Theoretical Computer Science ••• (••••) •••–•••

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Towards reasoning about Petri nets: A Propositional Dynamic Logic based approach ☆

Mario Benevides [a], Bruno Lopes [b,*], Edward Hermann Haeusler [c]

[a] Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Brazil
[b] Instituto de Computação, Universidade Federal Fluminense, Brazil
[c] Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Brazil

ARTICLE INFO

Article history:
Received 13 April 2017
Received in revised form 30 December 2017
Accepted 8 January 2018
Available online xxxx

Keywords:
Dynamic Logic
Petri nets
Modal Logic

ABSTRACT

This work extends our previous work [4,22] with the iteration operator. This new operator allows for representing more general networks and thus enhancing the former propositional logic for Petri nets. We provide an axiomatization and a new semantics, prove soundness and completeness with respect to its semantics and the EXPTIME-Hardness of its satisfiability problem, present a linear model checking algorithm and show that its satisfiability problem is in 2EXPTIME. In order to illustrate its usage, we also provide some examples.

1. Introduction

Propositional Dynamic Logic (PDL) [10] plays an important role in formal specification and reasoning about programs and actions. PDL is a multi-modal logic with one modality for each program π $\langle\pi\rangle$. It has been used in formal specification to reasoning about properties of programs and their behavior. Correctness, termination, fairness, liveness and equivalence of programs are among the properties usually desired. A Kripke semantics can be provided, with a frame $\mathcal{F} = \langle W, R_\pi \rangle$, where W is a 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.

There are a lot of variations of PDL for different approaches [2]. Propositional Algorithmic Logic [26] that analyzes properties of programs connectives, the interpretation of Deontic Logic as a variant of Dynamic Logic [25], applications in linguistics [19], Multi-Dimensional Dynamic Logic [29] that allows multi-agent [18] representation, Dynamic Arrow Logic [32] to deal with transitions in programs, Data Analysis Logic [8], Boolean Modal Logic [11], logics for reasoning about knowledge [9], logics for knowledge representation [20] and Dynamic Description Logic [34].

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

In [22], we present the logic Petri-PDL which uses marked Petri net programs as PDL programs. These marked Petri net programs are representations of marked Petri nets with only three types of transitions: (i) from one place to one place, (ii) from two places to one place and (iii) from one place to two places. The results presented in this paper consider only

☆ This work was supported by the Brazilian research agencies CNPq, CAPES and FAPERJ.

* Corresponding author.
E-mail addresses: mario@cos.ufrj.br (M. Benevides), bruno@ic.uff.br (B. Lopes), hermann@inf.puc-rio.br (E.H. Haeusler).
URLs: http://www.cos.ufrj.br/~mario/ (M. Benevides), http://www.ic.uff.br/~bruno (B. Lopes), http://www.tecmf.inf.puc-rio.br/EdwardHermann (E.H. Haeusler).

https://doi.org/10.1016/j.tcs.2018.01.007
0304-3975/

Petri nets with these types of transitions. So if π is a Petri net program with markup s, then the formula $\langle s, \pi\rangle\varphi$ means that after some running of 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 $\neg\langle s, \pi\rangle\neg\varphi$).

This work extends our previous work [4,22] with the iteration operator. This new operator allows for representing more general networks and thus enhancing the former propositional logic for Petri nets. We provide a sound and complete axiomatization and also prove the finite model property for the case of bounded Petri net programs, which together with the axiomatization yields decidability. In this paper we extend these previous works by establishing that, for the case of unbounded Petri net programs, the logic has no finite model property, proving that its SAT problem is EXPTIME-hard and that it is in 2EXPTIME; we also present a linear time model checking algorithm.

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 [13,14] and [21], that develop decidable dynamic logics for fragments of the class of context-free programs and [1,3,12,27] and [28], 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 [27] and Concurrent PDL with Channels [28]. 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 [28]. On the other hand, our logics have simple Kripke semantics, simple and complete axiomatizations and finite model property.

There are other approaches that use Dynamic Logic to reason about specifications of concurrent systems represented as Petri nets [16,17,31]. They differ from our approach by the fact that they use Dynamic logic as a 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 reason about Petri nets in a more natural way.

This paper is organized as follows. Section 2 presents all the background needed about (marked) Petri nets formalism and Propositional Dynamic Logic. Section 3, introduces our dynamic logic, with its language and semantics and also proposes an axiomatization and provides a prove of soundness and completeness. In Section 5 we show that the satisfiability problem is EXPTIME-Hard and is in 2EXPTIME and present a linear time model checking algorithm. Section 6 illustrates the use of our logic with some examples. Finally, Section 7, presents some final remarks and future works.

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 [15]. Second, we present the Petri nets formalism and its variant, marked Petri nets. Finally, the compositional approach introduced in [7] 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 [10]. We deliberately omit the test operator as we do not use it in this work.

Definition 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 formulae 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 paper, we use the standard abbreviations $\bot \equiv \neg\top$, $\varphi \vee \phi \equiv \neg(\neg\varphi \wedge \neg\phi)$, $\varphi \to \phi \equiv \neg(\varphi \wedge \neg\phi)$ 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 run of π.

The semantics of PDL is normally given using a transition diagram, which consists of a set of 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. 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

Please cite this article in press as: M. Benevides et al., Towards reasoning about Petri nets: A Propositional Dynamic Logic based approach, Theoret. Comput. Sci. (2018), https://doi.org/10.1016/j.tcs.2018.01.007