Logic Journal of IGPL Advance Access published April 25, 2014

Propositional dynamic logic for Petri Nets

BRUNO LOPES*, *Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro - RJ, Brazil*

MARIO BENEVIDES†, *Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro - RJ, Brazil*

EDWARD HERMANN HAEUSLER‡, *Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro - RJ, Brazil.*

Abstract

Propositional Dynamic Logic (PDL) is a multi-modal logic used for specifying and reasoning on sequential programs. Petri Net is a widely used formalism to specify and to analyse concurrent programs with a very nice graphical representation. In this work, we propose a PDL to reasoning about Petri Nets. First we define a compositional encoding of Petri Nets from basic nets as terms. Second, we use these terms as PDL programs and provide a compositional semantics to PDL Formulas. Finally, we present an axiomatization and prove completeness w.r.t. our semantics. The advantage of our approach is that we can do reasoning about Petri Nets using our dynamic logic and we do not need to to translate it to other formalisms. Moreover our approach is compositional allowing for construction of complex nets using basic ones.

Keywords: Dynamic logic, temporal logic, elementary net systems, Petri Nets, model checking, model synthesis, axiomatization of properties.

1 Introduction

Dynamic Logics [9, 14] are based on the idea proposed by von Pratt [30] of a modal system where each program composes a modality. They are often used to reasoning about programs and Propositional Dynamic Logic (PDL) [9] is one of its most well-known variants (detailed in Section 2.1). The logic has a set of basic programs and a set of operators (sequential composition, iteration and non-deterministic choice) that are used to inductively build the set of non-basic programs. PDL has been used to describe and verify properties and behaviour of sequential programs and systems. Correctness, termination, fairness, liveness and equivalence of programs are among the properties that one usually wants to verify. Each program π corresponds to a modality $\langle\pi\rangle$, where a formula $\langle\pi\rangle\alpha$ means that after the running of π, α eventually is true, considering that π halts. There is also the possibility of using $[\pi]\alpha$ (as an abbreviation for $\neg\langle\pi\rangle\neg\alpha$) indicating that the property denoted by α holds after every possible running of π. A Kripke semantics can be provided, with a frame $\mathcal{F} = \langle W, R_\pi\rangle$, where W is a non-empty set of possible program states and, for each program π, R_π is a binary relation on W such that $(s,t) \in R_\pi$ if and only if there is a computation of π starting in s and terminating in t.

*E-mail: bvieira@inf.puc-rio.br
†E-mail: mario@cos.ufrj.br
‡E-mail: hermann@inf.puc-rio.br

doi:10.1093/jigpal/jzu010

There are a lot of variations of PDL for different approaches [2]. Propositional Algorithmic Logic [25] that analyses properties of programs connectives, the interpretation of Deontic Logic as a variant of Dynamic Logic [24], applications in linguistics [18], Multi-Dimensional Dynamic Logic [28] that allows multi-agent [17] representation, Dynamic Arrow Logic [32] to deal with transitions in programs, Data Analysis Logic [7], Boolean Modal Logic [10], logics for reasoning about knowledge [8], logics for knowledge representation [19] and Dynamic Description Logic [33].

Petri Net is a widely used formalism to specify and to analyse concurrent programs with a very nice graphical representation. It allows for representing true concurrency and parallelism in neat way.

We present the logic Petri-PDL (detailed in Section 3) that replaces the conventional PDL programs by Marked Petri Net programs. So if π is a Petri Net program with markup s, then the formula $\langle s, \pi \rangle \varphi$ means that after running this program with the initial markup s, φ will eventually be true (also possible a $\Box$-like modality replacing the tags by brackets as an abbreviation for $\neg\langle s, \pi \rangle \neg\varphi$).

Our paper falls in the broad category of works that attempt to generalize PDL and build dynamic logics that deal with classes of non-regular programs. As examples of other works in this area, we can mention [13], [12] and [20], that develop decidable dynamic logics for fragments of the class of context-free programs and [1, 11, 26, 27] and [3], that develop dynamic logics for classes of programs with some sort of concurrency. Our logics have a close relation to two logics in this last group: Concurrent PDL [26] and Concurrent PDL with Channels [27]. Both of these logics are expressive enough to represent interesting properties of communicating concurrent systems. However, neither of them has a simple Kripke semantics. The first has a semantics based on *super-states* and *super-processes* and its satisfiability problem can be proved undecidable (in fact, it is Π^1_1-hard). Also, it does not have a complete axiomatization [27]. On the other hand, our logics have a simple Kripke semantics, simple and complete axiomatizations and the finite model property.

There are other approaches that use Dynamic Logic to reasoning about specifications of concurrent systems represented as Petri Nets [15, 16, 31]. They differ from our approach by the fact that they use Dynamic logic as an specification language for representing Petri Net, they do not encode Petri Nets as programs of a Dynamic Logic. They translate Nets into PDL language while we have a new Dynamic Logic tailored to reasoning about Petri Nets in a more natural way.

This article is organized as follows. Section 2.2 presents the (Marked) Petri Net formalism used in this work. Section 3 introduces our dynamic logic, with its language and semantics and also proposes an axiomatization. Section 4, provides a prove of soundness and completeness of our axiomatization. Finally, Section 5 illustrates the use of our logic with some examples.

2 Background

This section presents a brief overview of two topics on which the later development is based on. First, we make a brief review of the syntax and semantics of PDL [14]. Second, we present the Petri Nets formalism and its variant, Marked Petri Nets. Finally, the compositional approach introduced in [6] is briefly discussed.

2.1 Propositional Dynamic Logic

In this section, we present the syntax and semantics of the most used dynamic logic called PDL for regular programs.

DEFINITION 2.1
The PDL language consists of a set Φ of countably many proposition symbols, a set Π of countably many basic programs, the Boolean connectives $\neg$ and $\wedge$, the program constructors; (sequential composition), $\cup$ (non-deterministic choice) and * (iteration) and a modality $\langle\pi\rangle$ for every program π. The formulas are defined as follows:

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\pi\rangle\varphi, \text{ with } \pi ::= a \mid \pi_1;\pi_2 \mid \pi_1 \cup \pi_2 \mid \pi^\star,$$

where $p \in \Phi$ and $a \in \Pi$.

In all the logics that appear in this article, we use the standard abbreviations $\bot \equiv \neg\top$, $\varphi \vee \phi \equiv \neg(\neg\varphi \wedge \neg\phi)$, $\varphi \rightarrow \phi \equiv \neg(\varphi \wedge \neg\phi)$ and $[\pi]\varphi \equiv \neg\langle\pi\rangle\neg\varphi$.

Each program π corresponds to a modality $\langle\pi\rangle$, where a formula $\langle\pi\rangle\alpha$ means that after the running of π, α eventually is true, considering that π halts. There is also the possibility of using $[\pi]\alpha$ (as an abbreviation for $\neg\langle\pi\rangle\neg\alpha$) indicating that the property denoted by α holds after every possible running of π.

The semantics of PDL is normally, given using a transition diagram, which consists of a set o states and binary relations (one for each program) indicating the possible execution of each program at each state. In PDL literature a transition diagram is called a frame.

DEFINITION 2.2
A *frame* for PDL is a tuple $\mathcal{F} = \langle W, R_\pi\rangle$ where

- W is a non-empty set of states;
- R_a is a binary relation over W, for each basic program $a \in \Pi$;
- We can inductively define a binary relation R_π, for each non-basic program π, as follows
  - $R_{\pi_1;\pi_2} = R_{\pi_1} \circ R_{\pi_2}$,
  - $R_{\pi_1 \cup \pi_2} = R_{\pi_1} \cup R_{\pi_2}$,
  - $R_{\pi^\star} = R_\pi^\star$, where $R_\pi^\star$ denotes the reflexive transitive closure of R_π.

DEFINITION 2.3
A *model* for PDL is a pair $\mathcal{M} = \langle\mathcal{F}, \mathbf{V}\rangle$, where $\mathcal{F}$ is a Petri-PDL frame and $\mathbf{V}$ is a valuation function $\mathbf{V} : \Phi \rightarrow 2^W$.

The semantical notion of satisfaction for PDL is defined as follows:

DEFINITION 2.4
Let $\mathcal{M} = \langle\mathcal{F}, \mathbf{V}\rangle$ be a model. The notion of *satisfaction* of a formula φ in a model $\mathcal{M}$ at a state w, notation $\mathcal{M}, w \Vdash \varphi$, can be inductively defined as follows:

- $\mathcal{M}, w \Vdash p$ iff $w \in \mathbf{V}(p)$;
- $\mathcal{M}, w \Vdash \top$ always;
- $\mathcal{M}, w \Vdash \neg\varphi$ iff $\mathcal{M}, w \nVdash \varphi$;
- $\mathcal{M}, w \Vdash \varphi_1 \wedge \varphi_2$ iff $\mathcal{M}, w \Vdash \varphi_1$ and $\mathcal{M}, w \Vdash \varphi_2$;
- $\mathcal{M}, w \Vdash \langle\pi\rangle\varphi$ iff there is $w' \in W$ such that $wR_\pi w'$ and $\mathcal{M}, w' \Vdash \varphi$.

For more details on PDL see [14].

2.2 Petri Nets

A Petri Net [29] is a tuple $\mathcal{P} = \langle S, T, W\rangle$, where S is a finite non-empty set of places, T is a finite set of transitions where S and T are disjoint and W is a function which defines directed edges between

places and transitions and assigns a $w \in \mathbb{N}$ that represents a multiplicative weight for the transition, as $W : (S \times T) \cup (T \times S) \to \mathbb{N}$.

2.2.1 Marked Petri Nets

A markup function M is a function that assigns to each place a natural number, $M : S \to \mathbb{N}$. A Marked Petri Net is a tuple $\mathcal{P} = \langle S, T, W, M_0 \rangle$ where $\langle S, T, W \rangle$ is a Petri Net and M_0 as an initial markup. In the sequel, any reference to a Petri-Net means Marked Petri-Nets.

The flow of a Petri Net is defined by a relation $F = \{(x,y) \mid W(x,y) > 0\}$; in this work for all transitions $W(x,y) = 1$. Let $s \in S$ and $t \in T$. The preset of t, denoted by ${}^\bullet t$, is defined as ${}^\bullet t = \{s \in S : (s,t) \in F\}$; the postset of t, denoted by $t^\bullet$ is defined as $t^\bullet = \{s \in S : (t,s) \in F\}$. The preset of s, denoted by ${}^\bullet s$, is defined as ${}^\bullet s = \{t \in T : (t,s) \in F\}$; the postset of s, denoted by $s^\bullet$ is defined as $s^\bullet = \{t \in T : (s,t) \in F\}$.

Given a markup M of a Petri Net, we say that a transition t is enabled on M if and only if $\forall x \in {}^\bullet t, M(x) \geq 1$. A new markup generated by setting a transition that is enabled is defined as

$$M_{i+1}(x) = \begin{cases} M_i(x) - 1, & \forall x \in {}^\bullet t \setminus t^\bullet \\ M_i(x) + 1, & \forall x \in t^\bullet \cap {}^\bullet t \\ M_i(x), & \forall x \notin \{({}^\bullet t - t^\bullet) \cup (t^\bullet + {}^\bullet t)\} \end{cases} . \tag{1}$$

A program behaviour is described by the set $M = \{M_1, \ldots, M_n\}$ of a Petri Net markups.

A Petri Net may be interpretated in a graphical representation, using a circle to represent each $s \in S$, a rectangle to represent each $t \in T$, the relations defined by W as edges between places and transitions. The amount of tokens from M are represented as filled circles into the correspondent places. An example of a valid Petri Net is in Figure 1.

Just as an example, the Petri Net on Figure 2 represents the operation of an elevator for a building with five floors. A token in the place U indicates that the elevator is able to go up one floor; and, when T_1 fires, a token goes to D, so the elevator can go down a floor. If the elevator goes down a floor (i.e. T_2 fires) a token goes to the place U. Figure 2 illustrates the Petri Net with its initial markup.

Another example is in Figure 3, which represents a SMS send and receive of two cellphones. When the user sends a SMS from his cellphone, it goes to his phone buffer (i.e. T_1 fires and the token goes to p_2). When the phone sends the message to the operator (i.e. T_2 fires) it goes to the

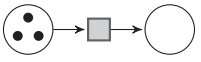

FIG. 1. Example of a valid Petri Net.

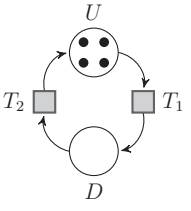

FIG. 2. Petri Net for a simple elevator of five floors.

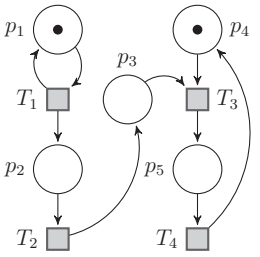

FIG. 3. Petri Net for a SMS send and receive.

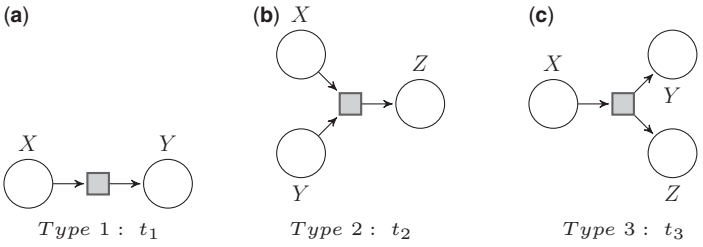

FIG. 4. Basic Petri Nets.

operator buffer; so, the messages must be sent to the receiver, but the receiver is able to receive only one message at a time. If there is a message in the operator buffer and the receiver is not receiving other message (i.e. there is at least a token in p_3 and there is a token in p_4), the receiver can receive the message (i.e. T_3 fires). At this point the user cannot receive other messages (i.e. there is no token in p_4, so T_3 is not enabled); but, after the complete receipt of the message (i.e. T_4 fires), the receiver is able to receive messages again (i.e. there is a token in p_4 and when p_3 have at least a token, T_3 will be enabled again).

2.2.2 Basic Petri Nets

The Petri Net model used in this work is as defined by de Almeida and Haeusler [6]. It uses three basic Petri Nets to define all valid Petri Nets due to its compositions. These basic Petri Nets are as in Figure 4.

To compose more complex Petri Nets from these three basic ones, it is used a gluing procedure described, and proved corrected in [6].

As an example, taking the Petri Net in Figure 2 it can be modelled as a composition of two Petri Nets of type 1, where UT_1D is composed with DT_2U, generating the Petri Net $UT_1D \odot DT_2U$ where $\odot$ denotes the Petri Net composition symbol. The Petri Net from Figure 3 can be modelled by composition of Petri Nets of the three basic types. The basic Petri Nets of this case are $p_2T_2p_3$, $p_5T_4p_4$ (type 1), $p_3p_4T_3p_5$ (type 2) and $p_1T_1p_1P_2$ (type 3), composing the Petri Net $p_2T_2p_3 \odot p_5T_4p_4 \odot p_3p_4T_3p_5 \odot p_1T_1p_1P_2$.

3 PDL for Petri Nets (Petri-PDL)

This section presents a PDL that uses Petri Nets terms as programs (Petri-PDL).

3.1 Language and semantics

The PDL language consists of a set Φ of countably many propositional symbols and

Place names: e.g.: $a, b, c, d \ldots$
Transition names: e.g.: $t_1, t_2, t_3 \ldots$, each transition name has a unique type;
Transition types: $T_1 : xt_1y$, $T_2 : xyt_2z$ and $T_3 : xt_3yz$
Petri Net Composition symbol: $\odot$
Sequence of names: $S = \{\epsilon, s_1, s_2, \ldots\}$, where ϵ is the empty sequence. We use the notation $a \in s$ to denote that name a occurs in s. Let $\#(s, a)$ be the number of occurrences of name a in s.

Next we define the programs that represent Petri Nets.

DEFINITION 3.1
Programs:

Basic programs: $\pi_b ::= at_1b \mid at_2bc \mid abt_3c$ where t_i is of type $T_i, i = 1, 2, 3$
Petri Net Programs: $\pi_{PN} ::= \pi_b \mid \pi_{PN} \odot \pi_{PN}$, denoted by $\eta_1, \eta_2, \ldots$
Marked Petri Net Programs: $\pi_{MPN} ::= s, \pi_{PN}$

DEFINITION 3.2
A formula is defined as

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle \pi_{MPN} \rangle \varphi.$$

$\Leftarrow M$
where $p \in \Phi$.

We use the standard abbreviations $\bot \equiv \neg\top$, $\varphi \vee \phi \equiv \neg(\neg\varphi \wedge \neg\phi)$, $\varphi \rightarrow \phi \equiv \neg(\varphi \wedge \neg\phi)$ and $[\pi]\varphi \equiv \neg\langle\pi\rangle\neg\varphi$.

The following definition introduces the *firing* function. It defines how the marking of a basic Petri Nets changes after a firing.

DEFINITION 3.3
We define the *firing* function $f : S \times \pi_b \rightarrow S$ as follows

- $f(s, at_1b) = \begin{cases} s_1bs_2, & \text{if} \quad s = s_1as_2 \\ \epsilon, & \text{if} \quad a \notin s \end{cases}$
- $f(s, abt_2c) = \begin{cases} s_1cs_2s_3, & \text{if} \quad s = s_1as_2bs_3 \\ \epsilon, & \text{if} \quad a \notin s \text{ or } b \notin s \end{cases}$
- $f(s, at_3bc) = \begin{cases} s_1s_2bc, & \text{if} \quad s = s_1as_2 \\ \epsilon, & \text{if} \quad a \notin s \end{cases}$
- $f(\epsilon, \eta) = \epsilon$, for all petri nets programs η.

The definitions the follow of frame, model and satisfaction are similar to the one presented in Section 2.1 for PDL. Now, we have to adapt them to deal with the firing of basic Petri Nets.

DEFINITION 3.4
A frame for Petri-PDL is a tuple $\langle W, R_\pi, M \rangle$, where

- W is a non-empty set of states;
- $M : W \rightarrow S$;
- R_{π_b} is a binary relation over W, for each basic program π_b, satisfying the following condition. Let $s = M(w)$

  - if $f(s,\pi_b)\neq\epsilon$, $wR_{\pi_b}v$ iff $f(s,\pi_b)\in M(v)$
  - if $f(s,\pi_b)=\epsilon$, $wR_{\pi_b}v$ iff $w=v$
- we inductively define a binary relation R_η, for each Petri Net program $\eta=\eta_1\odot\eta_2\odot\cdots\odot\eta_n$, as

$$R_\eta=\{(w,v)\mid \text{ for some } \eta_i,\exists u \text{ such that } s_i\in M(u) \text{ and } wR_{\eta_i}u \text{ and } uR_\eta v\}$$

  Where $s_i=f(s,\eta_i)$, for all $1\leq i\leq n$.

Definition 3.5
A *model* for Petri-PDL is a pair $\mathcal{M}=\langle\mathcal{F},\mathbf{V}\rangle$, where $\mathcal{F}$ is a PDL frame and $\mathbf{V}$ is a valuation function $\mathbf{V}\colon\Phi\to 2^W$.

The semantical notion of satisfaction for Petri-PDL is defined as follows.

Definition 3.6
Let $\mathcal{M}=(\mathcal{F},\mathbf{V})$ be a model. The notion of *satisfaction* of a formula φ in a model $\mathcal{M}$ at a state w, notation $\mathcal{M},w\Vdash\varphi$, can be inductively defined as follows:

- $\mathcal{M},w\Vdash p$ iff $w\in\mathbf{V}(p)$;
- $\mathcal{M},w\Vdash\top$ always;
- $\mathcal{M},w\Vdash\neg\varphi$ 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 $v\in W$, $wR_\eta v$, $s\in M(w)$ and $\mathcal{M},v\Vdash\varphi$.

If $\mathcal{M},v\Vdash A$ for every state v, we say that A is *valid in the model* $\mathcal{M}$, notation $\mathcal{M}\Vdash A$. And if A is valid in all $\mathcal{M}$ we say that A is *valid*, notation $\Vdash A$.

3.2 Axiomatic system

We consider the following set of axioms and rules, where p and q are proposition symbols, φ and ψ are formulas, $\eta=\eta_1\odot\eta_2\odot\cdots\odot\eta_n$ is a Petri Net program and π is a Marked Petri Net program.

(PL) Enough propositional logic tautologies
(K) $[s,\eta](p\to q)\to([s,\eta]p\to[s,\eta]q)$
(Du) $[s,\eta]p\leftrightarrow\neg\langle s,\eta\rangle\neg p$
(PC) $\langle s,\eta\rangle\varphi\leftrightarrow\langle s,\eta_1\rangle\langle s_1,\eta\rangle\varphi\vee\langle s,\eta_2\rangle\langle s_2,\eta\rangle\varphi\vee\cdots\vee\langle s,\eta_n\rangle\langle s_n,\eta\rangle\varphi$,
where $s_i=f(s,\eta_i)$, for all $1\leq i\leq n$.
(R_ϵ) $\langle s,\eta\rangle\varphi\leftrightarrow\varphi$, if $f(s,\eta)=\epsilon$
(Sub) If $\vdash\varphi$, then $\vdash\varphi^\sigma$, where σ uniformly substitutes proposition symbols by arbitrary formulas.
(MP) If $\vdash\varphi$ and $\vdash\varphi\to\psi$, then $\vdash\psi$.
(Gen) If $\vdash\varphi$, then $\vdash[\eta]\varphi$.

4 Soundness and completeness

The axioms **(PL)**, **(K)** and **(Du)** and the rules **(Sub)**, **(MP)** and **(Gen)** are standard in the modal logic literature. We prove the validity only for axioms (PC) and and (R_ϵ).

Lemma 4.1
Validity of Petri-PDL axioms

1. $\Vdash$ **PC**

PROOF. Suppose that there is a world w from a model $\mathcal{M} = \langle W, R_\pi, \mathbf{V}, M\rangle$ where PC is false. For PC to be false in w, there are two cases:

(a) Suppose $\mathcal{M}, w \Vdash \langle s, \eta\rangle\varphi$ and
$\mathcal{M}, w \nVdash \langle s, \eta_1\rangle\langle s_1, \eta\rangle\varphi \vee \langle s, \eta_2\rangle\langle s_2, \eta\rangle\varphi \vee \cdots \vee \langle s, \eta_n\rangle\langle s_n, \eta\rangle\varphi$
$\mathcal{M}, w \Vdash \langle s, \eta\rangle\varphi$ iff there is a v such that $wR_\eta v$, $s \in M(w)$ and $\mathcal{M}, v \Vdash \varphi$.
By definition 3.4 $R_\eta = \{(w, v) \mid$ *for some* $\eta_i, \exists u$ *such that* $s_i \in M(u)$ *and* $wR_{\eta_i}u$ *and* $uR_\eta v\}$,
so $\mathcal{M}, u \Vdash \langle s_i, \eta\rangle\varphi$ and $\mathcal{M}, w \Vdash \langle s, \eta_i\rangle\langle s_i, \eta\rangle\varphi$. This implies
$\mathcal{M}, w \Vdash \langle s, \eta_1\rangle\langle s_1, \eta\rangle\varphi \vee \langle s, \eta_2\rangle\langle s_2, \eta\rangle\varphi \vee \cdots \vee \langle s, \eta_n\rangle\langle s_n, \eta\rangle\varphi$, which contradicts
$\mathcal{M}, w \nVdash \langle s, \eta_1\rangle\langle s_1, \eta\rangle\varphi \vee \langle s, \eta_2\rangle\langle s_2, \eta\rangle\varphi \vee \cdots \vee \langle s, \eta_n\rangle\langle s_n, \eta\rangle\varphi$.

(b) Suppose $\mathcal{M}, w \nVdash \langle s, \eta\rangle\varphi$ and
$\mathcal{M}, w \Vdash \langle s, \eta_1\rangle\langle s_1, \eta\rangle\varphi \vee \langle s, \eta_2\rangle\langle s_2, \eta\rangle\varphi \vee \cdots \vee \langle s, \eta_n\rangle\langle s_n, \eta\rangle\varphi$, iff for some i $(1 \leq i \leq n)$, $M, w \Vdash \langle s, \eta_i\rangle\langle s_i, \eta\rangle\varphi$ iff
there is a u such that $wR_{\eta_i}u$, $s \in M(w)$ and $\mathcal{M}, u \Vdash \langle s_i, \eta\rangle\varphi$,
iff there is a v such that $uR_\eta v$, $s_i \in M(u)$ and $\mathcal{M}, v \Vdash \varphi$,
By Definition 3.4, and the above we have $wR_\eta v$ and $s \in M(w)$ and $M, v \Vdash \varphi$. Thus, $\mathcal{M}, w \Vdash \langle s, \eta\rangle\varphi$ which is a contradiction.

∎

2. $\Vdash$ $\mathbf{R}_\epsilon$

PROOF. Suppose that there is a w from a model $\mathcal{M} = \langle W, R_\pi, \mathbf{V}, M\rangle$ where R_ϵ is false. For R_ϵ be false in w, there are two cases:

(a) Suppose $\mathcal{M}, w \Vdash \langle \epsilon, \eta\rangle\varphi$ and
$\mathcal{M}, w \nVdash \varphi$
$\mathcal{M}, w \Vdash \langle \epsilon, \eta\rangle\varphi$ iff there is a v such that $wR_\eta v$ and $\mathcal{M}, v \Vdash \varphi$. As $f(\epsilon, \eta) = \epsilon$, then $w = v$ and $wR_\eta w$ and $\mathcal{M}, w \Vdash \varphi$, which contradicts $\mathcal{M}, w \nVdash \varphi$.

(b) Suppose $\mathcal{M}, w \nVdash \langle \epsilon, \eta\rangle\varphi$ and
$\mathcal{M}, w \Vdash \varphi$.
$\mathcal{M}, w \nVdash \langle \epsilon, \eta\rangle\varphi$ iff for all v such that, if $wR_\eta v$ then $\mathcal{M}, v \Vdash \neg\varphi$. As $f(\epsilon, \eta) = \epsilon$, then $w = v$ and $wR_\eta w$ and $\mathcal{M}, w \Vdash \neg\varphi$, which contradicts $\mathcal{M}, w \Vdash \varphi$.

∎

As pointed out by the work of Mazurkiewicz [22, 23], logics that deal with Petri Nets use to be incomplete due to the possibility of a place always increase its token amount (up to countable infinity). To restrict a subset of Petri Nets where we can achieve completeness, we denote by normalized Petri Net any Petri Net composed only by subnets of the three basic types above defined and that do not contain any place which can accumulate an infinity amount of tokens. From now on, all the proofs deal only with normalized Petri Nets.

The completeness proofs are similar to the ones presented in the work of Blackburn et al. [4], Harel et al. [14] and Goldblatt [11]. We define a filtration procedure that collapses every equivalent world, then defines a canonical model (a model where the set of worlds is the set of all maximal consistent sets) and finally shows that every consistent formula is satisfiable in a canonical model.

DEFINITION 4.2
The Fischer–Ladner Closure

It is inductively defined as follows, where $FL(\varphi)$ denotes the smallest set containing φ which is closed under sub formulae.

$FL : \Upsilon \to 2^{\Upsilon}$, where Υ is the set of all formulae

1. $FL(\varphi)$ is closed under subformulae;
2. if $\langle s,\eta\rangle\psi \in FL(\varphi)$, then $\langle s,\eta_i\rangle\langle s_i,\eta\rangle\psi \in FL(\varphi)$, where $\eta = \eta_1 \odot \eta_2 \odot \cdots \odot \eta_n$ and $s_i = f(s,\eta_i)$, for all $1 \le i \le n$.

LEMMA 4.3
Let $\eta = \eta_1 \odot \eta_2 \odot \cdots \odot \eta_n$ be a composed Petri Net program where each η_i, $1 \le i \le n$, is a basic Petri Net program and s is a sequence of names. For every sequence $(s_0,\eta) \to (s_1,\eta) \to \cdots \to (s_k,\eta)$, where $s_0 = s$, $k \ge 0$ and $s_j = f(s,\eta_i)$ for some $1 \le i \le n$, then either $s_k = \epsilon$ or one of $s_j = s_\ell$ for $0 < j \le k$, $0 < \ell \le k$ and $j \ne \ell$.

PROOF. As η' has no places which accumulates tokens infinitely, after firing all the basic Petri Net programs of η' continuously there are two possibilities for the markup of η':

1. there is no transition able to fire, so $s_k = \epsilon$;
2. some markup m of η' activated a loop in the Petri Net (e.g. η' as a graph is cyclic), so after a non empty serie of fires the markup m of η' will appear again, so $s_j = s_\ell$ for some $0 < j \le k$ and some $0 < \ell \le k$ such that $j \ne \ell$.

∎

LEMMA 4.4
$FL(\varphi)$ is finite.

PROOF. The only possibility of construct $\varphi \succ \sigma$ (i.e. σ is a derivative of the formula φ) is iff φ is in the form $\langle s,\pi\rangle\Psi$ and σ is in the form $\langle s,\pi_i\rangle\langle s_i,\pi\rangle\Psi$ for some $1 \le i \le n$ where n is the size of the Petri Net program π (i.e. the number of atomic programs of π) and π_i is an atomic program. Then the smallest closed set Γ containing a formula ρ is obtained by closing $FL(\rho)$ under $\succ$; hence $\phi \in \Gamma$ iff there is a finite sequence of the form $\varphi = \varphi_1 \succ \cdots \succ \varphi_j = \phi$, where $\forall_{m \ne n} \varphi_m \ne \varphi_n$ and $\varphi \in FL(\rho)$. So, if $\langle s,\kappa\rangle\Psi \succ \langle p,\tau\rangle\phi$ for κ a normalized Petri Net program, then τ is an atomic Petri Net program or is equal to κ. Therefore that can be no infinitely-long $\succ$-sequences.
So, $FL(\varphi)$ is finite. ∎

LEMMA 4.5
(i) If $\sigma \in FL(\varphi)$, then $FL(\sigma) \subseteq FL(\varphi)$
(ii) If $\sigma \in FL(\langle s,\pi_{MPN}\rangle\varphi)$, then $FL(\sigma) \subseteq FL(\langle s,\pi_{MPN}\rangle\varphi) \cup FL(\varphi)$

PROOF.

(i) If $\sigma \in FL(\varphi)$, then as by 1. in definition 4.2 $FL(\varphi)$ is closed under subformulae, then $FL(\sigma) \subseteq FL(\varphi)$.
(ii) If $\sigma \in FL(\langle s,\pi_{MPN}\rangle\varphi)$, then by 2. in definition 4.2 $FL(\langle s,\pi_{MPN}\rangle\varphi)$ is closed under subformulae, so $FL(\sigma) \subseteq FL(\langle s,\pi_{MPN}\rangle\varphi) \cup FL(\varphi)$.

∎

DEFINITION 4.6
Filtration
Given a Petri-PDL formula φ, a Petri-PDL model $\mathcal{K} = \langle W, R_\eta, M, \mathbf{V}\rangle$, we define a new model

$$\mathcal{K}^\varphi = \langle W^\varphi, R_\eta^\varphi, M^\varphi, \mathbf{V}^\varphi\rangle,$$

the filtration of $\mathcal{K}$ by $FL(\varphi)$, as follows.

The relation $\equiv$ over the worlds of $\mathcal{K}$ is defined as

$$u \equiv v \leftrightarrow \forall \phi \in FL(\varphi), M, u \Vdash \phi \text{ iff } M, v \Vdash \phi$$

and the relation R_η^φ is defined as

$$[u] R_\eta^\varphi [v] \leftrightarrow (\exists u' \in [u] \wedge \exists v' \in [v] \wedge u' R_\eta v').$$

1. $[u] = \{v \mid v \equiv u\}$
2. $W^\varphi = \{[u] \mid u \in W\}$
3. $[u] \in \mathbf{V}^\varphi(p)$ iff $u \in V(p)$
4. $M^\varphi([u]) = \langle s_1, s_2, \ldots \rangle$ where for all $j \geq 1, v_j \in [u]$ and $M_{\mathcal{K}}(v_j) = s_j$

All rules may be composed inductively to extend in order to compound all programs and propositions due to compositions as in Section 3.

LEMMA 4.7
Filtration Lemma
$\forall u,\ v \in W,\ uR_\eta v$ iff $[u] R_\eta^\varphi [v]$

PROOF. The proof is straightforward from Definition 4.6. ∎

LEMMA 4.8
$\mathcal{K}^\varphi$ is a finite Petri-PDL model.

PROOF.

- W^φ is finite a set of states by Definition 4.6 and Lemma 4.4.
- $M^\varphi : W^\varphi \to S$ by Definition 4.6.
- $R_\eta^\varphi = \{([w], [v]) \mid$ for some η_i $\exists [u]$ such that $s_i \in M^\varphi([u])$ and $[w] R_{\eta_i} [u]$ and $[u] R_\eta [v]\}$ for any program $\eta = \eta_1 \odot \cdots \odot \eta_n$, where $s_i = f(s, \eta_i)$ and $1 \leq i \leq n$.
- $\mathbf{V}^\varphi : \Phi \to 2^{W^\varphi}$ by Definition 4.6.

Then $\mathcal{K}^\varphi$ is a finite Petri-PDL model. ∎

COROLLARY 4.9
Petri-PDL is decidable

PROOF. As, by Lemma 4.8, the number of states is finite, there is an exponential-time algorithm to check whether a formula φ of Petri-PDL is satisfiable. ∎

DEFINITION 4.10
Canonic Model

A canonical model for Petri-PDL with language Λ is a 4-tupple $\mathcal{C}^\Lambda = \langle W^\Lambda, R^\Lambda, M^\Lambda, \mathbf{V}^\Lambda \rangle$, where W^Λ is the set of all maximal consistent sets; $\mathbf{V}^\Lambda$ is a valuation function where for all $w \in W^\Lambda$, $w \in \mathbf{V}^\Lambda(\varphi)$ iff $\varphi \in w$; M^Λ is the markup of the Petri Net programs, defined as

$$M^\Lambda(w) = \{s_1, \ldots, s_n \mid \langle s_i, \pi \rangle \varphi \in w, 1 \leq i \leq n, w \in W^\Lambda\};$$

and R^Λ is a binary relation between the elements of W^Λ defined for each program π as

$$R_\pi^\Lambda = \{(n, m) \mid n, m \in W^\Lambda, \{\varphi / [s, \pi] \varphi \in n, s \in M^\Lambda(n)\} \subseteq m\}.$$

LEMMA 4.11
$\mathcal{C}^{\Lambda}$ is a Petri-PDL model.

PROOF. By Definition 4.10:

- W^{Λ} is finite a set of states.
- $M^{\Lambda}: W^{\Lambda} \to S$.
- $R^{\Lambda}_{\pi} = \{(w,v) \mid$ for some $\eta_i \exists u$ such that $s_i \in M^{\Lambda}(u)$ and $wR_{\eta_i}u$ and $uR_{\eta}v\}$ for any program $\eta = \eta_1 \odot \cdots \odot \eta_n$, where $s_i = f(s,\eta_i)$ and $1 \leq i \leq n$.
- $\mathbf{V}^{\Lambda}: \Phi \to 2^{W^{\Lambda}}$. ∎

LEMMA 4.12
$[s,\pi]\varphi \in u$ iff in all v such that $uR^{\Lambda}v$, $\varphi \in v$.

PROOF.
Supose $[s,\pi]\varphi \in u$ and there is no $v \in W^{\Lambda}$ such that $uR^{\Lambda}v$ and $\varphi \in v$ (1).
As $\pi = \pi_1 \odot \cdots \odot \pi_n$, by R^{Λ} we have that all $[s,\pi_i][s_i,\pi]\varphi, 1 \leq i \leq n \in v$ for all $1 \leq i \leq n$ if $uR^{\Lambda}v$ (2).
By (PC), all $[s,\pi_i][s_i,\pi]\varphi, 1 \leq i \leq n \in u$ for all $1 \leq i \leq n$ (3).
But if (3) then φ is in some v such that $uR^{\Lambda}v$ by R^{Λ} definition, which contradicts (1).
So, in all v such that $uR^{\Lambda}v$, $\varphi \in v$.
Supose that exists some $u \in W^{\Lambda}$ where $[s,\pi]\varphi \notin u$ and such that in all v that $uR^{\Lambda}v$, $\varphi \in v$ (4).
By R^{Λ} definition, if in all v that $uR^{\Lambda}v$, $\varphi \in v$, then $[s,\pi]\varphi \in u$ (5).
Then, there is a contradiction.
So, this lemma is valid. ∎

LEMMA 4.13
Consider $\mathcal{C}^{\Lambda}$ as in Definition 4.10. Then, for any $w \in W^{\Lambda}$, $w \Vdash \varphi$ iff $\varphi \in w$.

PROOF.

1. If φ is an atomic formula, it holds by the definition of $\mathbf{V}^{\Lambda}$.
2. If φ is $\neg\phi$ then $w \Vdash \varphi$ iff $w \nVdash \phi$.
3. If φ is in the form $\phi_1 \wedge \phi_2$ then, as w is maximal consistent and by the inductive hypothesis, $w \Vdash \varphi$ iff $w \Vdash \phi_1$ and $w \Vdash \phi_2$, and $\phi_1 \in w$ and $\phi_2 \in w$.
4. If φ is a modal formula with a Petri Net program such as $[s,\pi]\phi$, then, $w \Vdash \varphi$ iff:
   ($\mathbf{R}_{\epsilon}$): $f(s,\pi) = \epsilon$, $wR^{\Lambda}_{\pi}w$, $w \Vdash \phi$ and $\phi \in w$ by the inductive hypothesis;
   (PC): $\exists u \exists v, wR^{\Lambda}_{\pi_i}u$, $uR^{\Lambda}_{\pi}v$, $1 \leq i \leq n$, $v \Vdash \phi$ and $\phi \in v$ by the inductive hypothesis and by Lemma 4.12, where $\pi = \pi_1 \odot \cdots \odot \pi_n$. ∎

THEOREM 4.14 (Completeness)
Petri-PDL is complete w.r.t the class of Petri-PDL models.

PROOF. Suppose φ is consistent, then it is contained in a maximal consistent set w, which is a state of $\mathcal{C}^{\Lambda}$. By Lemma 4.13 we have $w \Vdash \varphi$. And by Lemma 4.11 we know that $\mathcal{C}^{\Lambda}$ is a Petri-PDL model. Therefore, every consistent formula is satisfiable in a Petri-PDL model. ∎

5 Some usage examples

In this section we will present two examples of applications.

The execution of (b) in the Petri Net defined in Figure 5 does not imply that (c) is achieved: $\langle (b), abt_2c\rangle \top \to \neg\langle (c), abt_2c\rangle \top$.

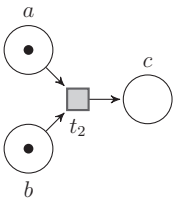

FIG. 5. A Petri Net where a and b have one token each.

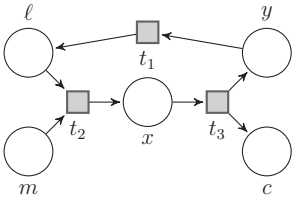

FIG. 6. A Petri Net for a chocolate vending machine.

A more abstract example may be composed as: looking at the Petri Net defined in the Figure 6, the upper left place (ℓ) is the power button of a vending machine; the bottom left is the coin inserted (m) and the bottom right is the chocolate output (c); if the vending machine is powered on, always when a coin is inserted you will have a chocolate as an output: $\langle(\ell,m),\ell mt_2x\odot xt_3yc\odot yt_1\ell\rangle\top\rightarrow\langle(\ell,c),\ell mt_2x\odot xt_3yc\odot yt_1\ell\rangle\top$.

In the Petri Net example in Figure 2, it is possible to say that, in the initial mark, it is not possible that the elevator goes down a floor. That is, $\langle(U,U,U,U),Ut_1D\odot Dt_1U\rangle\top\rightarrow\neg\langle(U,U,U,U,U),Ut_1D\odot Dt_1U\rangle\top$; and that it the elevator goes up a floor it can go down too, as in $\langle(U,U,U,U),Ut_1D\odot Dt_1U\rangle\top\rightarrow\langle(U,U,U,D),Ut_1D\odot Dt_1U\rangle\top$.

Concerning the Petri Net example in Figure 3, we can say that when a message is being received, it is not possible to receive another at the same time: $\langle(p_1,p_3,p_4),p_1t_3p_1p_2\odot p_2t_1p_3\odot p_3p_4t_2p_5\odot p_5t_1p_4\rangle\top\rightarrow\neg\langle(p_3,p_4),p_1t_3p_1p_2\odot p_2t_1p_3\odot p_3p_4t_2p_5\odot p_5t_1p_4\rangle\top$.

Another case study in which the Petri Nets can be applied to model is the 'Rock-Paper-Scissors' game. It is presented in Figure 7 where, once a player inserts a coin in a supposed machine (i.e. the place 'Coin' has a token), a transition may fire and the place 'Player_1' (the gamer) and 'Player_2' (the machine bot) will have a token. Now the player can select one of the options at the same time that the machine can choose one. If the user wins, a token will be placed at 'Win_1' and the user will be able to play again; if he loses, at the place 'Win_2' or the game restarts if there is a draw match.

Consider the Petri Net represented in Figure 8a. The upper left place (H) is the handle to open a door to the next stage in the actual game level; the bottom left is the key (K) used by the player in the previous level and the bottom right is the door opening (O); if the key is in the lock, always when the handle is lowered the door for the next stage is open. A token in place x means that the door is unlocked and in y the player is handling the door. The formula $\langle(HKKK),HKt_2x\odot xt_3yO\odot yt_1H\rangle\varphi$ models this actual state of the Petri Net program in Figure 8a where after the Petri Net program stops φ will be true in a future state. Supposing that $\langle(HKKK),HKt_2x\odot xt_3yO\odot yt_1H\rangle\varphi$ is true, the only enabled transition is t_2, so we have that $\langle(HKKK),HKt_2x\rangle\langle(KKx),HKt_2x\odot xt_3yO\odot yt_1H\rangle\varphi$ is true; then t_2 fires and we have that $\langle(KKx),HKt_2x\odot xt_3yO\odot yt_1H\rangle\varphi$ is true. Now the only enabled transition is t_3, so we have that $\langle(KKx),xt_3yO\rangle\langle(KKOy),HKt_2x\odot xt_3yO\odot yt_1H\rangle\varphi$ will be true and,

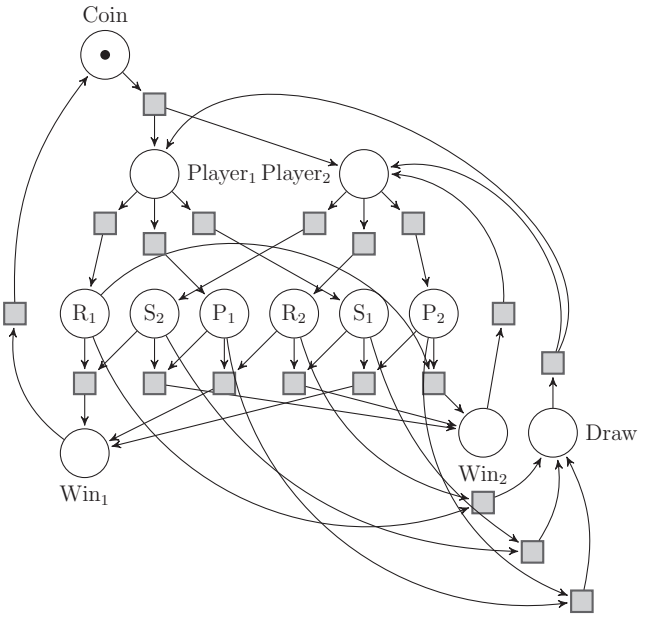

FIG. 7. Petri Net for 'Rock-Paper-Sicissors' game.

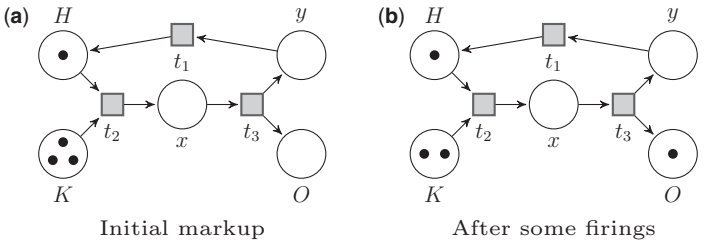

FIG. 8. A Petri Net for a game where doors separate stages.

after the firing of t_3, the formula $\langle(KKOy), HKt_2x \odot xt_3yO \odot yt_1H\rangle\varphi$ will be true. Finally t_1 is the only enabled transition. Hence we can now verify that $\langle(KKOy), yt_1H\rangle\langle(KKOH), HKt_2x \odot xt_3yO \odot yt_1H\rangle\varphi$ will be true where after the firing of t_1 will achieve the markup of Figure 8b and $\langle(KKOH), HKt_2x \odot xt_3yO \odot yt_1H\rangle\varphi$ will be true. So we can verify that if $\langle(HKKO), HKt_2x \odot xt_3yO \odot yt_1H\rangle\varphi$ is true in an actual state, $\langle(HKKO), HKt_2x \odot xt_3yO \odot yt_1K\rangle\varphi$ is true in a future state (i.e. one door is opened, the player has two keys and it is possible to unlock another door).

Note that the player has three keys and, once one of them is used the resource amount is decreased (i.e. one token moves out from place H) and it is possible to set the amount of doors opened (i.e. the amount of tokens in O). There is, it is possible to deal with count of resources and its consumption.

Another property inherited from Petri Nets is the simplified way to deal with concurrence. In a scenario where the player can not only open a door (as in Figure 8a) but there is also the possibility to take an object disposed into the room, as in Figure 9, where place B denotes that the user is taking same object in the current room. If t_1' fires then t_2 will be unable to fire. So, it is possible to model in Petri-PDL that when a player is taking something in the current room he cannot open a

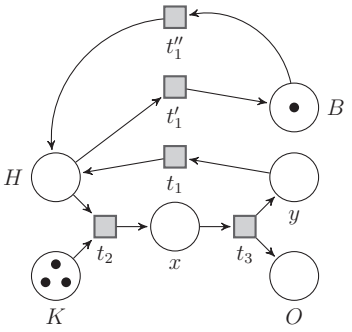

FIG. 9. A Petri Net for a game where doors separate stages.

door. So, modelling the scenario in Figure 9 we have that $\langle (KKK, B), HKt_2x \odot xt_3yO \odot yt_1H \odot Ht'_1B \odot Bt''_1H \rangle \top \rightarrow \neg \langle (KKK, B), HKt_2x \rangle \langle \epsilon, HKt_2x \odot xt_3yO \odot yt_1H \odot Ht'_1B \odot Bt''_1H \rangle \top$. Looking to the left side of the implication, as t''_1 is able to fire it is straightforward that the left side of the implication is true by the semantical notion of satisfaction of Petri-PDL. But, for the right side of the implication, t_1 is not able to fire, so cannot be true, hence its negation is true and the formula is valid. Note that the ϵ in the second modality of the right side of the implication is the result of the firing function for $f(KKKB, HKt_2x)$.

6 Conclusions and further work

The contribution of this work is fourfold. First, using previous ideas from [6], we propose a novel encoding of Petri Nets as programs of a Dynamic Logic. This is a modular and compositional encoding, we have three basic types of Petri Nets, and using a composition operator we can build more complex Nets. Second, we introduce a Propositional Dynamic Logic which the programs are Marked Petri Nets. Unlike previous approaches [15, 16, 31], which translate Petri Nets into Dynamic Logic, in our's we have Petri Nets encoded as programs of PDL yielding a new Dynamic Logic tailored to reasoning about Petri Nets in a more natural way. Third, we present an axiomatization to our logic and prove its soundness and completeness. Finally, we establish the decidability and finite model property of the logic. We also provide some examples of the usage of our approach at the end of the article.

As future work, we would like to investigate the usage of Petri-PDL in the formal verification of software, game modelling [5] and multiagent systems verification [21]. We also would like to extend our approach to other Petri Nets, like Timed and Stochastic Petri Nets. Finally, we would like to study issues concerning Model Checking and Automatic Theorem Prover to Petri-PDL.

References

[1] K. R. Abrahamson. *Decidability and Expressiveness of Logics of Processes*. PhD Thesis, Department of Computer Science, University of Washington, 1980.

[2] P. Balbiani and D. Vakarelov. PDL with intersection of programs: a complete axiomatization. *Journal of Applied Non-Classical Logics*, **13**, 231–276, 2003.

[3] M. R. F. Benevides and L. M. Schechter. A propositional dynamic logic for CCS programs. In *Proceedings of the XV Workshop on Logic, Language*, vol. 5110, pp. 83–97, 2008.

[4] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic. Theoretical Tracts in Computer Science*. Cambridge University Press, 2001.
[5] V. Gonçalves da Costa, B. Lopes, and E. Hermann Haeusler. Computer game modelling and reasoning by Petri Nets. In *Anais do XVI Simpósio Brasileiro de Métodos Formais*, pp. 24–29, 2013.
[6] E. S. de Almeida and E. H. Haeusler. Proving properties in ordinary Petri Nets using LoRes logical language. *Petri Net Newsletter*, **57**, 23–36, 1999.
[7] L. Fariñas del Cerro and E. Orlowska. DAL – a logic for data analysis. *Theoretical Computer Science*, **36**, 251–264, 1985.
[8] R. Fagin, J. Y. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge*. MIT Press, 2004.
[9] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, **18**, 194–211, 1979.
[10] G. Gargov and S. Passy. A note on boolean modal logic. In PetioPetrov Petkov, editor, *Mathematical Logic*, pp. 299–309. Springer US, 1990.
[11] R. Goldblatt. Parallel action: Concurrent dynamic logic with independent modalities. *Studia Logica*, **51**, 551–558, 1992.
[12] D. Harel and M. Kaminsky. Strengthened results on nonregular PDL. Technical Report MCS99-13, Faculty of Mathematics and Computer Science, Weizmann Institute of Science, 1999.
[13] D. Harel and D. Raz. Deciding properties of nonregular programs. *SIAM Journal on Computing*, **22**, 857–874, 1993.
[14] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. Foundations of Computing Series. MIT Press, 2000.
[15] R. Hull. Web services composition: a story of models, automata, and logics. In *Proceedings of the 2005 IEEE International Conference on Web Services*, 2005.
[16] R. Hull and J. Su. Tools for composite web services: A short overview. *ACM SIGMOD*, **34**, 86–95, 2005.
[17] S. Khosravifar. Modeling multi agent communication activities with Petri Nets. *International Journal of Information and Education Technology*, **3**, 310–314, 2013.
[18] M. Kracht. Synctatic codes and grammar refinement. *Journal of Logic, Language and Information*, **4**, 41–60, 1995.
[19] M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 205–212. AAAI Press, 1994.
[20] C. Löding, C. Lutz, and O. Serre. Propositional dynamic logic with recursive programs. *Journal of Logic and Algebraic Programming*, **73**, 51–69, 2007.
[21] B. Lopes, M. Benevides, and E. Hermann Haeusler. Verifying properties in multi-agent systems using stochastic Petri Nets and Propositional Dynamic Logic. In *X Encontro Nacional de Inteligncia Artificial e Computacional*, 2013.
[22] A. Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, vol. 255 of *Lecture Notes in Computer Science*, W. Brauer, W. Reisig, and G. Rozenberg, eds, pp. 278–324. Springer, 1987.
[23] A. Mazurkiewicz. Basic notions of trace theory. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, vol. 354 of *Lecture Notes in Computer Science*, J. W. Bakker, W.-P. Roever, and G. Rozenberg, eds, pp. 285–363. Springer, 1989.
[24] J.-J. Ch. Meyer. A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic*, **29**, 109–136, 1987.

[25] G. Mirkowska. PAL – Propositional Algorithmic Logic. *Fundamenta Informaticæ*, **4**, 675–760, 1981.
[26] D. Peleg. Concurrent dynamic logic. *Journal of the Association for Computing Machinery*, **34**, 450–479, 1897.
[27] D. Peleg. Communication in concurrent dynamic logic. *Journal of Computer and System Sciences*, **35**, 23–58, 1987.
[28] A. Petkov. Propositional dynamic logic in two and more dimensions. In *Mathematical Logic and its Applications*. Plenum Press, 1987.
[29] C. A. Petri. Fundamentals of a theory of asynchronous information flow. *Communications of the ACM*, **5**, 319–319, 1962.
[30] V. R. Pratt. Semantical considerations on Floyd-Hoare logic. Technical report, Massachusetts Institute of Technology, 1976.
[31] H. Tuominen. Elementary net systems and dynamic logic. In *Advances in Petri Nets 1989*, G. Rozenberg, ed., *Lecture Notes in Computer Science*, pp. 453–466. Springer Berlin Heidelberg, 1990.
[32] J. van Benthem. Logic and information flow. In *Foundations of Computing*. MIT Press, 1994.
[33] F. Wolter and M. Zakharyaschev. Dynamic description logics. In *Proceedings of AiML'98*, pp. 290–300. CSLI Publications, 2000.

Received 5 December 2012