A system autonomous step $\{t_{SN}\}$ consists of a single system net transition t_{SN} labelled with τ and a binding ν of the variables on the input arcs of t_{SN}. When such a step fires, involved net tokens are moved from input places to output places without changing internal markings. If an input place p_{in} of t_{SN} belongs to a trap, then t_{SN} must put a token back to the trap. As t_{SN} produces tokens — net or atomic — to all its output places, one of its output places shall belong to the trap. Consider an example of system autonomous step

execution in Figure 5a). A firing of t_{SN} consumes a net token from p_1 and put tokens to system net places p_2, p_3, and p_4; to secure the fundamental property of traps, we can add any of places p_2, p_3, and p_4 to the trap.

If an outgoing arc $\langle t_{SN}, p_{out}\rangle$ is labelled with net constant α_c, then a firing of t_{SN} produces a net token equal to α_c to the system net place p_{out}. Let the type of net constant α_c be element net E_c, and α_c has a marked place p. If element net place $E_c.p$ within system net place p_{out} belongs to the trap, then firing of t_{SN} produces a net token $\alpha = \alpha_c$ into p_{out} with a token in internal place p, and, thus, marks the trap. Therefore, to compensate the consumption of a token from p_{in}, we can include $p_{out} : E_c.p$ into the trap. In the example in Figure 5a), we can add any of p_2, p_3, and p_4 to the trap, but we can, as well, include element net places $p_3 : E_c.p_1$ or $p_3 : E_c.p_2$ to the trap.

Therefore, a system autonomous step $\{t_{SN}\}$ and its input system net place $p_{SN} \in {}^\bullet t_{SN}$, induce a clause

$$\overline{y}_{p_{in}} \rightarrow \underbrace{\bigvee_{p \in t_{SN}^\bullet} \overline{y}_p}_{(a)} \vee \underbrace{\bigvee_{\substack{\langle t_{SN}, p_{out}\rangle \in F_{SN} \\ \langle\langle P_c, T_c, F_c\rangle, m_0\rangle \in Con(\langle t_{SN}, p_{out}\rangle)}} \left(\bigvee_{\substack{p_c \in P_c \\ m_0(p_c) > 0}} \overline{y}_{p_{out} : E_c.p_c} \right)}_{(b)} \tag{3}$$

where subterm $(3a)$ corresponds to the output system net places of t_{SN}, and subterm $(3b)$ corresponds to all marked places of all net constants on outgoing arcs of t_{SN}. For example, system net place p_1 in Figure 5a) induces a clause $\overline{y}_{p_1} \rightarrow \underbrace{p_2 \vee p_3 \vee p_4}_{(a)} \vee \underbrace{p_3 : E_c.p_1 \vee p_3 : E_c.p_2}_{(b)}$, where part (a) corresponds to output places $SN : p_2, p_3, p_4$, part (b) corresponds to the marked places of α_c, and E_c is the element net of net constant α_c on arc $\langle t_1, p_3\rangle$.

Now, it is rest to handle internal places of net tokens that are moved from one system net place to another by system autonomous step execution. Due to the execution rules, the marking of such places is not changed. However, if such internal element net place $p_{in} : E.p_{nt}$, where $p_{in} \in {}^\bullet t_{SN}$, is a part of a trap, and it contains a token before a step execution, then the token is removed from the trap, and such removal shall be compensated. As in the previous case for input system net places, we can compensate it with output system net places of t_{SN} or marked places of net constants. Another option is to introduce the same internal place p_{nt} as an element net trap place for an output system net place p_{out} into which the consumed net token α is moved, i.e. there is an arc $\langle t_{SN}, p_{out}\rangle$ labelled with a variable x such that there is an arc $\langle p_{in}, t_{SN}\rangle$ labelled with x. Thus, when a net token moves from p_{in} to p_{out}, its internal tokens in $p_{in} : E.p_{nt}$ move to $p_{out} : E.p_{nt}$, which we add as a part of the trap. Therefore, a system autonomous step $\{t_{SN}\}$ and an internal place $p_{in} : E.p_{nt}$ of net token α located in an input system net place p_{in} of t_{SN}, induce a clause

$$\begin{aligned} \overline{y}_{p_{in}:E.p_{nt}} \rightarrow \underbrace{\bigvee_{p \in t_{SN}^\bullet} \overline{y}_p}_{(a)} \vee \underbrace{\bigvee_{\substack{\langle t_{SN}, p_{out}\rangle \in F_{SN} \\ \langle\langle P_c, T_c, F_c\rangle, m_0\rangle \in Con(\langle t_{SN}, p_{out}\rangle)}} \left(\bigvee_{\substack{p_c \in P_c \\ m_0(p_c) > 0}} \overline{y}_{p_{out} : E_c.p_c} \right)}_{(b)} \vee \\ \underbrace{\bigvee_{\substack{\langle t_{SN}, p_{out}\rangle \in F_{SN} \\ \exists x \in Var(\langle t_{SN}, p_{out}\rangle) : \nu(x) = \alpha}} \overline{y}_{p_{out}:E.p_{nt}}}_{(c)} \end{aligned} \tag{4}$$

where subterm (4*a*) corresponds to the output system net places of t_{SN}; subterm (4*b*) corresponds to all marked places of all net constants on outgoing arcs of t_{SN}; and, finally, subterm (4*c*) corresponds to internal place p_{nt} within output system net places where the net token can be moved from p_{in}. For example, place $p_1 : E.p_3$ and transition $SN : t_1$ in Figure 5a) induce a clause $\overline{y}_{p_1:E.p_3} \rightarrow \underbrace{p_2 \vee p_3 \vee p_4}_{(a)} \vee \underbrace{p_3 : E_c.p_1 \vee p_3 : E_c.p_2}_{(b)} \vee \underbrace{p_2 : E.p_3}_{(c)}$, where parts (*a*) and (*b*) are just as in the previous case, part (*c*) corresponds to the same internal place in the output system net place p_2 where α_1 moves by a step execution, and E is an element net of net tokens in system net places p_1 and p_2.

The last kind of steps is synchronization steps; a synchronization step is a combination $\{t_{SN}, p_1 : \alpha_1.t_1, \ldots, p_q : \alpha_q.t_q\}$ of a system net transition t_{SN}, a set of transitions of the net tokens involved in the synchronization, and a binding ν of the variables on the input arcs of t_{SN} to these net tokens; all step transitions are labelled with the same synchronization label. A synchronization step execution consumes tokens from input places of t_{SN} and input places of internal transitions $t_1, \ldots, t_q$; in addition, it produces tokens into the output places of t_{SN}, the output places of transitions $t_1, \ldots, t_q$, and the marked places of net constants. Comparing to the system autonomous steps, the right part of an implication induced by a synchronization step is extended with the output places of internal net token transitions $t_1 \ldots t_q$.

Therefore, if p_{in} is an input system net place $p_{in} \in {}^\bullet t_{SN}$ or an input place for an involved internal transition $p_{in} \in {}^\bullet t_i$, where $t_i \in \{t_1 \ldots t_q\}$, then a synchronization step $s = \{t_{SN}, p_1 : \alpha_1.t_1, \ldots, p_q : \alpha_q.t_q\}$, where α_i is of type E_i, induces a clause

$$\overline{y}_{p_{in}} \rightarrow \underbrace{\bigvee_{p \in t_{SN}^\bullet} \overline{y}_p}_{(a)} \vee \underbrace{\bigvee_{\substack{\langle t_{SN}, p_{out}\rangle \in F_{SN} \\ \langle\langle P_c, T_c, F_c\rangle, m_0\rangle \in Con(\langle t_{SN}, p_{out}\rangle)}} \left(\bigvee_{\substack{p_c \in P_c \\ m_0(p_c) > 0}} \overline{y}_{p_{out}:E_c.p_c} \right)}_{(b)} \vee$$

$$\underbrace{\bigvee_{\substack{p_j : \alpha_j.t_j \in \{p_1 : \alpha_1.t_1, \ldots, p_q : \alpha_q.t_q\} \\ p' \in (p_j : \alpha_j.t_j)^\bullet}} \bigvee_{\substack{\langle t_{SN}, p_{out}\rangle \in F_{SN} \\ \exists x \in Var(\langle t_{SN}, p_{out}\rangle) : \nu(x) = \alpha_j}} \overline{y}_{p_{out}:E.p'}}_{(c)} \tag{5}$$

where subterm (6*a*) corresponds to the output system net places of t_{SN}; subterm (6*b*) corresponds to all marked places of all net constants on outgoing arcs of t_{SN}; and, finally, subterm (6*c*) corresponds to the output places of $t_1 \ldots t_q$ within output system net places of t_{SN} where the net token can be moved from p_{in}. For example, internal place $p_1 : \alpha_1.p_3$ in Figure 5b) and a synchronization step $s = \{SN : t_1, p_1 : \alpha_1.t_2\}$ with binding $\nu : x \rightarrow \alpha_1$, induce a clause $\overline{y}_{p_1:E_1.p_3} \rightarrow \underbrace{p_2 \vee p_3 \vee p_4}_{(a)} \vee \underbrace{p_3 : E_c.p_1 \vee p_3 : E_c.p_2}_{(b)} \vee \underbrace{p_2 : E.p_1 \vee p_2 : E.p_2}_{(c)}$, where E is an element net of net tokens in system net places p_1 and p_2, and E_c is the element net of net constant α_c on arc $\langle t_1, p_3\rangle$. Part (*a*) corresponds to output places $SN : p_2, p_3, p_4$, part (*b*) corresponds to the marked places of α_c, and part (*c*) corresponds to the output places of internal transition $p_1 : E.t_2$ of α_1, which moved from system net place p_1 to p_2.

Finally, for each internal place $p_{in} : E.p_{nt}$ of involved net token α_i within an input system net place p_{in} of t_{SN}, such that p_{nt} is not an input place for an internal transition t_i, we extend the previous implication clause with the same place p_{nt} for each output place of t_{SN} where α_i can be produced to, just similarly to subterm (4*c*) for internal net token places, in the case of a system

autonomous step.

$$\overline{y}_{p_{in}:E.p_{nt}} \rightarrow \underbrace{\bigvee_{p \in t_{SN}^{\bullet}} \overline{y}_p}_{(a)} \vee \underbrace{\bigvee_{\substack{\langle t_{SN}, p_{out}\rangle \in F_{SN} \\ \langle\langle P_c, T_c, F_c\rangle, m_0\rangle \in Con(\langle t_{SN}, p_{out}\rangle)}} \left(\bigvee_{\substack{p_c \in P_c \\ m_0(p_c)>0}} \overline{y}_{p_{out}:E_c.p_c} \right)}_{(b)} \vee$$

$$\underbrace{\bigvee_{\substack{p_j : \alpha_j.t_j \in \{p_1 : \alpha_1.t_1, \ldots, p_q : \alpha_q.t_q\} \\ p' \in (p_j : \alpha_j.t_j)^{\bullet}}} \bigvee_{\substack{\langle t_{SN}, p_{out}\rangle \in F_{SN} \\ \exists x \in Var(\langle t_{SN}, p_{out}\rangle) : \nu(x) = \alpha_j}} \overline{y}_{p_{out}:E_j.p'}}_{(c)} \vee \underbrace{\bigvee_{\substack{\langle t_{SN}, p_{out}\rangle \in F_{SN} \\ \exists x \in Var(\langle t_{SN}, p_{out}\rangle) : \nu(x) = \alpha_i}} \overline{y}_{p_{out}:E_i.p_{nt}}}_{(d)} \tag{6}$$

Subterm (6a) corresponds to the output system net places of t_{SN}; subterm (6b) corresponds to the marked places of net constants on outgoing arcs of t_{SN}; subterm (6c) corresponds to the output places of $t_1 \ldots t_q$ within output system net places of t_{SN} where the net token can be moved from p_{in}; subterm (6d) corresponds to internal place p_{nt} within output system net places where the net token α_i can be moved from p_{in}. For example, place $p_1 : \alpha_1.p_2$ and synchronization step s with binding $\nu : x \rightarrow \alpha_1$ induce a clause $\overline{y}_{p_1:E.p_3} \rightarrow \underbrace{p_2 \vee p_3 \vee p_4}_{(a)} \vee \underbrace{p_3 : E_c.p_1 \vee p_3 : E_c.p_2}_{(b)} \vee \underbrace{p_2 : E.p_3}_{(c)} \vee \underbrace{p_2 : E.p_2}_{(d)}$, where parts ($a$), ($b$), ($c$) are just as in the previous clause, part (d) corresponds to the same internal place in the output system net place p_2 where α_1 moved by a step execution, and E is an element net of net tokens in system net places p_1 and p_2.

The procedure for finding traps suggested here consists of two phases. Firstly, exploiting the structure of an NP-net, a Horn-formula represented as a system of implications is constructed; its solutions correspond to the traps of the NP-net. Secondly, the constructed Horn-formula is used to find such traps. At the second phase, it is possible to apply LTUR algorithm to find the solutions in linear time; we refer reader to [27] for details.

The first phase consists of 3 steps.

Step 1. For each system net place typed with an element net, for each elementary autonomous transition of the net, build an implication and add it to the system $\mathcal{E}$ (lines 4–11).

Step 2. For each system autonomous transition, for each input system net place p_{SN} of the transition

a) construct a clause (an implication) and add it to $\mathcal{E}$ (lines 17–37);

b) if p_{SN} is typed with an element net E, then, for each internal place of E, construct a clause (an implication) and add it to $\mathcal{E}$ (lines 17–37)

Step 3. For each vertical synchronization transition, for each possible combination of system net and element net transitions, for each input system net place p_{SN} of the transition

a) construct a clause (an implication) and add it to $\mathcal{E}$ (lines 17–37);

b) if p_{SN} is typed with an element net E, then, for each internal place p of E,

(a) if p is an input place of transition t_E involved in the synchronization, then construct an implication and add it to $\mathcal{E}$ (lines 17–37);

Algorithm 1: Algorithm for computing NP-nets traps.

Input : $NP = \langle N_1, \ldots, N_k, SN\rangle$, $SN = \langle P_{SN}, T_{SN}, F_{SN}, \upsilon, \gamma, \Lambda\rangle$,
$Ctx : P_{SN} \to A \cup \{N_1, \ldots, N_k\}$ — function that defines possible scopes for trap places within a system net place, where A is a set of net tokens in the initial marking.

Output: $C = \{c_1, c_2, \ldots, c_n\}$ — set of implications (clauses)

Begin
  $C \leftarrow \emptyset$;
  /* Elementary autonomous step implications (??) */
  foreach $p_{SN} \in \{p_{SN} \mid p \in P_{SN} \ \&\ \upsilon(p) = E = \langle P, T, F\rangle\}$ **do**
    foreach $t \in \{t \mid t \in T \ \&\ \Lambda(t) = \tau\}$ **do**
      foreach $p \in {}^\bullet t$ **do**
        if $E \in Ctx(p)$ **then** insert $\left(\overline{y}_{p_{SN}:E.p} \to \bigvee\limits_{q \in t^\bullet} \overline{y}_{p_{SN}:E.q}\right)$ into C ;
        foreach $\alpha \in Ctx(p)$ **do**
          insert $\left(\overline{y}_{p_{SN}:\alpha.p} \to \bigvee\limits_{q \in t^\bullet} \overline{y}_{p_{SN}:\alpha.q}\right)$ into C
  /* System autonomous step implications (??),(??) */
  foreach $t_{SN} \in \{t \mid t \in T_{SN} \,\&\, \Lambda(t) = \tau\}$ **do**
    $right_1 \leftarrow \left(\bigvee\limits_{p \in t_{SN}^\bullet} \overline{y}_p\right)$;
    $right_1 \leftarrow right_1 \vee CollectMarkedNetConstantPlaces(NP, t_{SN})$;
    foreach $p_{SN} \in {}^\bullet t_{SN}$ **do** insert $\left(\overline{y}_{p_{SN}} \to right_1\right)$ into C;
    insert $TrapClausesForUninvolvedInternalPlaces(NP, t_{SN}, \emptyset, Ctx, right_1)$ into C;
  /* Synchronization (2) step implications */
  foreach $t_{SN} \in \{t_{SN} \mid t_{SN} \in T_{SN} \,\&\, \Lambda(t_{SN}) \neq \tau\}$ **do**
    $right_1 \leftarrow \left(\bigvee\limits_{p \in t_{SN}^\bullet} \overline{y}_p\right)$;
    $right_1 \leftarrow right_1 \vee CollectMarkedNetConstantPlaces(NP, t_{SN})$;
    foreach $\{T_{step} = \langle t^{1,1} \ldots t^{1,n_1} \cdots t^{k,1} \ldots t^{k,n_k}\rangle \in (T_1)^{n_1} \times \cdots \times (T_k)^{n_k} \mid$
    ${}^\bullet t_{SN} = \{p_1, \ldots, p_k\}$ & $\forall i \in \overline{1,k} : \gamma(\langle p_i, t_{SN}\rangle) = x_1 + \ldots + x_{n_i}$
    & $\forall l \in \overline{1,n_i} : \Lambda(t^{i,l}) = \Lambda(t_{SN})$ & $t_{SN}{}^\bullet = \{q_1, \ldots, q_w\}$ & $\upsilon(q_j) = \langle P_{q_j}, T_{q_j}, F_{q_j}\rangle = E_{q_j}\}$
    do

Algorithm 2: Algorithm for computing NP-nets traps (contd.)

Output: C — set of implications (clauses)

foreach {

$$\langle \overset{1,1}{ctx}_{1,1}, \ldots, \overset{1,1}{ctx}_{1,m_1^{1,1}} \cdots \overset{1,1}{ctx}_{w,1}, \ldots, \overset{1,1}{ctx}_{w,m_w^{1,1}} \;, \cdots, \overset{1,n_1}{ctx}_{1,1}, \ldots, \overset{1,n_1}{ctx}_{1,m_1^{1,n_1}} \cdots \overset{1,n_1}{ctx}_{w,n_1}, \ldots, \overset{1,n_1}{ctx}_{w,m_w^{1,n_1}} \;, \cdots$$

$$\overset{k,1}{ctx}_{1,1}, \ldots, \overset{k,1}{ctx}_{1,m_1^{k,1}} \cdots \overset{k,1}{ctx}_{w,1}, \ldots, \overset{k,1}{ctx}_{w,m_w^{k,1}} \;, \cdots, \overset{k,n_k}{ctx}_{1,1}, \ldots, \overset{k,n_k}{ctx}_{1,m_1^{k,n_1}} \cdots \overset{k,n_k}{ctx}_{w,n_1}, \ldots, \overset{k,n_1}{ctx}_{w,m_w^{k,n_1}} \rangle |$$

$\forall i \in \overline{1,k} : \forall l \in \overline{1,n_i} : \forall j \in \overline{1,w} : \forall x_l \in \gamma(\langle p_i, t_{SN} \rangle) : m_j^{i,l} = |\gamma(\langle t_{SN}, q_j \rangle)|_{x_l}$ &

$\forall i \in \overline{1,k} : \forall l \in \overline{1,n_i} : \forall j \in \overline{1,w} : \forall r \in \overline{1, m_j^{i,l}} : \overset{i,l}{ctx}_{j,r} \in Ctx(p_j)$ &

$\forall i_1, i_2 \in \overline{1,k} : \forall l_1 \in \overline{1,i_1} : \forall l_2 \in \overline{1,i_2} : \forall j_1, j_2 \in \overline{1,w} : \forall r_1 \in \overline{1, m_{j_1}^{i_1,l_1}} :$

$\forall r_2 \in \overline{1, m_{j_2}^{i_2,l_2}} : (j_1 = j_2 \;\&\; (i_1 \neq i_2 \vee l_1 \neq l_2 \vee r_1 \neq r_2)$

$\&\; ctx_{j_1,r_1}^{i_1,l_1} = ctx_{j_2,r_2}^{i_2,l_2}) \rightarrow ctx_{j_1,r_1}^{i_1,l_1} = E_{q_{j_1}} \}$ **do**

  $right_2 \leftarrow right_1$;

  foreach $t^{i,l} \in T_{step}$ **do**

    foreach $j \in \overline{1,w}$ **do**

      foreach $r \in \overline{1, m_j^{i,l}}$ **do**

        foreach $q \in (t^{i,l})^\bullet$ **do** $right_2 \leftarrow (right_2 \vee \overline{y}_{p_j : ctx_{j,r}^{i,l} . q})$;

  $involvedPlaces \leftarrow \emptyset$;

  foreach $t^{i,l} \in T_{step}$ **do**

    foreach $ctx \in Ctx(p_i)$ **do**

      foreach $q \in {}^\bullet(t^{i,l})$ **do**

        insert $\left(\overline{y}_{p_i : ctx \,. q} \rightarrow right_2\right)$ into C;

        insert $p_i : ctx \,. q$ into $involvedPlaces$;

  foreach $p_{SN} \in {}^\bullet t_{SN}$ **do** insert $\left(\overline{y}_{p_{SN}} \rightarrow right_2\right)$ into C;

  insert $TrapClausesForUninvolvedInternalPlaces($
  $NP, t_{SN}, involvedPlaces, Ctx, right_2)$ into C;

Procedure CollectMarkedNetConstantPlaces(NP, t_{SN})

Input : $NP = \langle N_1, \ldots, N_k, SN\rangle$, $SN = \langle P_{SN}, T_{SN}, F_{SN}, \upsilon, \gamma, \Lambda\rangle$, $t_{SN} \in P_{SN}$
Output: $placesMarked = \{p_1, p_2, \ldots, p_n\}$ — set of element net places marked in the net constants that are produced by a t_{SN}-firing.

```
1 begin
2     placesMarked ← ∅;
3     foreach ⟨t_SN, p'⟩ ∈ {⟨t_SN, p'⟩ ∈ F_SN | |Con(⟨t_SN, p'⟩)| > 0} do
4         foreach α_c ∈ Con(⟨t_SN, p'⟩) & υ(p') = ⟨P_c, T_c, F_c⟩ = E_c & α_c = ⟨E_c, m_0⟩ do
5             foreach p_c ∈ {p_c ∈ P_c | m_0(p_c) > 0} do
6                 placesMarked ← (placesMarked ∨ ȳ_{p':E_c.p_c});
```

Procedure TrapClausesForUninvolvedInternalPlaces($NP, t_{SN}, involvedPlaces, Ctx, fixedContext$)

Input : $NP = \langle N_1, \ldots, N_k, SN\rangle$, $SN = \langle P_{SN}, T_{SN}, F_{SN}, \upsilon, \gamma, \Lambda\rangle$,
$involvedPlaces$ — internal places involved in t_{SN}-firing,
$Ctx : P_{SN} \to A \cup \{N_1, \ldots, N_k\}$, $t_{SN} \in T_{SN}$,
$fixedContext$ — disjunction that represents the set of places into which the transition t_{SN} is guaranteed to produce tokens.
Output: C_{result} — set of Horn-clauses representing trap restrictions imposed by the places uninvolved by a firing of t_{SN}.

```
1  begin
2      C_result ← ∅;
3      foreach p_SN ∈ •t_SN do
4          foreach x ∈ γ(⟨p_SN, t_SN⟩) do
5              placesTo ← ∅;
6              foreach ⟨t_SN, p'⟩ ∈ {⟨t_SN, p'⟩ ∈ F_SN | |γ(⟨t_SN, p'⟩)|_x > 0} do
7                  insert p' into placesTo;
8              foreach q ∈ {q | υ(p_SN) = E = ⟨P, T, F⟩ & q ∈ P} do
9                  foreach ctx ∈ Ctx(p_SN) do
10                     if p_SN : ctx.q ∉ involvedPlaces then
11                         insert ( ȳ_{p_SN:ctx.q} → ( fixedContext ∨ ⋁_{p'_SN ∈ placesTo} ȳ_{p'_SN:ctx.q} ) )
                           into C_result
```

(b) if p is an input place of transition t_E not involved in the synchronization, then construct an implication and add it to $\mathcal{E}$ (lines 17–37);

Step 4. Find the solutions of the system $\mathcal{E}$. Each solution of $\mathcal{E}$ corresponds to a trap of *NP*.

Thus, similar to the classical case, trap for NP-nets can be effectively represented and computed with the help of logic programming. Inferring properties from traps and siphons, probably combining them with NP-invariants, requires some manual model analysis and even some insight. However, heuristic approaches can be developed for some classes of properties. This is a subject for further research.

5. Conclusion

In this paper we have suggested one of possible generalizations of the *trap* notion to nested Petri nets formalism. The suggested definition enables to determine important behavioural properties of NP-nets models. Several behavioural properties were determined by traps in the last section. The work is another step of developing Nested Petri nets analysis techniques.

The further research will cover: extension of traps and invariants to multi-level and recursive NP-nets; estimate the (parameterized) complexity of traps finding for NP-nets;

Acknowledgments.

The publication was prepared within the framework of the Academic Fund Program at the National Research University Higher School of Economics (HSE) in 2017-2018 (grant №17-01-0089) and by the Russian Academic Excellence Project "5-100".

References

[1] A fleet of self-driving trucks just completed a 1,000-mile trip across europe, http://www.popularmechanics.com, accessed: 2016-04-07.

[2] R. Valk, Petri nets as token objects: An introduction to elementary object nets, in: Proceedings of the 19th International Conference on Application and Theory of Petri Nets, Springer-Verlag London, UK, 1998, pp. 1–25. URL http://www.springerlink.com/content/hxw7uqfxve884cr0/fulltext.pdf

[3] K. Hoffmann, H. Ehrig, T. Mossakowski, High-level nets with nets and rules as tokens, in: ICATPN, 2005, pp. 268–288. doi:10.1007/11494744_16.

[4] M. Köhler, B. Farwer, Object nets for mobility, in: ICATPN, 2007, pp. 244–262. doi:10.1007/978-3-540-73094-1_16.

[5] R. Valk, Object Petri nets using the nets-within-nets paradigm, Lecture notes in computer science (2004) 819–848.

[6] I. Lomazova, Nested petri nets - a formalism for specification and verification of multi-agent distributed systems, Fundamenta Informaticae 43 (1) (2000) 195–214. doi:10.3233/FI-2000-43123410. URL https://content.iospress.com/articles/fundamenta-informaticae/fi43-1-4-10

[7] K. M. van Hee, O. Oanea, A. Serebrenik, N. Sidorova, M. Voorhoeve, I. A. Lomazova, Checking properties of adaptive workflow nets, Fundamenta Informaticae 79 (2007) 347–362. URL http://dl.acm.org/citation.cfm?id=2367396.2367404

[8] E. Lopez-Mellado, H. Almeyda-Canepa, A three-level net formalism for the modelling of multiple mobile robot systems, in: Systems, Man and Cybernetics, 2003. IEEE International Conference on, Vol. 3, 2003, pp. 2733–2738. doi:10.1109/ICSMC.2003.1244298.

[9] L. Chang, X. He, J. Li, S. M. Shatz, Applying a nested petri net modeling paradigm to coordination of sensor networks with mobile agents, in: PNDS 2008, Proceedings, Xian, China, 2008, pp. 132–145.

[10] F. Cristini, C. Tessier, Nets-within-nets to model innovative space system architectures, in: ICATPN 2012, Vol. 7347 of LNCS, Springer, Heidelberg, 2012, pp. 348–367. doi:978-3-642-31131-4_19.

[11] L. Dworzański, I. Lomazova, Cpn tools-assisted simulation and verification of nested petri nets, Automatic Control and Computer Sciences 47 (7) (2013) 393–402. doi:10.3103/S0146411613070201.

[12] M. L. F. Venero, F. S. C. da Silva, Model checking multi-level and recursive nets, Software & Systems Modeling (2016) 1–28doi:10.1007/s10270-015-0509-6.

[13] L. Dworzanski, I. Lomazova, On compositionality of boundedness and liveness for nested petri nets, Fundamenta Informaticae 120 (3-4) (2012) 275–293. doi:10.3233/FI-2012-762.

[14] L. Dworzanski, D. Frumin, Npntool: Modelling and analysis toolset for nested petri nets, in: Proceedings of the 7th Spring/Summer Young Researchers' Colloquium on Software Engineering, 2013, pp. 9–14. doi:10.15514/syrcose-2013-7-1.

[15] L. W. Dworzanski, I. A. Lomazova, Structural place invariants for analyzing the behavioral properties of nested petri nets, in: ICATPN 2016, Proceedings, 2016, pp. 325–344.

[16] M. Minoux, K. Barkaoui, Deadlocks and traps in petri nets as horn-satisfiability solutions and some related polynomially solvable problems, Discrete Applied Mathematics 29 (2-3) (1990) 195–210. doi:10.1016/0166-218X(90)90144-2.

[17] K. Jensen, L. M. Kristensen, Coloured Petri Nets - Modelling and Validation of Concurrent Systems, Springer, 2009. doi:10.1007/b95112.

[18] I. A. Lomazova, Nested petri nets for adaptive process modeling, in: A. Avron, N. Dershowitz, A. Rabinovich (Eds.), Pillars of Computer Science, Vol. 4800 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 460–474. doi:10.1007/978-3-540-78127-1_25.

[19] M. Hack, The recursive equivalence of the reachability problem and the liveness problem for petri nets and vector addition systems, in: Switching and Automata Theory, 1974., IEEE Conference Record of 15th Annual Symposium on, IEEE, 1974, pp. 156–164.

[20] I. A. Lomazova, Nested petri nets: Multi-level and recursive systems, Fundamenta Informaticae 47 (2001) 283–293.
URL http://dl.acm.org/citation.cfm?id=1220035.1220044

[21] W. Reisig, Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies, Springer, 2013. doi:10.1007/978-3-642-33278-4.

[22] W. Reisig, Petri Nets: An Introduction, Springer-Verlag New York, Inc., New York, NY, USA, 1985.
URL https://www.springer.com/us/book/9783642699702

[23] J. Desel, J. Esparza, Free Choice Petri Nets, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2005.
URL http://books.google.ru/books?id=29cclsSodxEC

[24] F. Nabli, T. Martinez, F. Fages, S. Soliman, On enumerating minimal siphons in petri nets using clp and sat solvers: theoretical and practical complexity, Constraints 21 (2) (2016) 251. doi:10.1007/s10601-015-9190-1.
URL http://dx.doi.org/10.1007/s10601-015-9190-1

[25] R. Cordone, L. Ferrarini, L. Piroddi, Some results on the computation of minimal siphons in petri nets, in: Decision and Control, 2003. Proceedings. 42nd IEEE Conference on, Vol. 4, IEEE, 2003, pp. 3754–3759.

[26] R. Cordone, L. Ferrarini, L. Piroddi, Enumeration algorithms for minimal siphons in petri nets based on place constraints, IEEE Trans. Systems, Man, and Cybernetics, Part A 35 (6) (2005) 844–854. doi:10.1109/TSMCA.2005.853504.

[27] M. Minoux, Ltur: A simplified linear-time unit resolution algorithm for horn formulae and computer implementation, Information Processing Letters 29 (1) (1988) 1–12.