Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 312 (2015) 125–141

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

A Calculus for Automatic Verification of Petri Nets Based on Resolution and Dynamic Logics[1]

Cláudia Nalon[2]

Departament of Computer Science
University of Brasília

Bruno Lopes[3]

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro

Gilles Dowek[4]

Deducteam
Institut National de Recherche en Informatique et en Automatique

Edward Hermann Haeusler[5]

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro

Abstract

Petri Nets are a widely used formalism to deal with concurrent systems. Dynamic Logics (DL) are a family of modal logics where each modality corresponds to a program. This works presents a resolution-based method for Petri-PDL, a DL where programs are replaced by Petri Nets. We present a procedure to convert any Petri-PDL formula into a normal form, a set of resolution-based inference rules, examples of application of the method, and discuss soundness and completeness.

Keywords: Dynamic logic, Petri Nets, Resolution, Deductive systems

[1] This work was partially supported by the Brazilian research agencies CNPq, CAPES and FAPERJ.
[2] Email: `nalon@unb.br`
[3] Email: `bvieira@inf.puc-rio.br`
[4] Email: `gilles.dowek@inria.fr`
[5] Email: `hermann@inf.puc-rio.br`

http://dx.doi.org/10.1016/j.entcs.2015.04.008

1 Introduction

Concurrency and synchronisation play an important role in computer science, specially concerning distributed and asynchronous environments, which reflect usual situations in network-based systems. Modelling and verifying if the requirements are satisfied in these systems takes a lot of effort and are always a challenge. Among the most used formalisms to solve these problems, Petri Nets stands in a special place [2,4,19]. They present a theoretical background that support these characteristics and present an intuitive graphical representation. Quoting James L. Peterson [15]

> "[...] when synchronization is necessary, for instance when both a job and an idle processor must be available for processing to start, the situation is also easily modelled. Thus a Petri net would seem to be ideal for modelling systems of distributed control with multiple processes occurring concurrently."

Dynamic Logics, in general, are modal logic systems where each program is a modality. These logics are used to deal with the verification of properties in programs and Propositional Dynamic Logic (PDL) [6] is one of its most well-known variants, dealing only with regular programs (*regular* in the sense that they are specified similar to regular expressions). In PDL each program P corresponds to a modality $\langle P\rangle$ where a formula $\langle P\rangle\varphi$ means that after the running of P, φ is possibly true, considering that P stops. There is also the possibility of using $[P]\varphi$ (as an abbreviation for $\neg\langle P\rangle\neg\varphi$) indicating that the property denoted by φ holds after every possible running of P. Several systems and tools have been proposed to deal with problems expressed in Dynamic Logics, including tools for automatic verification of programs [7,8,9].

Petri-PDL [11] is a dynamic logic derived from PDL where each modality is a Marked Petri Net. If π is a Petri Net with markup s, then a Petri-PDL formula $\langle s,\pi\rangle\varphi$ means that after the running of π with the initial markup s, φ will possibly be true (the $\Box$-like modality, given by $[s,\pi]$, is also possible, where $[s,\pi]\varphi$ is an abbreviation for $\neg\langle s,\pi\rangle\neg\varphi$). In order to simplify the approach, the language of Petri-PDL refers to a subclass of Petri Nets, where each Petri Net is a composition of basic Petri Nets built from only three kinds of transitions. However, this restriction in the language does not reflect on the class of problems that can be expressed in Petri-PDL: as shown in [1], the combination of those three types of transitions is enough to represent any possible Marked Petri Net.

The axiomatisation of Petri-PDL is sound and complete, and the satisfiability problem is decidable [11]. Other systems have been proposed in the literature to reason about the properties and behaviour of Petri Nets, but they lack some of these properties, as, for instance, the Trace Theory [12,13], which is incomplete and undecidable. Other systems may retain those properties, but reasoning can only be carried out after the translation of Petri Nets into the target language, as, for instance, in [18], where PDL is used as a query language for properties of Petri Nets. By embedding Petri Nets as part of the language, Petri-PDL provides a natural way of specifying complex systems and the properties to be verified. Also, the provision of such abstractions at the (logical) specification level comes at no extra cost: the complexity of the satisfiability problem for Petri-PDL is the same as the complexity of the satisfiability problem for PDL, that is, EXPTIME-complete [11].

In this paper, as a first step towards the automation of proofs for Petri-PDL, we present a resolution-based calculus to deal with the (un)satisfiability problem in Petri-PDL. The method is clausal: a formula to be proved unsatisfiable is translated into a normal form, which separates the different contexts to which a set of resolution-based inference rules are applied. Correctness results are briefly discussed and some examples are provided.

The paper is structured as follows. In the next section, we explain the basics of Marked Petri Nets. In Section 3, we introduce the syntax, semantics, and the axiomatisation of Petri-PDL. The resolution-based method for Petri-PDL is presented in Section 4: the transformation into the normal form, the inference rules, the main results, and a few examples are given. Conclusions and future work are given in Section 5.

2 Petri Nets

A Petri Net [16] is a 3-tuple $\langle P, T, W\rangle$, where P is a finite set of places, T is a finite set of transitions with $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$ and W is a partial weighting function which associates directed edges between places and transitions to a multiplicative weight, that is, $W\colon (P \times T) \cup (T \times P) \to \mathbb{N}$. In this work we assume $w = 1$ for all edges.

Petri Nets can be extended to represent the flow of resources from one place to another. Tokens denote the amount of a resource available in a place. A Marked Petri Net is a 4-tuple $\pi = \langle P, T, W, M_0\rangle$, where P, T and W are defined as above. M_0 is an initial distribution of tokens over the places, that is, it is formally defined as $M_0 : P \to \mathbb{N}$. In the following, when referring to a Petri Net we mean a Marked Petri Net.

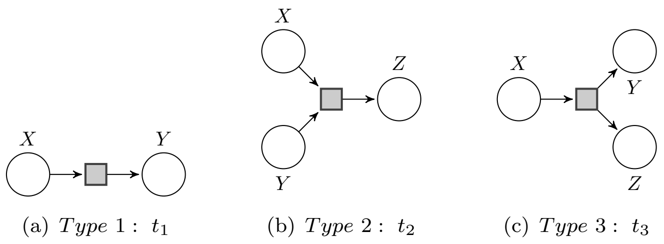
We define the preset of $t \in T$, denoted by ${}^{\bullet}t$, as the set of all $s_i \in P$ that origins an edge to t. The postset of t, denoted by $t^{\bullet}$ is defined as the set of all $s_j \in P$ that t origins an edge to. We say that a transition t is enabled if, and only if, there is at least one token in each place $s \in {}^{\bullet}t$.

From M_0 we can define the behaviour of a Marked Petri Net π by the flow of tokens (i.e. a list of markups). The function M_i (i.e. the distribution of tokens over π after i firings) over enabled transitions t is defined as:

$$M_{i+1}(x) = \begin{cases} M_i(x) - 1, \ \forall x \in {}^{\bullet}t \setminus t^{\bullet} \\ M_i(x) + 1, \ \forall x \in t^{\bullet} \setminus {}^{\bullet}t \\ M_i(x), \qquad \text{other case} \end{cases} . \tag{1}$$

In the following, if $\pi = \langle P, T, W, M_0\rangle$ and $\pi' = \langle P', T', W', M_0'\rangle$ are Petri Nets, with $P' \subseteq P$, $T' \subseteq T$, where for all $t \in T$ exists $t' \in T'$ such that ${}^{\bullet}t' = {}^{\bullet}t$, $W' \subseteq W$, and $M_0' \subseteq M_0$, we say that π' is a subnet of π, denoted by $\pi' \subseteq \pi$. The concept of subnet corresponds to that of graph inclusion with the appropriate restrictions on the relation W and on the function M_0.

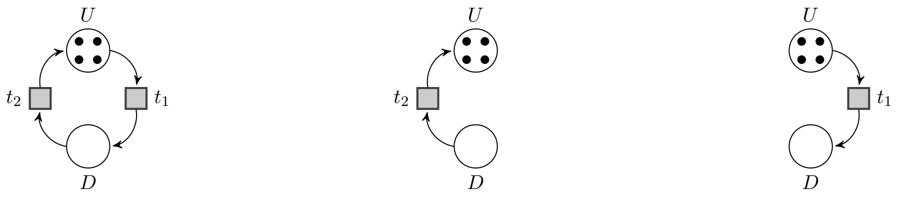
We define Petri Nets as in the work of Almeida & Haeusler [1], where it has been shown that every valid Petri Net can be represented by the composition of only three basic Petri Nets. As shown in Fig. 1, in a basic Petri Net of Type 1, a transition connects two places; in a basic Petri Net of Type 2, two places are connected by a transition to one place; and in a basic Petri Net of Type 3, one place is connected by a transition to two places, where places are represented by circles, tokens are represented by black circles within a place, and transitions are represented by boxes.



(a) *Type* $1: t_1$ (b) *Type* $2: t_2$ (c) *Type* $3: t_3$

Fig. 1. Basic Petri Nets

As an example, the Petri Net on Fig. 2(a) represents the operation of an elevator for a building with five floors. There are two places, U (the elevator can go up) and D (the elevator can go down), and tokens within those places say how many floors the elevator can travel in the specified direction. For instance, one token in the place U indicates that the elevator is able to go up one floor. If t_1 fires, a token goes from U to D, representing that the elevator can go down one floor. If the elevator goes down one floor (i.e. t_2 fires) a token goes from D to the place U. Figure 2(a) illustrates the Petri Net with its initial markup, that is, the elevator is on the ground floor and can go up to the highest floor. Note that this Petri net is a composition of the two Petri Nets of Figures 2(b) and 2(c), where places have the same names given in Figure 2(a).



(a) Petri Net for an elevator (b) To go up (c) To go down

Fig. 2. Petri Nets of an elevator and basics for composition

3 Petri-PDL Language and Semantics

The language of Petri-PDL [11] is constructed from a denumerable set of propositional symbols $\Phi = \{p, q, \ldots\}$, a set of place names $\mathcal{P} = \{a, b, c, d, \ldots\}$, a set of transition names $\mathcal{T} = \{t_1, t_2, t_3, \ldots\}$, the classical operators for negation ($\neg$) and conjunction ($\wedge$), and a set of modal operators, $\langle s, \pi \rangle$, where s is a sequence of names (i.e. $s \in \mathcal{P}^\star$ is a sequence representing the marking on a Petri Net, where $\epsilon \in \mathcal{P}^\star$ is the empty sequence), and π is a Petri Net program, defined as follows. Firstly, each transition name is associated with an unique type, either T_1, T_2, or T_3. We write $T_i : t$, for $t \in \mathcal{T}$, to denote that the transition t has type T_i, $i \in \{1, 2, 3\}$. In the following, we will often identify a transition name t_i with a type T_i, that is, we have that $T_i : t_i$. A *basic Petri Net program* π_b has one of the syntactical forms given by the following BNF:

$$\pi_b ::= at_1b \mid at_2bc \mid abt_3c$$

where t_i are transition names of type T_i and $a, b, c \in \mathcal{P}$. A *Petri Net program* π is built from basic Petri Net programs as given by the following BNF:

$$\pi ::= \pi_b \mid \pi \odot \pi$$

where π_b is a basic Petri Net program and $\odot$ is the composition operator. The set of well-formed formulae of Petri-PDL can now be defined as follows:

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle s, \pi \rangle \varphi$$

where $p \in \Phi$, $\top$ is a nullary connective, φ is a formula, $s \in \mathcal{P}^\star$, and π is a Petri Net program. We use the standard abbreviations for the nullary connective $\perp$ ($\perp \equiv \neg\top$), disjunction ($\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$), implication ($\varphi \to \psi \equiv \neg(\varphi \wedge \neg\psi)$), and the box-like modality ($[s, \pi]\varphi \equiv \neg\langle s, \pi \rangle \neg\varphi$).

The semantics of Petri-PDL is given in terms of a Kripke structure, as usual in modal logics. First, we define the firing function. Let s and s' be sequences of place names. We use the notation $s \prec s'$ to denote that all names occurring in s also occur in s', regardless of order.

Definition 3.1 We define the *firing function* $f \colon \mathcal{P}^\star \times \Pi \to \mathcal{P}^\star$ as follows, where $a, b, c \in \mathcal{P}$, $T_i : t_i \in \mathcal{T}$, for $i = 1, 2, 3$, $s, s_1, s_2, s_3 \in \mathcal{P}^\star$, Π is a set of basic Petri Net programs, and $\pi \in \Pi$:

$$f(s, at_1b) = \begin{cases} s_1bs_2 \text{ if } s = s_1as_2 \text{ and } a \not\prec s_1 \\ \epsilon \text{ if } a \not\prec s \end{cases}$$

$$f(s, abt_2c) = \begin{cases} s_1cs_2s_3 \text{ if } s = s_1as_2bs_3, a \not\prec s_1, \text{ and } b \not\prec s_2 \\ \epsilon \text{ if } a \not\prec s \text{ or } b \not\prec s \end{cases}$$

$$f(s, at_3bc) = \begin{cases} s_1s_2bc \text{ if } s = s_1as_2 \text{ and } a \not\prec s_1 \\ \epsilon \text{ if } a \not\prec s \end{cases}$$

$$f(\epsilon, \pi) = \epsilon, \text{ for all basic Petri Net programs } \pi$$

Note that, because order of places in a sequence are not regarded, in the case of a transition of type T_2, $f(s, abt_2c)$ also applies if $s = s_2bs_1as_3$, $b \not\prec s_2$ and $a \not\prec s_1$. The firing function is only defined for basic Petri Net programs. The behaviour of the whole Petri Net can be determined by the interleaving of the application of this function to its components (see axiom PC below).

Given a Petri Net $\pi = \langle P', T', W, M_0 \rangle$, where $P' \subseteq \mathcal{P}$, $T' \subseteq \mathcal{T}$, W and M_0 are defined as before, we define a Kripke model for π as follows.

Definition 3.2 A *Petri-PDL model for* $\pi = \langle P', T', W, M_0 \rangle$ is a tuple $\langle S, s_0, R_\pi, L, V \rangle$, where S is a non-empty set of states; $s_0 \in S$ is the initial state; $L\colon S \to \mathcal{P}^\star$ is a labelling function which associates each state with a sequence of place names, R_π is a binary relation over S, and $\mathbf{V}\colon \Phi \to 2^S$ is a valuation function. The relation R_π is inductively defined where $R_{\pi_b} \subseteq S \times S$.

- $f(L(w), \pi) \neq \epsilon$ iff $wR_{\pi_b}v \circ vR_\pi u$ for all $\pi_b \subset \pi$
- if $f(L(w), \pi) = \epsilon$, $wR_\pi v$ iff $w = v$

Let $\{\pi_1, \ldots, \pi_n\}$ be all the basic Petri Net programs in π. The relation R_π is defined as $\{(w, v) \mid$ for some π_i there exists u such that $f(L(w), \pi_i) \prec L(u)$, $wR_{\pi_i}u$ and $uR_\pi v\}$ for all $1 \leq i \leq n$.

The relation R_π over S denotes the flow of a Petri Net, according to equation (1), where s_0 is the state that corresponds to the initial markup, M_0. The markup of a Petri-PDL program that corresponds to π in each state can be retrieved by L.

Definition 3.3 Let $\mathcal{M} = \langle S, s_0, R_\pi, L, V \rangle$ be a Petri-PDL model for π. The *satisfaction* of a formula in $\mathcal{M}$ at a state w, denoted by $\mathcal{M}, w \Vdash \varphi$, can be inductively defined as follows (where φ, φ_1, φ_2 are formulae, s is a sequence of names, π_b, η_i are basic Petri Net programs, and $\eta = \eta_1 \odot \ldots \odot \eta_n$, $n \in \mathbb{N}$, is a Petri Net program):

- $\mathcal{M}, w \Vdash p$ iff $w \in \mathbf{V}(p)$;

- $\mathcal{M},w \Vdash \top$;
- $\mathcal{M},w \Vdash \neg\varphi$ iff $\mathcal{M},w \nVdash \varphi$;
- $\mathcal{M},w \Vdash \varphi_1 \wedge \varphi_2$ iff $\mathcal{M},w \Vdash \varphi_1$ and $\mathcal{M},w \Vdash \varphi_2$;
- $\mathcal{M},w \Vdash \langle s,\eta\rangle\varphi$ if there exists $v \in S$, $wR_\eta v$, $s \prec L(w)$ and $\mathcal{M},v \Vdash \varphi$.

Note that, from Definitions 3.3 and 3.2, a modality labelled by a basic Petri Net program is satisfied if the respective transition is *fired* with the markup s. That is, if π_b is of the form at_1b, $\langle s,\pi_b\rangle\varphi$ is satisfied at a state $w \in S$ iff there exists $v \in S$, $wR_{at_1b}v$, $a \prec s$ and $\mathcal{M},v \Vdash \varphi$; if π_b is of the form abt_2c, $\langle s,\pi_b\rangle\varphi$ is satisfied at w iff there exists $v \in S$, $wR_{abt_2c}v$, $a \prec s$, $b \prec s$ and $\mathcal{M},v \Vdash \varphi$; finally, if π_b is of the form at_3bc, $\langle s,\pi_b\rangle\varphi$ is satisfied at w iff there exists $v \in S$, $wR_{at_3bc}v$, $a \prec s$ and $\mathcal{M},v \Vdash \varphi$. If a Petri Net program π is a composition of basic Petri Net programs of the form $\pi_1 \odot \ldots \odot \pi_n$, then the modality is satisfied if at least one of its compounds π_i is *fired* with the markup s and the Petri Net π program satisfies φ taking as input the resulting markup $f(s,\pi_i)$, i.e. from the definition of R_π,we have that $\langle s,\pi\rangle\varphi$ is satisfied at w iff $\bigvee_{i=1}^{n}\langle s,\pi_i\rangle\langle f(s,\pi_i),\pi\rangle\varphi$ is satisfied at w.

Satisfaction in a Petri-PDL model $\mathcal{M} = \langle S, s_0, R_\pi, L, V\rangle$ is given with respect to the initial state. If $\mathcal{M}, s_0 \Vdash \varphi$, we say that φ is *satisfied in the model* $\mathcal{M}$, denoted by $\mathcal{M} \Vdash \varphi$. If φ is satisfied in all models $\mathcal{M}$ we say that φ is *valid*, denoted by $\Vdash \varphi$.

The axiomatisation for Petri-PDL consists of the following set of axioms and inference rules, where p and q are propositional symbols, φ and ψ are formulae, and π and $\eta = \eta_1 \odot \eta_2 \odot \cdots \odot \eta_n$ are Petri Net programs.

(PL) Enough propositional logic tautologies

(K) $[s,\pi](p \to q) \to ([s,\pi]p \to [s,\pi]q)$

(Du) $[s,\pi]p \leftrightarrow \neg\langle s,\pi\rangle\neg p$

(PC) $\langle s,\eta\rangle\varphi \leftrightarrow \langle s,\eta_1\rangle\langle f(s,\eta_1),\eta\rangle\varphi \vee \langle s,\eta_2\rangle\langle f(s,\eta_2),\eta\rangle\varphi \vee \cdots \vee \langle s,\eta_n\rangle\langle f(s,\eta_n),\eta\rangle\varphi$, where $\eta = \eta_1 \odot \eta_2 \odot \cdots \odot \eta_n$

(R_ϵ) $\langle s,\pi\rangle\varphi \leftrightarrow \varphi$, if $f(s,\pi) = \epsilon$

(Sub) If $\Vdash \varphi$, then $\Vdash \varphi^\sigma$, where σ uniformly substitutes proposition symbols by arbitrary formulae

(MP) If $\Vdash \varphi$ and $\Vdash \varphi \to \psi$, then $\Vdash \psi$

(Gen) If $\Vdash \varphi$, then $\Vdash [s,\pi]\varphi$

The axioms (PL), (K), and (Du), together with the inference rules (Sub), (MP), and (GEN), correspond to the usual axiomatisation for normal modal logics. The axiom (PC) expresses how the interleaving of firings in a Petri Net program occurs and (R_ϵ) expresses the behaviour when no transition is enabled in a Petri Net.

4 A Resolution system for Petri-PDL

In this section, we present a clausal resolution-based calculus for Petri-PDL. In order to prove that a formula φ is valid, we apply the inference rules to the clausal form of the negated formula, $\neg\varphi$. The transformation into the normal form follows [14] and [3], which use anti-prenexing together with simplification, followed by rewriting and renaming to separate the contexts to which the inference rules are applied.

4.1 Normal Form

Let φ be a formula in the language of Petri-PDL. The set of inference rules are applied to the transformation of φ into a specific normal form, called *Divided Separated Normal Form for Petri-PDL* ($\mathsf{DSNF_{PPDL}}$), which separates the contexts (formulae which are true only at the initial state; formulae which are true in all states) for reasoning. Before applying the transformation, we require that a formula φ to be in Anti-Prenex Normal Form (APNF), i.e. when modal operators are moved inwards a formula as much as possible. It has been shown in [5] that the transformation of a given problem into anti-prenex normal form (i.e. when quantifiers are moved inwards a formula) results in a better set of clauses for First-Order Logics. The same approach for modal logics was investigated in [14], where it has been shown that anti-prenexing together with simplification may also result in a better set of clauses for a particular logic if its language allows for collapsing and/or simplification of modal operators (e.g. $\Box\Box\phi$ is simplified to ϕ, in $\mathsf{S5}$, which reduces the nesting of modal operators). The application of such a technique to formulae in the language of Petri-PDL is justified by the axiom (PC), which allows similar simplifications. The transformation rules into APNF are given after the following definitions.

Definition 4.1 A *literal* is either p or $\neg p$, for $p \in \Phi$. The literals l and $\neg l$ are called *complementary literals.* A *modal literal* is either $\langle s, \pi\rangle l$ or $[s, \pi]l$, where s is a sequence of names, π is a Petri Net program, and l is a literal.

Definition 4.2 A *modal term* is a formula of the form $M_1 \dots M_k l$, where l is a literal and M_i is of the form $[s_i, \pi_i]$ or $\langle s_i, \pi_i\rangle$, $1 \leq i \leq k$, $k \in \mathbb{N}$, where each s_i is a sequence of place names and π_i is a Petri Net program.

Note that when $k = 0$, the literal l is not preceded by any modal operator.

Definition 4.3 Let φ and ψ be formulae in the language of Petri-PDL. A formula χ is in Anti-Prenex Normal Form (APNF) if, and only if,

(i) χ is a modal term; or

(ii) χ is of the form $(\varphi \wedge \psi)$ or $(\varphi \vee \psi)$, and φ and ψ are in APNF;

(iii) χ is of the form $[s, \pi]\varphi$, φ is disjunctive, and φ is in APNF; or

(iv) χ is of the form $\langle s, \pi\rangle\varphi$, φ is conjunctive, and φ is in APNF.

We define a function $\alpha(\varphi)$, where φ is a formula, which produces the anti-prenex normal form of φ. The base case (i) occurs when the formula φ is already in APNF, that is, φ is a modal term. In this case, $\alpha(\varphi) = \varphi$. If the main operator is modal, it can be distributed over subformulae in the following cases (where φ and ψ are formulae):

$$\alpha([s,\pi](\varphi \to \psi)) = \alpha([s,\pi]\varphi \to [s,\pi]\psi)$$
$$\alpha([s,\pi](\varphi \wedge \psi)) = \alpha([s,\pi]\varphi \wedge [s,\pi]\psi)$$
$$\alpha([s,\pi]\neg(\varphi \to \psi)) = \alpha([s,\pi]\varphi \wedge [s,\pi]\neg\psi)$$
$$\alpha([s,\pi]\neg(\varphi \vee \psi)) = \alpha([s,\pi]\neg\varphi \wedge [s,\pi]\neg\psi)$$
$$\alpha(\langle s,\pi\rangle(\varphi \to \psi)) = \alpha(\langle s,\pi\rangle\neg\varphi \vee \langle s,\pi\rangle\psi)$$
$$\alpha(\langle s,\pi\rangle(\varphi \vee \psi)) = \alpha(\langle s,\pi\rangle\varphi \vee \langle s,\pi\rangle\psi)$$
$$\alpha(\langle s,\pi\rangle\neg(\varphi \wedge \psi)) = \alpha(\langle s,\pi\rangle\neg\varphi \vee \langle s,\pi\rangle\neg\psi)$$

If we have two consecutive modal operators, the function is applied recursively, where φ is of the form $[s',\pi']\psi$ or $\langle s',\pi'\rangle\psi$, for any s' a sequence of place names and π' a Petri Net program, and ψ is a formulae which is not in APNF:

$$\alpha([s,\pi]\varphi) = \alpha([s,\pi]\alpha(\varphi))$$
$$\alpha(\langle s,\pi\rangle\varphi) = \alpha(\langle s,\pi\rangle\alpha(\varphi))$$

If the main operator is a modal operator, but the formula inside its scope is not one of the above, we apply the anti-prenexing function to this formula, that is:

$$\alpha([s,\pi]\varphi) = [s,\pi]\alpha(\varphi)$$
$$\alpha(\langle s,\pi\rangle\varphi) = \langle s,\pi\rangle\alpha(\varphi)$$

When the main operator is classical, the transformation function is also applied recursively. Note that when the polarity of a subformula is negative, we rewrite the formula in order to make this explicit.

$$\alpha(\neg[s,\pi]\varphi) = \alpha(\langle s,\pi\rangle\neg\varphi) \qquad \alpha(\neg\langle s,\pi\rangle\varphi) = \alpha([s,\pi]\neg\varphi)$$
$$\alpha(\varphi \to \psi) = \alpha(\neg\varphi) \vee \alpha(\psi) \qquad \alpha(\neg(\varphi \to \psi)) = (\alpha(\varphi) \wedge \alpha(\neg\psi))$$
$$\alpha(\varphi \wedge \psi) = \alpha(\varphi) \wedge \alpha(\psi) \qquad \alpha(\neg(\varphi \wedge \psi)) = (\alpha(\neg\varphi) \vee \alpha(\neg\psi))$$
$$\alpha(\varphi \vee \psi) = \alpha(\varphi) \vee \alpha(\psi) \qquad \alpha(\neg(\varphi \vee \psi)) = (\alpha(\neg\varphi) \wedge \alpha(\neg\psi))$$

Because of the axiom (PC) and seriality, simplification of modal operators can be applied at any step of the transformation into the anti-prenexing normal form, as follows (where $\pi = \pi_1 \odot \pi_2 \odot \cdots \odot \pi_n$, $1 \leq i \leq n$):

$$\alpha([s,\pi_i][f(s,\pi_i),\pi]\varphi) = \alpha(\langle s,\pi\rangle\varphi)$$
$$\alpha([s,\pi_i]\langle f(s,\pi_i),\pi\rangle\varphi) = \alpha(\langle s,\pi\rangle\varphi)$$
$$\alpha(\langle s,\pi_i\rangle[f(s,\pi_i),\pi]\varphi) = \alpha(\langle s,\pi\rangle\varphi)$$
$$\alpha(\langle s,\pi_i\rangle\langle f(s,\pi_i),\pi\rangle\varphi) = \alpha(\langle s,\pi\rangle\varphi)$$

Note that, at the end of the transformation, $\alpha(\varphi)$ is in both APNF and in Negation

Normal Form (that is, a formula where the connectives are restricted to $\neg$, $\vee$, $\wedge$, the modal operators are applied to literals, disjunctions or conjunctions, as given in Definition 4.3, and the negations are applied only to propositional symbols). The proof that the transformation into APNF is satisfiability preserving is given by the following lemma.

Lemma 4.4 *Let φ be a Petri-PDL formula and let $\alpha(\varphi)$ be a formula resulting from the transformation of φ into APNF. $\Vdash \varphi$ if, and only if, $\Vdash \alpha(\varphi)$.*

Proof. *The proof follows immediately from the soundness and completeness of the axiomatisation of Petri-PDL.* $\square$

In order to separate the contexts for reasoning, we define a *Petri-PDL problem* to be a tuple $\langle \mathcal{I}, \mathcal{U} \rangle$, where $\mathcal{I}$, the set of initial formulae, is a finite set of non-modal propositional formulae; and $\mathcal{U}$, the set of universal formulae, is a finite set of formulae in the language of Petri-PDL. Let $\mathcal{M} = \langle S, s_0, R_\pi, M, \mathbf{V} \rangle$ be a Petri-PDL model. We say that a Petri-PDL problem $\mathcal{Q} = \langle \mathcal{I}, \mathcal{U} \rangle$ is satisfied in $\mathcal{M}$ (denoted by $\mathcal{M} \Vdash \mathcal{Q}$) if and only if $\mathcal{M}, s_0 \Vdash \mathcal{I}$ and, for all $s \in S$, $\mathcal{M}, s \Vdash \mathcal{U}$ (where satisfiability of sets is defined in the usual way).

Let $\alpha(\varphi)$ be a formula in APNF. The set of resolution-based inference rules, given in Section 4.2 are applied to the transformation of $\alpha(\varphi)$ into a *clausal Petri-PDL problem*, which is formally defined as a Petri-PDL problem $\langle \mathcal{I}, \mathcal{U} \rangle$, where $\mathcal{I}$, the set of initial clauses, contains formulae in the form of

$$\bigvee_i l_i,$$

$i \in \mathbb{N}$, where l_i are literals; and $\mathcal{U}$, the set of universal clauses, contains formulae in the form of

$$\bigvee_i l_i \vee \bigvee_j [s_j, \pi_j] l'_j \vee \bigvee_k \neg [s'_k, \pi'_k] l''_k,$$

$i, j, k \in \mathbb{N}$, where l_i, l'_j, l''_k are literals, s_j, s'_k are sequences of place names, and π_j, π'_k are Petri Net programs.

The transformation of a formula φ into the clausal form starts by taking the problem $\langle \{t_0\}, \{t_0 \rightarrow \alpha(\varphi)\} \rangle$, where t_0 is a new propositional symbol (i.e. a propositional symbol that does not occur in φ), and applying exhaustively the following rewriting rules (where t is a literal, t_1 is a new propositional symbol, and ψ_1, ψ_2 are formulae):

(τ_1) $\langle \mathcal{I}, \mathcal{U} \cup \{t \rightarrow (\psi_1 \wedge \psi_2)\} \rangle \longrightarrow \langle \mathcal{I}, \mathcal{U} \cup \{t \rightarrow \psi_1, t \rightarrow \psi_2\} \rangle$;

(τ_2) $\langle \mathcal{I}, \mathcal{U} \cup \{t \rightarrow (\psi_1 \vee \psi_2)\} \rangle \longrightarrow \langle \mathcal{I}, \mathcal{U} \cup \{t \rightarrow (\psi_1 \vee t_1), t_1 \rightarrow \psi_2\} \rangle$,
if ψ_2 is not a modal literal;

(τ_3) $\langle \mathcal{I}, \mathcal{U} \cup \{t \rightarrow \langle s, \pi \rangle \psi_1\} \rangle \longrightarrow \langle \mathcal{I}, \mathcal{U} \cup \{t \rightarrow \langle s, \pi \rangle t_1, t_1 \rightarrow \psi_1\} \rangle$,
if ψ_1 is not a literal;

(τ_4) $\langle \mathcal{I}, \mathcal{U} \cup \{t \to [s,\pi]\psi_1\rangle \longrightarrow \langle \mathcal{I}, \mathcal{U} \cup \{t \to [s,\pi]t_1, t_1 \to \psi_1\}\rangle$,
if ψ_1 is not a literal .

Note that we take conjunctions and disjunctions as being associative and commutative. Thus, for instance, the transformation rule (τ_2) also applies when ψ_1 is not a literal. As a final step, we replace the modal operator $\langle s,\pi\rangle$ by its dual and rewrite implications as disjunctions, that is, we apply the following rewriting rules (where t, l are literals, D is a disjunction of literals and/or modal literals, s is a sequence of names, and π is a Petri Net program):

(τ_5) $\langle \mathcal{I}, \mathcal{U} \cup \{t \to D \vee \langle s,\pi\rangle l\}\rangle \longrightarrow \langle \mathcal{I}, \mathcal{U} \cup \{t \to D \vee \neg[s,\pi]\neg l\}\rangle$;

(τ_6) $\langle \mathcal{I}, \mathcal{U} \cup \{t \to D\}\rangle \longrightarrow \langle \mathcal{I}, \mathcal{U} \cup \{\neg t \vee D\}\rangle$.

We note that simplification takes place at any step of the transformation (i.e. in the application of both α and τ), that is, we remove occurrences of the constants $\top$ and $\bot$ as well as duplicates of formulae in conjunctions and disjunctions. This is achieved by exhaustively applying the following simplification rules (where conjunctions and disjunctions are commutative, φ is a formula, s is a sequence of place names, and π is a Petri Net program):

$$\begin{array}{lcllcllcl}
\varphi \wedge \top & \longrightarrow & \varphi & \neg\top & \longrightarrow & \bot & \varphi \vee \neg\varphi & \longrightarrow & \top \\
\varphi \vee \top & \longrightarrow & \top & \neg\bot & \longrightarrow & \top & \varphi \wedge \neg\varphi & \longrightarrow & \bot \\
\varphi \wedge \bot & \longrightarrow & \bot & \varphi \vee \varphi & \longrightarrow & \varphi & [s,\pi]\top & \longrightarrow & \top \\
\varphi \vee \bot & \longrightarrow & \varphi & \varphi \wedge \varphi & \longrightarrow & \varphi & [s,\pi]\bot & \longrightarrow & \bot \\
 & & & \neg\neg\varphi & \longrightarrow & \varphi & & &
\end{array}$$

The transformation of a Petri-PDL formula into the clausal normal form is satisfiability preserving, as shown by the next lemma.

Lemma 4.5 *Let φ be a well-formed formula in the language of Petri-PDL. φ is satisfiable if, and only if, the transformation of $\alpha(\varphi)$ into $\mathsf{DSNF}_{\mathsf{PPDL}}$ is satisfiable.*

Proof. *The proof that the transformation of a Petri-PDL formula into its normal form is satisfiability preserving follows immediately from Lemma 4.4, as only equivalences are used in the transformation into APNF, from the soundness and completeness of the axiomatisation of Petri-PDL, and from the fact that renaming of formulae by means of the introduction of new propositional symbols, during the transformation into the clausal normal form, is satisfiability preserving.* □

4.2 Resolution rules

Let φ be a formula in the language of Petri-PDL and let $\tau(\alpha(\varphi))$ be the set of clauses resulting from the transformation of φ into the clausal normal form, as given in the previous section. The resolution method for Petri-PDL, named $\mathsf{RES}_{\mathsf{PPDL}}$, consists of applying the following inference rules to clauses in $\tau(\alpha(\varphi))$ (where C, C' are disjunctions of literals, D, D' are disjunctions of literals and/or modal literals, l, l_i, $0 \le i \le n$, are literals, m is a literal or a modal literal, s is a sequence of place names, and π, η are Petri Net programs).

$$\text{(ires)}\ \frac{\begin{array}{ll} C \lor l & \in \mathcal{I} \cup \mathcal{U} \\ C' \lor \neg l & \in \mathcal{I} \end{array}}{C \lor C' \in \mathcal{I}} \qquad \text{(ures)}\ \frac{\begin{array}{ll} D \lor m & \in \mathcal{U} \\ D' \lor \neg m & \in \mathcal{U} \end{array}}{D \lor D' \in \mathcal{U}}$$

$$\text{(ser1)}\ \frac{D \lor [s,\pi]l \quad \in \mathcal{U}}{D \lor \neg[s,\pi]\neg l \in \mathcal{U}} \qquad \text{(ser2)}\ \frac{\begin{array}{ll} D \lor \neg[s,\pi]l & \in \mathcal{U} \\ l_1 \lor \cdots \lor l_n \lor l & \in \mathcal{U} \end{array}}{D \lor \neg[s,\pi]\neg l_1 \lor \cdots \lor \neg[s,\pi]\neg l_n \in \mathcal{U}}$$

$$\begin{array}{l} \text{(ref)} \\ \text{if } f(s,\pi) = \epsilon \end{array} \frac{D \lor [s,\pi]l \in \mathcal{U}}{D \lor l \quad\quad \in \mathcal{U}} \qquad \begin{array}{l} \text{(comp)} \\ \text{if } \eta \subseteq \pi \text{ and} \\ \text{for any } \pi_b \in \pi, \end{array} \frac{\begin{array}{ll} D \lor \neg[s,\pi]l & \in \mathcal{U} \\ D' \lor [f(s,\pi_b),\eta]l & \in \mathcal{U} \end{array}}{D \lor \neg[s,\pi]\neg D' \in \mathcal{U}}$$

The inference rules (ires) and (ures) are equivalent to classical resolution applied within each context of a given problem. The inference rules (ser1) and (ser2) deal with seriality: a Petri Net cannot lead to a contradicting state. The inference rule (ref) corresponds to reflexivity and it can only be applied if $f(s,\pi) = \epsilon$. The inference rule, (comp), deals with compositionality: if η is a Petri subnet of π, then we cannot have that both $\pi = \pi_1 \odot \cdots \odot \pi_n$ and η lead to a contradicting state, through sequences of names s and $f(s,\pi_b)$ for any basic program π_b, $b = 1,\ldots,n$. This rule can only be applied to clauses where all modal literals are subnets of π. The subformula $\neg[s,\pi]\neg D'$ in the resolvent of (comp) must be translated into its normal form through rewriting as follows. If $D' = l_1 \lor \ldots \lor l_m$ is a disjunction, then $\neg[s,\pi]\neg(l_1 \lor \ldots \lor l_m)$ is translated into $\neg[s,\pi]\neg l_1 \lor \ldots \lor \neg[s,\pi]\neg l_m$, where each $\neg[s,\pi]l_i$, $i = 1,\ldots,m$, is rewritten in its simplified form. For instance, if l_i is of the form $[f(s,\pi_b),\eta']l'$, where l' is a literal, then $\neg[s,\pi]\neg l_i = \neg[s,\pi]\neg[f(s,\pi_b),\eta']l' = \neg[s,\pi]\neg l'$. The next lemma shows that the resolution rules for Petri-PDL are sound.

Lemma 4.6 *The resolution rules for Petri-PDL are sound.*

Proof. *Soundness of* (ires) *and* (ures) *follow from soundness of the resolution inference rule for propositional logic [17]. Soundness of* (ser1) *and* (ser2) *follow from* (PC) *and* (Du)*. Soundness of* (ref) *follows from* (R_ϵ)*. Soundness of* (comp) *follows from the definition of satisfiability of clauses in $\mathcal{U}$,* (K) *and* (PC). □

Definition 4.7 A *derivation* from a Petri-PDL problem in $\mathsf{DSNF}_{\mathsf{PPDL}}$ $\mathcal{Q} = \langle \mathcal{I}, \mathcal{U} \rangle$ by $\mathsf{RES}_{\mathsf{PPDL}}$ is a sequence $\mathcal{Q}_0, \mathcal{Q}_1, \mathcal{Q}_2, \ldots$ of problems such that $\mathcal{Q}_0 = \mathcal{Q}$, $\mathcal{Q}_i = \langle \mathcal{I}_i, \mathcal{U}_i \rangle$, and $\mathcal{Q}_{i+1}$ is either

- $\langle \mathcal{I}_i \cup \{D\}, \mathcal{U} \rangle$, where D is the conclusion of an application of (ires); or

- $\langle \mathcal{I}_i, \mathcal{U}_i \cup \{D\} \rangle$, where D is the conclusion of an application of (ures), (ser1), (ser2), (ref), or (comp);

and $D \neq \top$.

We note that the resolvent D is only included in the set of clauses if it is not a tautology. Also, a resolvent is always kept in the simplest form: duplicate literals are removed; $\top$ and $\bot$ are removed from conjunctions and disjunctions with more than one conjunct/disjunct, respectively; conjunctions (resp. disjunctions) with either complementary literals or $\bot$ (resp. $\top$) are simplified to $\bot$ (resp. $\top$).

Definition 4.8 A *refutation* for a Petri-PDL problem in $\mathsf{DSNF}_{\mathsf{PPDL}}$ $\mathcal{Q} = \langle \mathcal{I}, \mathcal{U} \rangle$ (by $\mathsf{RES}_{\mathsf{PPDL}}$) is a derivation from $\mathcal{Q}$ such that for some $i \geq 0$, $\mathcal{Q}_i = \langle \mathcal{I}_i, \mathcal{U}_i \rangle$ contains a contradiction, where a contradiction is given by either $\bot \in \mathcal{I}_i$ or $\bot \in \mathcal{U}_i$.

A derivation *terminates* if, and only if, either a contradiction is derived or no new clauses can be derived by further application of resolution rules of $\mathsf{RES}_{\mathsf{PPDL}}$.

4.3 Correctness

In this section, we present the main results concerning the calculus for Petri-PDL, $\mathsf{RES}_{\mathsf{PPDL}}$: soundness, termination, and complexity. We also discuss completeness of the method.

The first theorem shows that the resolution method for Petri-PDL is sound.

Theorem 4.9 *The resolution method for Petri-PDL is sound.*

Proof. *Soundness of the resolution method for Petri-PDL follows from Lemma 4.5, which shows that the transformation of a Petri-PDL formula into* $\mathsf{DSNF}_{\mathsf{PPDL}}$ *is satisfiability preserving, and from Lemma 4.6, which shows that each of the resolution inference rules is satisfiability preserving.* □

The next theorem ensures that the application of the method is terminating.

Theorem 4.10 *The resolution method for Petri-PDL is terminating.*

Proof. *Termination follows from the fact that, in a given clausal Petri-PDL problem, there are only a finite number of literals and modal literals. Also, from the fact that a derivation (as given in Def. 4.7) cannot produce an infinite number of clauses. Note, in particular, that the resolution rules* (ser1), (ser2) *and* (comp) *can generate new modal literals. As the number of modal literals of the form* $[s, \pi]l$ *is finite, only a finite number of new modal literals of the form* $\neg[s, \pi]\neg l$ *can be generated by* (ser1) *or* (comp)*. Similarly, as the number of literals is finite,* (ser2) *can only generate a finite number of new modal literals of the form* $\neg[s, \pi]\neg l$*. Therefore, only a finite number of clauses (modulo ordering and simplification) can be expressed. Thus, either we find an empty clause or the method terminates as no new clauses can be generated.* □

The following theorem shows that the resolution-based calculus for Petri-PDL is optimal, as the complexity of the satisfiability problem for Petri-PDL is EXPTIME-complete [11].

Theorem 4.11 *The resolution method for Petri-PDL runs in exponential deterministic time in the size of the Petri-PDL formula being tested for unsatisfiability.*

Proof. *Let φ be a Petri-PDL formula. It is easy to see that the transformation into the normal form is linear in the size n of φ and so it is the size of a problem (given as the number of clauses). The inference rules* (ser1)*,* (ser2)*, and* (ref) *also produce a linear number of clauses in the size of the problem. The remaining rules are syntactical variations of classical propositional resolution, which takes exponential time in the size of a formula [17]. The number of applications of such inference rules depends on the modal depth of the original formula, but this cannot be greater then the size of φ. Therefore, the resolution method for Petri-PDL runs in $O(n) \times 2^{O(n)}$, i.e. it takes exponential deterministic time.* $\square$

Completeness of the resolution method for Petri-PDL is ongoing work.

Claim 4.12 *The resolution method for Petri-PDL is complete.*

We hope to prove completeness as in the following argumentation, which is standard for proving completeness of calculi for modal logics. If the problem $\langle \mathcal{I}, \mathcal{U} \rangle$ is valid then the canonical model $\mathcal{M}$ (derived from the Petri-PDL language) satisfies $\langle \mathcal{I}, \mathcal{U} \rangle$. This canonical model generates a problem $\langle \mathcal{I}', \mathcal{U}' \rangle$ where $\mathcal{I}'$ is defined from the propositional formulae in $\mathcal{I}$ and $\mathcal{U}'$ is the Hintikka set associated with the canonical model. We can show that the resolvents from the application of the inference rules to formulae in $\langle \mathcal{I}', \mathcal{U}' \rangle$ are formulae in $\langle \mathcal{I}', \mathcal{U}' \rangle$. Thus, the empty clause cannot be derived from $\langle \mathcal{I}, \mathcal{U} \rangle$. Therefore, if $\langle \mathcal{I}, \mathcal{U} \rangle$ is valid in $\mathcal{M}$ then there is no refutation of $\langle \mathcal{I}, \mathcal{U} \rangle$.

4.4 Examples

Before concluding, we show two examples of the application of the method.

Example 4.13 Suppose we want to test the formula

$$\varphi = [s, \pi_1 \odot \pi_2](p \to q) \to ([s, \pi_1][f(s, \pi_1), \pi_1 \odot \pi_2]p \to [s, \pi_2][f(s, \pi_2), \pi_1 \odot \pi_2]q)$$

for unsatisfiability — an instance of the (K) axiom. Firstly, we transform $\neg\varphi$ into APNF, which results in:

$$\alpha(\neg\varphi) = (\langle s, \pi_1 \odot \pi_2 \rangle \neg p \vee [s, \pi_1 \odot \pi_2]q) \wedge [s, \pi_1 \odot \pi_2]p \wedge \langle s, \pi_1 \odot \pi_2 \rangle \neg q$$

The transformation of $\alpha(\neg\varphi)$ into the normal form results in the following clauses:

1. t_0 $[\mathcal{I}]$
2. $\neg t_0 \vee \neg[s, \pi_1 \odot \pi_2]p \vee [s, \pi_1 \odot \pi_2]q$ $[\mathcal{U}]$
3. $\neg t_0 \vee [s, \pi_1 \odot \pi_2]p$ $[\mathcal{U}]$
4. $\neg t_0 \vee \neg[s, \pi_1 \odot \pi_2]q$ $[\mathcal{U}]$

The refutation proceeds as follows:

5. $\neg t_0 \vee \neg[s, \pi_1 \odot \pi_2]p$ [$\mathcal{U}$,(ures),2,4]
6. $\neg t_0$ [$\mathcal{U}$,(ures),5,3]
7. $\perp$ [$\mathcal{I}$,(ires),6,1]

Example 4.14 Suppose a game where a player walks through scenarios, as represented by the Petri Net in Fig. 3. The player uses keys to open doors which separate these scenarios. The number of open doors is given by the amount of tokens in place O. The amount of keys available to the player is denoted by the amount of tokens in place K. A key can be used to open a door only if the player has a free hand, which is denoted by a token in place H. If both hands are busy there is a token in place B. We can show that, after three rounds of the game, the player has opened one door, has a free hand and still has two keys to continue the play. We represent the conjunction of these conditions by p. The three rounds are represented by the following set of clauses, where $\pi = HKt_2x \odot xt_3yO \odot yt_1H \odot Ht'_1B \odot Bt''_1H$:

1. p_0 $[\mathcal{I}]$
2. $\neg p_0 \vee \neg[(KKKH), \pi]\neg p_1$ $[\mathcal{U}]$
3. $\neg p_1 \vee \neg[(KKx), \pi]\neg p_2$ $[\mathcal{U}]$
4. $\neg p_2 \vee \neg[(KKyO), \pi]\neg p_3$ $[\mathcal{U}]$
5. $\neg p_3 \vee \neg p$ $[\mathcal{U}]$

where each p_i, $i = 1, \dots, 3$, represents the i-th moment in time. We now show that this corresponds to a possible output of the game, that is, that $\langle KKHO, \pi\rangle\neg p$ cannot hold in the initial time, which is given by the following clause:

6. $\neg p_0 \vee [(KKHO), \pi]p$ $[\mathcal{U}]$

The refutation proceeds as follows:

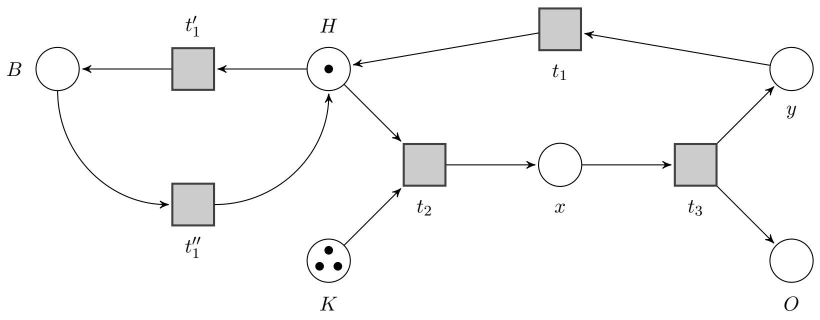


Fig. 3. A Petri Net denoting a game where doors separate scenarios

7.	$\neg p_2 \vee \neg[(KKyO), \pi]p$	$[\mathcal{U}$,(ser2),4,5]
8.	$\neg p_1 \vee \neg[(KKx), \pi]p$	$[\mathcal{U}$,(comp),7,3]
9	$\neg p_0 \vee \neg[(KKKH), \pi]p$	$[\mathcal{U}$,(comp),8,2]
10.	$\neg p_0$	$[\mathcal{U}$,(ures),9,6]
11.	$\perp$	$[\mathcal{I}$,(ires),10,1]

5 Conclusions and further work

Petri Net is one of the most used formalisms to deal with concurrent systems. Given a logical representation of a Petri Net program, it is desirable to have tools to reason about properties of the program within this framework. In this paper we have presented a sound resolution-based method for proving satisfiability of formulae in Petri-PDL, a logic system that takes advantage of the graphical interpretation of Petri Nets within a known decidable and complete fragment. In order to apply the resolution method, formulae in Petri-PDL are firstly converted into Anti-Prenex Normal Form (APNF) and simplification is applied whenever possible, which might lead to the generation of a better set of clauses. A formula in APNF is then transformed into a normal form ($\mathsf{DSNF}_{\mathsf{PPDL}}$), which separates the contexts for reasoning: an initial context (a finite set of non-modal formulae) and an universal context (a finite set of Petri-PDL formulae). The inference rules presented in Section 4 are then applied to the different contexts until a contradiction is found or no new clauses can be derived.

We have proved soundness and termination of the method and shown that the running time for testing satisfiability of a set of clauses is optimal. We are currently investigating its completeness, as given in Claim 4.12. The design of strategies for the method is left as future work. Further work also includes the implementation of an automatic theorem-prover and the extension of this method to $\mathcal{DS}_3$ [10], a logic system which extends Petri-PDL in order to deal with Stochastic Petri Nets (Petri Nets with probabilistic temporal variables).

References

[1] de Almeida, E. S. and E. H. Haeusler, *Proving properties in ordinary Petri Nets using LoRes logical language*, Petri Net Newsletter **57** (1999), pp. 23–36.

[2] Bourcerie, M., F. Bousseau and F. Guegnard, *Petri Nets for production systems: Teaching and research in europe*, in: *Global Cooperation in Engineering Education: Innovative Technologies, Studies and Professional Development - International Conference Proceedings*, 2008, pp. 85–89.

[3] Degtyarev, A., M. Fisher and B. Konev, *Monodic temporal resolution*, ACM Trans. Comput. Log **7** (2006), pp. 108–150.
URL http://doi.acm.org/10.1145/1119439.1119443

[4] Denaro, G. and M. Pezze, *Petri nets and software engineering*, in: J. Desel, W. Reisig and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, Lecture Notes in Computer Science **3098**, 2004, pp. 439–466.

[5] Egly, U., *On the value of antiprenexing*, in: F. Pfenning, editor, *Proceedings of the 5th International Conference on Logic Programming and Automated Reasoning*, LNAI **822** (1994), pp. 69–83.

[6] Fischer, M. J. and R. E. Ladner, *Propositional Dynamic Logic of regular programs*, Journal of Computer and System Sciences **18** (1979), pp. 194–211.

[7] de Giacomo, G. and F. Massacci, *Combining deduction and model checking into tableaux and algorithms for Converse-PDL*, Information and Computation **160** (1998), p. 2000.

[8] Göller, S. and M. Lohrey, *Infinite state model-checking of Propositional Dynamic Logics*, in: Z. Ésik, editor, *Computer Science Logic*, Lecture Notes in Computer Science **4207**, Springer, 2006 pp. 349–364.

[9] Lange, M., *Model checking Propositional Dynamic Logic with all extras*, Journal of Applied Logic **4** (2006), pp. 39–49.

[10] Lopes, B., M. Benevides and E. H. Haeusler, *Extending Propositional Dynamic Logic for Petri Nets*, Electronic Notes in Theoretical Computer Science **305** (2014), pp. 67–83.

[11] Lopes, B., M. Benevides and E. H. Haeusler, *Propositional dynamic logic for petri nets*, Logic Journal of the IGPL **22** (2014).

[12] Mazurkiewicz, A., *Trace theory*, in: W. Brauer., W. Reisig and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, Lecture Notes in Computer Science **255**, Springer, 1987 pp. 278–324.

[13] Mazurkiewicz, A., *Basic notions of trace theory*, in: J. W. Bakker, W.-P. Roever and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, Lecture Notes in Computer Science **354**, Springer, 1989 pp. 285–363.

[14] Nalon, C. and C. Dixon, *Anti-prenexing and prenexing for modal logics*, in: *Proceedings of the 10th JELIA* (2006), pp. 333–345.

[15] Peterson, J. L., *Petri nets*, Computing Surveys **9** (1977).

[16] Petri, C. A., *Fundamentals of a theory of asynchronous information flow*, Communications of the ACM **5** (1962), pp. 319–319.

[17] Robinson, J. A., *A Machine–Oriented Logic Based on the Resolution Principle*, ACM Journal **12** (1965), pp. 23–41.

[18] Tuominen, H., *Elementary net systems and Dynamic Logic*, in: G. Rozenberg, editor, *Advances in Petri Nets 1989*, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1990 pp. 453–466.

[19] Zurawski, R. and M. C. Zhou, *Petri Nets and industrial applications – a tutorial*, IEEE Transactions on Industrial Electronics **41** (1994), pp. 567–583.