Logics for Petri nets with propagating failures [⋆]

Leandro Gomes[1], and Alexandre Madeira[1,2] and Mario Benevides[3]

[1] HASLab INESC TEC - Univ. Minho, Portugal
[2] CIDMA - Univ. Aveiro, Portugal
[3] COPPE/PESC - Instituto de Matemática/DCC
Universidade Federal do Rio de Janeiro, Brasil

Abstract. Petri nets play a central role in the formal modelling of a wide range of complex systems and scenarios. Their ability to handle with both concurrency and resource awareness justifies their spread in the current formal development practices. On the logic side, Dynamic Logics are widely accepted as the *de facto* formalisms to reason about computational systems. However, as usual, the application to new situations raises new challenges and issues.
The ubiquity of failures in the execution of current systems, interpreted in these models as triggered events that are not followed by the corresponding transition, entails not only the adjustment of these structures to deal with this reality, but also the introduction of new logics adequate to this emerging phenomenon.
This paper contributes to this challenge by exploring a combination of two previous works of the authors, namely the Propositional Dynamic Logic for Petri Nets [1] and a parametric construction of multi-valued dynamic logics presented in [13]. This exercise results in a new family of Dynamic Logics for Petri Nets suitable to deal with firing failures.

1 Introduction

Petri nets are semantic structures widely used in computer science. Their adequacy to model, specify and analyze complex systems dealing with concurrency and resource awareness is well known. At the core of this success is their rich and intuitive graphical syntax. Nevertheless, for property oriented specification and verification purposes, it is useful to consider logic systems having this structure as a semantics. The first attempt in such direction was done by the linear logic community (e.g. [6]). A number of other logics were then proposed in the literature for standard and timed versions of Petri nets eg. [3].

⋆ This work is financed by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within projects POCI-01-0145-FEDER-016692/POCI-01-0145-FEDER-029946. The second author is also supported by the individual grant SFRH/BPD/103004/2014. This paper is a result of the project SmartEGOV: Harnessing EGOV for Smart Governance (Foundations, Methods, Tools) NORTE-01-0145-FEDER-000037, supported by Norte Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, through the European Regional Development Fund (EFDR). It received further support from the PT-FLAD Chair in Smart Cities & Smart Governance.

Propositional dynamic logic (PDL) [8], a very versatile logic for verification of computational systems, was also explored in this context. Particularly, Petri-PDL [12] was introduced as an extention of PDL to give logical semantics to Petri nets. In such variant, the programs are marked Petri nets expressed by a textual syntax (with a choice and a composition operators). Another approach was taken in [5], where the well established theory of modal semirings was used to develop a generic modal algebra for reasoning about reachability properties in Petri nets. The BI resource based semantics presented in [16] introduces another proposal to allow comparison of "amounts of information" modelled by the possible worlds of the model. In this work, Petri nets are presented as concrete instances of this semantics. Later, the work of [12] was extended [1] to include the iteration operator. This logic is expressive enough to describe properties of systems like the one shown in the example below:

Example 1. Let us present the following situation, based on the example presented in [1], which illustrates the behaviour of a chocolate vending machine. The system works as follows: we turn the machine on (l) and put one coin (m) and then it releases the chocolate (c).

Its behaviour can be specified by the Petri net of Figure 1. 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 released. We can express that *once we turn the machine on and put one coin we can obtain a chocolate* by the formula $\langle(\ell, m), (\ell m t_2 x \odot x t_3 y c)\rangle\top$, meaning that, if we are in a state were states ℓ and m are marked, after executing the Petri net program $\ell m t_2 x \odot x t_3 y c$, we reach a state were $\top$ is satisfied.

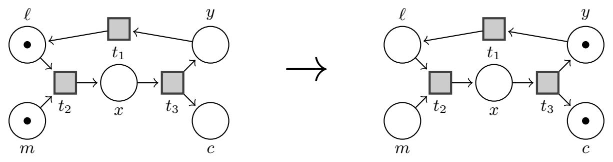

Fig. 1. Execution of the program $\ell m t_2 x \odot x t_3 y c$ in the chocolate machine

The above formula can be proved using the proof system presented in [1].

The complexity of modern systems, namely the heterogeneity of the environments where they live, entails a more demanding approach in their design and building processes. The inevitability that human beings eventually make mistakes in such processes extols the advantage of adopting formalisms which deal with the possibility of failures at the outset.

Recalling again the previous example, we would bet that the reader already experienced undesirable situations like *after putting a coin in a vending machine, the desired chocolate gets stuck behind of the glass.* The analysis of such kinds of failures would require a formalism able to express statements like *once we turn the machine on and put one coin we have an assurance α that we will obtain a chocolate.* A possible model for handling theses scenarios would be a variation of the classical definition of Petri nets, where the triggering of a transition happens with some reliability degree. The resulting assurance would depend on two aspects: on one side, the reliability degree associated to the execution of a program; on the other, an appropriate evaluation of a formula in a many-valued truth space. Despite their adequacy in the reasoning about several properties of Petri nets, none of the approaches presented in [11], [5] or [16] aim to formalise the attribution of such degrees.

In order to achieve a Dynamic Logic analogous to [1], but suitable to this scenario, we base this work on the construction of multi-valued dynamic logics introduced in [14, 13]. First, we introduce a new variant of the Petri Nets, the *Petri nets with* **A**-*failures*, which explicitly assumes that the modelled system may eventually fail, supporting a claim of reliability to each firing of a transition. In the case where a failure occurs, the firing event is consumed without the occurrence of the expected transition. Depending on the system modelled, it would make sense to measure these reliability degrees in a discrete scale, in a continuous interval, or simply in an universe only with the values *true* and *false*. Hence, instead of fixing the domain of the assurances degrees as the usual real interval $[0, 1]$, we will be more generic: the proposed models are parametric to the nature of the reliability degrees that are most suitable for each concrete situation. This flexibility is realised by assuming, as parameter, an (action) lattice $\mathbf{A}$. Some additional considerations on this parameter are in order.

As in standard PDL, the interpretation of programs in Petri-PDL [11] relies on the Kleene algebra of relations (as we will see in the next section, firing functions are interpreted as binary relations). However, firing functions in Petri nets with **A**-failures are based on reliability degrees given by elements of an action lattice, and not on the interpretation of classic binary relations.

To give meaning to Petri net programs in this new formalism, this paper adopts a class of Kleene algebras, parameterised by an action lattice $\mathbf{A}$. These algebraic structures base the interpretation of (composed) programs in Petri nets with **A**-failures, by reflecting how failures in transitions are propagated into the whole execution of a program. This is also reflected in the kind of assertions we can express in the logic, as well as in the outcome we expect for the validity of a Dynamic Logic formula in a Petri net with **A**-failures. For that purpose, we follow a strategy similar to that used in [14, 13], where an action lattice will follow a similar role in our method, namely as a computational model (by representing weighted fails) and as a truth universe (by giving a (possible) many-valued truth space to the addressed logics). The proposed method for constructing graded dynamic logics is parametric on whatever truth/computational domain is the most suitable to the system under consideration.

Outline. The remaining of this paper is organized as follows. Section 2 presents all the background needed about the Petri-PDL* formalism and the concept of action lattice. Section 3 is devoted to Petri Net with **A**-failures, as well as, to suitable Kleene Algebras to interpret programs in these structures. In Section 4, we introduce a parametric construction of dynamic logics for Petri nets with **A**-failures, and proper illustrations for such a method. Finally, Section 5 concludes the paper with some final remarks on future work.

2 Background

In this section, we present a brief overview of two topics on which the later development is based. First, we review the syntax and semantics of Petri-PDL* [1]. Second, we recall the main notions behind the method introduced in [13], for the generation of many-valued dynamic logics parametrized by an action lattice. This parameter supports both the (possible non-bivalent) truth spaces and the base computational model.

2.1 Propositional Dynamic Logic for Petri nets with Iteration (Petri-PDL*)

This subsection recalls the syntax and semantics of Petri-PDL*, as presented in [1]. The language of Petri-PDL* consists of

Propositional symbols: $Prop = \{p, q, \dots\}$
Place names: $a, b, c, d, \dots$
Transition types: $T_1 : xt_1y$, $T_2 : xyt_2z$ and $T_3 : xt_3yz$
Petri net Composition symbol: $\odot$
PDL operator: $\_^\star$ (iteration)
Markings: $S = \{\epsilon, s_1, s_2, \dots\}$, where ϵ is the empty multiset and s_i is a multiset of place names.

Definition 1. *Petri-PDL* Programs for a set of places P*

Basic programs: *set $\Pi_0(P)$ defined by the grammar*

$$\pi ::= at_1b \mid abt_2c \mid at_3bc$$

where t_i is of type $T_i, i = 1, 2, 3$ and $a, b, c \in P$.
Petri net Programs: *set $\Pi(P)$ defined by the grammar*

$$\eta ::= \pi \mid \pi \odot \eta \mid \eta^\star$$

for $\pi \in \Pi_0(P)$

Definition 2. *Let* Prop *be a set of propositions. The set of* $\mathrm{Petri-PDL}^\star(\mathbf{A})$ *formulas for* Prop*, denoted by* $\mathrm{Fm}^{\mathrm{Petri-PDL}^\star(\mathbf{A})}(\mathrm{Prop})$*, is defined by the grammar*

$$\rho ::= p \mid \top \mid \neg\rho \mid \rho \wedge \rho \mid \langle s, \eta\rangle\rho$$

where $p \in \mathrm{Prop}$.

We use the standard abbreviations $\bot \equiv \neg\top$, $\rho \vee \rho \equiv \neg(\neg\rho \wedge \neg\rho)$, $\rho \to \rho \equiv \neg(\rho \wedge \neg\rho)$ and $[s,\eta]\rho \equiv \neg\langle s,\eta\rangle\neg\rho$.

Before presenting the Petri net dynamics, let us first introduce the following definition.
The definition below introduces the *firing* function. It defines how the marking of a basic Petri net changes after a firing.

Definition 3. *For a set of markings S, we define the firing function $f : S \times \Pi_0 \to S$ as follows*

- $f(s, at_1b) = \left\{ \begin{array}{r} s_1 \mid b \in s_1,\, if\ a \in S \\ \epsilon,\, if\ a \notin s \end{array} \right\}$
- $f(s, abt_2c) = \left\{ \begin{array}{r} s_2 \mid c \in s_2,\, if\ a, b \in s \qquad\qquad \\ \epsilon,\, if\ a \notin s\ or\ b \notin s \end{array} \right\}$
- $f(s, at_3bc) = \left\{ \begin{array}{r} s_3 \mid b, c \in s_3,\, if\ a \in s \\ \epsilon,\, if\ a \notin s \end{array} \right\}$

The definitions of frame, model and satisfaction, that we recall below, are adapted from PDL to deal with the firing of basic Petri nets.

Definition 4. *A frame for Petri-PDL* is a triple $\langle W, R_\pi, M\rangle$, where*

- *W is a non-empty set of states;*
- *$M\colon W \to S$;*
- *R_π is a binary relation over W, for each basic program π, satisfying the following condition: let $s = M(w)$*
  - *if $f(s,\pi) \neq \epsilon$, if $wR_\pi v$ then $f(s,\pi) \preceq M(v)$*
  - *if $f(s,\pi) = \epsilon$, $(w,v) \notin R_\pi$*
- *we inductively define a binary relation R_η, for each Petri net program η, as follows:*
  - *$R_{\eta^*} = R^*_\eta$, where R^*_η denotes the reflexive transitive closure of R_η.*
  - *$\eta = \eta_1 \odot \eta_2 \odot \cdots \odot \eta_n$*

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

  where $s = M(w)$, η_i are Petri net programs and $s_i = f(s,\eta_i)$, for all $1 \leq i \leq n$.

Definition 5. *A* model *for Petri-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 from a set of* propositions Prop, $\mathbf{V}\colon \text{Prop} \to 2^W$.

The semantical notion of satisfaction for Petri-PDL* is defined below.

Definition 6. *Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ be a model. The notion of* satisfaction *of a formula ρ in a model $\mathcal{M}$ at a state w, notation $\mathcal{M}, w \models \rho$, can be inductively defined as follows:*

- $\mathcal{M}, w \models p$ *iff* $w \in \mathbf{V}(p)$;
- $\mathcal{M}, w \models \top$ *always;*
- $\mathcal{M}, w \models \neg\rho$ *iff* $\mathcal{M}, w \not\models \rho$;
- $\mathcal{M}, w \models \rho \wedge \rho'$ *iff* $\mathcal{M}, w \models \rho$ *and* $\mathcal{M}, w \models \rho'$;
- $\mathcal{M}, w \models \langle s, \eta \rangle \rho$ *if there exists* $w' \in W$, $wR_\eta w'$, $s \subseteq M(w)$ *and* $\mathcal{M}, w' \models \rho$.

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

2.2 Kleene algebras and Action Lattice

We review in this section the notion of Action Lattice [10] as it was used in [13].

Definition 7 (Kleene Algebra and Action lattice). *An* action lattice *is a tuple* $\mathbf{A} = (A, +, ;, 0, 1, *, \to, \cdot)$, *where A is a set, 0 and 1 are constants, $*$ is an unary operation in A and $+, ;, \to$ and $\cdot$ are binary operations in A satisfying the axioms enumerated in Figure 2, where the relation $\leq$ is induced by $+$: $a \leq b$ iff $a + b = b$. An* integral action lattice *consists of an action lattice satisfying $a \leq 1$, for all $a \in A$. A* Kleene Algebra *is a structure $(A, +, ;, 0, 1, *)$ satisfying (1)-(13).*

$$a + (b + c) = (a + b) + c \tag{1}$$
$$a + b = b + a \tag{2}$$
$$a + a = a \tag{3}$$
$$a + 0 = 0 + a = a \tag{4}$$
$$a; (b; c) = (a; b); c \tag{5}$$
$$a; 1 = 1; a = a \tag{6}$$
$$a; (b + c) = (a; b) + (a; c) \tag{7}$$
$$(a + b); c = (a; c) + (b; c) \tag{8}$$
$$a; 0 = 0; a = 0 \tag{9}$$
$$1 + (a; a^*) = a^* \tag{10}$$
$$1 + (a^*; a) = a^* \tag{11}$$
$$a; x \leq x \Rightarrow a^*; x \leq x \tag{12}$$
$$x; a \leq x \Rightarrow x; a^* \leq x \tag{13}$$
$$a; x \leq b \Leftrightarrow x \leq a \to b \tag{14}$$
$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \tag{15}$$
$$a \cdot b = b \cdot a \tag{16}$$
$$a \cdot a = a \tag{17}$$
$$a + (a \cdot b) = a \tag{18}$$
$$a \cdot (a + b) = a \tag{19}$$

Fig. 2. Axiomatisation of action lattices (from [10])

As stated in the introduction, the structure of an action lattice is explored in this paper along a double dimension: as a computational model and as a truth space. The intuitions for some of its operations shall be taken from both of these perspectives. Such is the case of operation $+$, which plays the role of non-deterministic choice, in the interpretation of programs, and of logical disjunction, in the interpretation of sentences. However, there are operations whose intuition is borrowed from just in one of these domains. For instance, while operations $*$ and ; are taken as iterative execution and sequential composition of actions, operations $\rightarrow$ and $\cdot$ play the role of logical implication and conjunction, respectively.

The following structures are examples of action lattices:

*Example 2 (**2** - linear two-values lattice.).* As a first action lattice example, we consider the two valued boolean lattice $\mathbf{2} = (\{\top, \bot\}, \vee, \wedge, \bot, \top, *, \rightarrow, \wedge)$ with the standard boolean connectives defined as follows:

$$\begin{array}{c|cc} \vee & \bot & \top \\ \hline \bot & \bot & \top \\ \top & \top & \top \end{array} \quad \begin{array}{c|cc} \wedge & \bot & \top \\ \hline \bot & \bot & \bot \\ \top & \bot & \top \end{array} \quad \begin{array}{c|cc} \rightarrow & \bot & \top \\ \hline \bot & \top & \top \\ \top & \bot & \top \end{array} \quad \begin{array}{c|c} * & \\ \hline \bot & \top \\ \top & \top \end{array}$$

Example 3 ($\mathbf{W}_k$ finite Wajsberg hoops). We consider now an action lattice endowing the finite *Wajsberg hoops* [2] with a suitable star operation. Hence, for a fix natural $k > 0$ and a generator a, we define the structure $\mathbf{W}_k = (W_k, +, ;, 0, 1, *, \rightarrow, \cdot)$, where $W_k = \{a^0, a^1, \cdots, a^k\}$, $1 = a^0$ and $0 = a^k$, and for any $m, n \leq k$: $a^m + a^n = a^{min\{m,n\}}$, $a^m ; a^n = a^{min\{m+n,k\}}$, $(a^m)^* = a^0$, $a^m \rightarrow a^n = a^{max\{n-m,0\}}$ and $a^m \cdot a^n = a^{max\{m,n\}}$.

*Example 4 (**Ł** - the Łukasiewicz arithmetic lattice).* The *Łukasiewicz arithmetic lattice* is the structure $\mathbf{Ł} = ([0,1], \max, \odot, 0, 1, *, \rightarrow, \min)$, where $x \rightarrow y = \min(1, 1 - x + y)$, $x \odot y = \max(0, y + x - 1)$ and $x^* = 1$.

More examples and properties of action lattices can be found in [13].

3 Petri nets with failures

This section introduces the notion of Petri net with **A**-failures, as well as suitable Kleene algebras to interpret (composed) programs. As referred in the introduction, the use of an action lattice **A** as parameter is due to the necessity of supporting a double dimension: (i) attribute a reliability degree to the firing of a transition, referring to the interpretation of Petri net programs; (ii) state a degree for a specific property of a Petri net, on the logical side.

As stated, this work is concerned with Petri nets where transitions between markings may fail. This assumption entails adjusting the system dynamics of classical Petri nets: while, in such case, we argue that a system evolves to another markings if a transition is enabled, in our approach, a transition to a new marking occurs with a reliability degree α, where α is an element of the lattice **A**. In cases where the Petri net does not transit to another marking, the transition is still consumed. Formally:

Definition 8. *Given an action lattice $\mathbf{A} = (A, +, ;, 0, 1, *, \rightarrow, \cdot)$, a set S of markings (over a set of place names P) and a basic Petri net program $\pi \in \Pi_0$, an α-*firing function*, for $\alpha \in A$, is a function $f^{\alpha}_{\pi} : S \times S \rightarrow A$ defined as*

– *for any $a\, t_1\, b \in \Pi_1$,* $f^{\alpha}_{at_1b}(s, s') = \left\{ \begin{array}{r} \alpha, \text{ if } a \in s \text{ and } b \in s' \\ \alpha \rightarrow 0, \text{ if } a \in s \text{ and } a \in s' \\ 0 \text{ if } a \notin s \end{array} \right\}$

– *for any* $ab\,t_2\,c \in \Pi_2$, $f^{\alpha}_{abt_2c}(s,s') = \left\{ \begin{array}{r} \alpha,\ \text{if } a,b \in s \text{ and } c \in s' \\ \alpha \to 0,\ \text{if } a,b \in s \text{ and } a,b \in s' \\ 0 \text{ if } a,b \notin s \end{array} \right\}$

– *for any* $a\,t_3\,bc \in \Pi_3$, $f^{\alpha}_{at_3bc}(s,s') = \left\{ \begin{array}{r} \alpha,\ \text{if } a \in s \text{ and } b,c \in s' \\ \alpha \to 0,\ \text{if } a \in s \text{ and } a \in s' \\ 0 \text{ if } a \notin s \end{array} \right\}$

where $\Pi_i(P)$ are the following partitions of the atomic programs: $\Pi(P)$, $\Pi_1(P) = \{x\,t_1\,y \mid x,y \in P\}$, $\Pi_2(P) = \{xy\,t_2\,z \mid x,y,z \in P\}$ and $\Pi_3(P) = \{x\,t_3\,yz \mid x,y,z \in P\}$.

Now, we have conditions to introduce the intended model.

Definition 9 (Petri net with A-failures). *Let $\mathbf{A} = (A, +, ;, 0, 1, *, \to, \cdot)$ be an action lattice. A* Petri net with **A**-failures *consists of a tuple $\mathcal{P} = \big(P, S, \Pi_0, I, M_0\big)$ where P is a set of places; $S \subseteq P^*$ is the* set of (admissible) markings*; $\Pi_0 \subseteq \Pi(P)$ is the* set of atomic programs*; $I : \Pi_0 \to A$ is the* atomic programs reliability degree *and $M_0 \in S$ is the* initial marking. *The interpretation of an atomic program $\pi \in \Pi_0$ is given by the firing function $f^{I(\pi)}_{\pi}$.*

In this work, as in [13], the underlying Kleene algebra of **A** (c.f. Defn 7) provides a generic computational model for interpreting programs. However, differently form such work, we interpret computations as α-firing functions of Definition 8, which carry the information about their effect when executed. Hence, we define the following algebra:

Definition 10. *Let $\mathbf{A} = (A, +, ;, 0, 1, *, \to, \cdot)$ be an action lattice and S be a finite set. The algebra of* **A**-firing functions *is the structure $\mathbf{F} = (F, \cup, \circ, \varnothing, \chi, *)$ where:*

– *F is the universe of all the α-firing functions, for all $\alpha \in A$*
– $(f^{\alpha_1}_{\pi_1} \cup f^{\alpha_2}_{\pi_2})(s,s') = f^{\alpha_1}_{\pi_1}(s,s') + f^{\alpha_2}_{\pi_2}(s,s')$
– $(f^{\alpha_1}_{\pi_1} \circ f^{\alpha_2}_{\pi_2})(s,s') = \sum\limits_{s'' \in S} f^{\alpha_1}_{\pi_1}(s,s''); f^{\alpha_2}_{\pi_2}(s'',s')$
– $\varnothing(s,s') = 0$
– $\chi(s,s') = \begin{cases} 1, & \text{if } s = s' \\ 0, & \text{otherwise} \end{cases}$
– $(f^{\alpha}_{\pi})^*(s,s') = \bigvee\limits_{i \geq 0} (f^{\alpha}_{\pi})^i(s,s') = (f^{\alpha}_{\pi})^0(s,s') + (f^{\alpha}_{\pi})^1(s,s') + (f^{\alpha}_{\pi})^2(s,s') + \ldots$

The next theorem states that **F** represents an adequate structure to interpret Petri net programs for Petri nets with **A**-failures [4].

Theorem 1. **F** *is a Kleene Algebra*

Proof. This proof is analogous to the proof of the classical result [4], stating that the algebra of $n \times n$ matrices over a Kleene algebra is a Kleene algebra.

[4] A more generic algebraic structure, suitable to deal with generic weighted computations was recently introduced by the authors in [7].

With this Kleene algebra, we are able to interpret regular programs in Petri nets with **A**-failures, i.e. regular expressions of atomic transitions in **A**:

Definition 11 (A-interpretations of programs).
Let $\mathbf{A} = (A, +, ;, 0, 1, *, \rightarrow, \cdot)$ *be an action lattice and* $\big(P, S, \Pi_0, I, M_0\big)$ *a Petri net with* **A***-failures. The interpretation of a Petri net program* η *is a firing function recursively defined as follows:*

- *for any atomic program* $\pi \in \Pi_0$, $[\![\pi]\!](s, s') = f_\pi^{F(\pi)}(s, s')$
- $[\![\pi_1; \pi_2]\!](s, s') = ([\![\pi_1]\!] \circ [\![\pi_2]\!])(s, s')$,
- $[\![\eta^*]\!](s, s') = [\![\eta]\!]^*(s, s')$, *for* $[\![\eta]\!]^*(s, s') = \sum_{i \geq 0} [\![\eta]\!]^i(s, s')$ *where* $[\![\eta]\!]^0(s, s') = \chi(s, s')$ *and for any* $i \geq 0$, $[\![\eta]\!]^{i+1}(s, s') = \big([\![\eta]\!]^i \circ [\![\eta]\!]\big)(s, s')$
- $[\![\eta + \eta']\!](s, s') = [\![\eta]\!](s, s') + [\![\eta']\!](s, s')$, *for* η, η' *Petri net programs.*

The interpretation of Petri Net composed programs, as presented in Definition 1, where the *global composition* $\odot$ is considered rather than the *sequential composition* ;, can be handled in our logic indirectly by defining $\odot$ as

$$\eta \odot \eta' \equiv \eta; (\eta + \eta')^* + \eta'; (\eta + \eta')^* \tag{20}$$

where η and η' are Petri net programs.

4 Parametric construction of dynamic logics for Petri nets with failures

This section introduces a parametric method to build Petri-PDL to reason about Petri nets with **A**-failures, inspired by the construction proposed in [13]. The semantic and satisfaction of these logics are built on top of an arbitrary action lattice $\mathbf{A} = (A, +, ;, 0, 1, *, \rightarrow, \cdot)$ (c.f. Definition 7). Hence, the resulting logics will be denoted by $\mathbf{GP}(\mathbf{A})$. Petri-PDL*, as introduced in [12], is captured as an instance of this construction (by using, as parameter, the lattice **2** of Example 2). Beyond the reliability degrees for transitions, the action lattice also supports the truth space for the (possible multi-valued) outcomes of the logic.

The language for $\mathbf{GP}(\mathbf{A})$ is the same of Petri-PDL*, except for *formulae*, that we define below.

Definition 12. *Let* Prop *be a set of propositions. The set of* $\mathbf{GP}(\mathbf{A})$ *formulas for the set of propositions* Prop *and for the set of place names* P, *denoted by* $\mathrm{Fm}^{\mathrm{GP}(\mathbf{A})}(\mathrm{Prop}, P)$, *is defined by the grammar*

$$\rho ::= p \mid \top \mid \bot \mid \rho \wedge \rho \mid \rho \vee \rho \mid \rho \rightarrow \rho \mid \langle \xi \rangle \rho \mid [\xi] \rho$$

where $p \in \mathrm{Prop}$, $\xi ::= \pi \mid \pi; \xi \mid \xi^\star \mid \xi + \xi$ *for* $\pi ::= at_1 b \mid abt_2 c \mid at_3 bc$, *and* $a, b, c \in P$.

Observe that we denoted the regular programs (with the sequential composition ;) by letter ξ instead of η used for the Petri-net programs (with global composition $\odot$). However, in the sequel, we will relax this convention by using η for both cases. The symbol $\odot$ will also be used as meta-syntax of our logic, to be interpreted according to (20). For instance, the expression $\langle \eta \odot \eta' \rangle \varphi$ is just notation for the formula $\langle \eta; (\eta + \eta')^* + \eta'; (\eta + \eta')^* \rangle \varphi$.

Note that, differently from previous work [17], we do not include the negation as a primitive operator, and use, instead, the defined negation $\neg x \equiv x \to \bot$. Actually, as stated, we intend to deal with generic truth spaces for possible non bivalent interpretation of assertions (e.g. we are not requiring negative involution).

Definition 13. *A* model *for* $\mathbf{GP}(\mathbf{A})$ *is a pair* $\mathcal{M} = \langle \mathcal{P}, \mathbf{V} \rangle$*, where* $\mathcal{P}$ *is a Petri net with* $\mathbf{A}$*-failures and* $\mathbf{V}$ *is a valuation function over a set of propositions* Prop*, defined as* $\mathbf{V} : \mathrm{Prop} \times S \to A$.

Now, we define the semantic notion of satisfaction for $\mathbf{GP}(\mathbf{A})$.

Definition 14. *Let* $\mathbf{A} = (A, +, ;, 0, 1, *, \to, \cdot)$ *be an action lattice. The* (graded) satisfaction $\models : (\mathcal{M} \times S) \times \mathrm{Fm}^{\mathrm{GP}(\mathbf{A})}(\mathrm{Prop}) \to A$ *for* $\mathbf{GP}(\mathbf{A})$ *is recursively defined for each model* $\mathcal{M}$*, any marking* $s \in S$ *and for any formula* $\rho \in \mathrm{Fm}^{\mathrm{GP}(\mathbf{A})}(\mathrm{Prop})$ *as follows:*

- $(\mathcal{M}, s \models p) = \mathbf{V}(p, s)$;
- $(\mathcal{M}, s \models \top) = \top$;
- $(\mathcal{M}, s \models \bot) = \bot$;
- $(\mathcal{M}, s \models \rho \wedge \rho') = (\mathcal{M}, s \models \rho) \cdot (\mathcal{M}, s \models \rho')$;
- $(\mathcal{M}, s \models \rho \vee \rho') = (\mathcal{M}, s \models \rho) + (\mathcal{M}, s \models \rho')$;
- $(\mathcal{M}, s \models \rho \to \rho') = (\mathcal{M}, s \models \rho) \to (\mathcal{M}, s \models \rho')$;
- $(\mathcal{M}, s \models \langle \eta \rangle \rho) = \sum_{s' \in S} \Big([\![\pi]\!](s, s'); (\mathcal{M}, s' \models \rho) \Big)$;
- $(\mathcal{M}, s \models [\eta] \rho) = \bigwedge_{s' \in S} \Big([\![\pi]\!](s, s') \to (\mathcal{M}, s' \models \rho) \Big)$

Example 5. Let us start by revisiting Example 1, by using $\mathbf{GP}(\mathbf{2})$ (see Example 2). For that, we denote $f^{\top}_{\ell m t_2 x}(\ell m, x)$ by a and $f^{\top}_{x t_3 y c}(x, yc)$ by b and, in this situation $a = b = \top$. So, we can write the sentence of Example 1 as *once we*

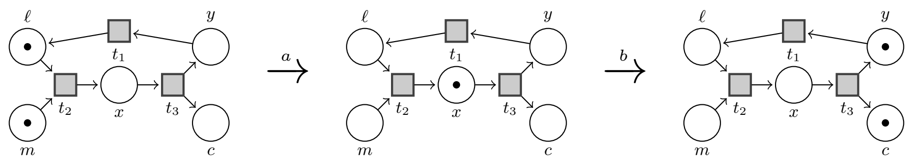

Fig. 3. A Petri net for a (possibly) defective chocolate vending machine

turn the machine on and put one coin we have the (total) reliability that we will *obtain a chocolate*. This expression can, then, be represented in $\mathbf{GP}(\mathbf{2})$ by the formula $(\mathcal{M}, \ell m \models \langle \ell m t_2 x \odot x t_3 y c\rangle \top) = \top$. Hence:

$$\begin{aligned}
&\mathcal{M}, \ell m \models \langle \ell m t_2 x; x t_3 y c\rangle \top \\
&= \sum_{s \in S} \Big([\![\ell m t_2 x; x t_3 y c]\!](\ell m, s); (M, s \models \top) \Big) \\
&= \sum_{s \in S} \Big(([\![l m t_2 x]\!] \circ [\![x t_3 y c]\!])(\ell m, s); (M, s \models \top) \Big) \\
&= \sum_{s \in S} \Big(\big(\sum_{s' \in S} ([\![\ell m t_2 x]\!](\ell m, s'); [\![x t_3 y c]\!](s', s)) \big); (M, s \models \top) \Big) \\
&= \sum_{s \in S} \Big(\big([\![\ell m t_2 x]\!](\ell m, x); [\![x t_3 y c]\!](x, s) + [\![\ell m t_2 x]\!](\ell m, \ell m); [\![x t_3 y c]\!](\ell m, s) \big); (M, s \models \top) \Big) \\
&= \big([\![\ell m t_2 x]\!](\ell m, x); [\![x t_3 y c]\!](x, y c) + [\![\ell m t_2 x]\!](\ell m, \ell m); [\![x t_3 y c]\!](\ell m, y c) \big); (M, y c \models \top) \\
&+ \big([\![\ell m t_2 x]\!](\ell m, \ell m); [\![x t_3 y c]\!](\ell m, \ell m) \big); (M, \ell m \models \top) \\
&+ \big([\![\ell m t_2 x]\!](\ell m, x); [\![x t_3 y c]\!](x, x) + [\![\ell m t_2 x]\!](\ell m, \ell m); [\![x t_3 y c]\!](\ell m, x) \big); (M, x \models \top) = \top
\end{aligned}$$

Assume now that this machine has a technical problem and it can not assure with total reliability the release of a chocolate every time a coin is inserted. Suppose also that we can express such reliability degrees in a $1 \ldots 10$ discrete scale. Hence, using $\mathbf{GP}(W_{10})$ (see Ex.3), we can express that the machine transits from marking lm to x with a reliability 8 and from x to yc with 9, i.e. $a = 8$ and $b = 9$. Now, the verification of the property *once we turn the machine on and put one coin we have a reliability* 7 *out of* 10 *that we will obtain a chocolate*, from the marking ℓm, can be computed by
$\mathcal{M}, \ell m \models \langle \ell m t_2 x; x t_3 y c\rangle \top \ = \big(8; 9 + (8 \to 0); 0\big); 10 + \big((8 \to 0); 0\big); 10 + \big(8; (9 \to 0) + (8 \to 0); 0\big); 10 \ = \ 7.$

Nevertheless, for some situations, it could be more appropriate to use a continuous scale. Suppose, for instance, that we want to be more precise, by stating that the reliability degree of the machine to evolve from marking lm to x is 0.78 and from x to yc is 0.93, which corresponds to $a = 0.78$ and $b = 0.93$. This can be expressed using $\mathbf{GP}(\text{Ł})$ (see Ex. 4). In this case we have:
$\mathcal{M}, \ell m \models \langle \ell m t_2 x; x t_3 y c\rangle \top = max\{max\{0.78 \odot 0.93, (0.78 \to 0) \odot 0\} \odot 1, ((0.78 \to 0) \odot 0) \odot 1, max\{0.78 \odot (0.93 \to 0), (0.78 \to 0) \odot 0\} \odot 1\} = 0.71.$

Let us now use the global composition operator $\odot$, in place of the simple sequential composition ;. Given the Petri net program $\eta = \ell m t_2 x \odot x t_3 y c$, we compute the reliability degree of the formula $\langle \ell m t_2 x \odot x t_3 y c\rangle \top$, using $\mathbf{GP}(\mathbf{2})$, as follows:

$$\begin{aligned}
&\mathcal{M}, \ell m \models \langle \ell m t_2 x \odot x t_3 y c\rangle \top \\
&= \sum_{s \in S} \Big([\![\ell m t_2 x \odot x t_3 y c]\!](\ell m, s); (M, s \models \top) \Big) \\
&= \sum_{s \in S} \Big([\![\ell m t_2 x; \eta^* + x t_3 y c; \eta^*]\!](\ell m, s); (M, s \models \top) \Big) \\
&= \sum_{s \in S} \Big(([\![\ell m t_2 x; \eta^*]\!](\ell m, s) + [\![x t_3 y c; \eta^*]\!](\ell m, s)); (M, s \models \top) \Big)
\end{aligned}$$

$$= \sum_{s \in S} \Big(((\llbracket \ell m t_2 x \rrbracket \circ \llbracket \eta^* \rrbracket)(\ell m, s) + (\llbracket x t_3 y c \rrbracket \circ \llbracket \eta^* \rrbracket)(\ell m, s)); (M, s \models \top) \Big)$$

$$= \sum_{s \in S} \Big((\sum_{s' \in S} (\llbracket \ell m t_2 x \rrbracket (\ell m, s'); \llbracket \eta^* \rrbracket (s', s)) + \sum_{s' \in S} (\llbracket x t_3 y c \rrbracket (\ell m, s'); \llbracket \eta^* \rrbracket (s', s))); (M, s \models \top) \Big)$$

$$= \sum_{s \in S} \Big((\llbracket \ell m t_2 x \rrbracket (\ell m, x); \llbracket \eta^* \rrbracket (x, s) + \llbracket \ell m t_2 x \rrbracket (\ell m, \ell m); \llbracket \eta^* \rrbracket (\ell m, s) + \llbracket x t_2 y c \rrbracket (\ell m, x); \llbracket \eta^* \rrbracket (x, s)$$

$$+ \llbracket x t_3 y c \rrbracket (\ell m, \ell m); \llbracket \eta^* \rrbracket (\ell m, s)); (M, s \models \top) \Big)$$

$$= ((\top \wedge \top) \vee (\bot \wedge \top) \vee (\bot \wedge \top) \vee (\bot \wedge \top)) \wedge \top \vee \big(\bot \wedge \top\big) \wedge \top \vee ((\top \wedge \top) \vee (\bot \wedge \top)) \wedge \top = \top$$

5 Conclusions and further work

In this work, we contributed with the generalisation of the logic presented in [1], by considering that the firing of a Petri net may fail. The approach taken in order to handle this variation was based on previous work done in [13], where an action lattice is considered to model both the notion of reliability degree of transitions in models and to support the (possible) multi-valued truth degree of a formula. This goal was accomplished by introducing: (i) a new definition of Petri net, where transitions between markings may fail; (ii) an underlying class of Kleene algebras (parametric on an action lattice), suitable for interpreting (composed) Petri net programs; (iii) a parametric method to build Dynamic logics with these semantics.

The extension of this work can be done in several directions. First, the necessity to have supporting computational tools for these logics suggests the development of a proof calculi and model checking algorithms, with their computational complexity. In this line, we expect to obtain characterizations, parametric to the base action lattice adopted in each situation. Moreover, comparing these logics with the literature is also in our agenda, namely to establish a formal relation between the models of **GP**(L) and the *Fuzzy Petri nets* [18].

The behaviour of Petri nets is concurrent by nature, being defined by the simultaneous firing of sets of transitions. However, in the method introduced, global composition is presented as a derived operator from the base (sequential composition, choice and reflexive transitive closure) Kleene operations. Hence, in future, we intend to adapt the presented construction to another parameter supporting a concurrent computational model like, for instance, a Concurrent Kleene Algebra [9]. An approach in this direction was already addressed in [5], which is based on the well known work on modal semirings [15]. Note however that, although such approach is capable of handling concurrency, it lacks the expressiveness for generating a logic able to reason in a multi-valued truth space.

References

1. Mario Benevides, Bruno Lopes, and Edward Hermann Haeusler. Propositional dynamic logic for Petri nets with iteration. In Augusto Sampaio and Farn Wang, editors, *ICTAC 2016*, volume 9965 of *LNCS*, pages 441–456. 2016.
2. W. J. Blok and Isabel Ferreirim. On the structure of hoops. *algebra universalis*, 43(2-3):233–257, 2000.

3. Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. The logical view on continuous Petri nets. *ACM Trans. Comput. Log.*, 18(3):24:1–24:28, 2017.
4. J.H. Conway. Printed in GB by William Clowes & Sons Ltd.
5. Han-Hing Dang and Bernhard Möller. Modal algebra and petri nets. *Acta Inf.*, 52(2-3):109–132, 2015.
6. Uffe Engberg and Glynn Winskel. Linear logic on Petri nets. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *A Decade of Concurrency Reflections and Perspectives*, pages 176–229. Springer, 1994.
7. Leandro Gomes, Alexandre Madeira, and Luis S. Barbosa. Generalising KAT to verify weighted computations. Technical Report submition under consideration.
8. David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. Foundations of Computing Series. MIT Press, 2000.
9. Tony Hoare, Stephan van Staden, Bernhard Möller, Georg Struth, Jules Villard, Huibiao Zhu, and Peter W. O'Hearn. Developments in concurrent kleene algebra. In Peter Höfner, Peter Jipsen, Wolfram Kahl, and Martin Eric Müller, editors, *RAMiCS 2014*, volume 8428 of *LNCS*, pages 1–18. Springer, 2014.
10. Dexter Kozen. On action algebras. *Logic and Information Flow*, pages 78–88, 1994.
11. Bruno Lopes, Mario Benevides, and Edward Hermann Haeusler. Extending Propositional Dynamic Logic for Petri Nets. *ENTCS*, 305(11):67–83, 2014.
12. Bruno Lopes, Mario Benevides, and Hermann Haeusler. Propositional dynamic logic for Petri nets. *Logic Journal of the IGPL*, 22:721–736, 2014.
13. Alexandre Madeira, Renato Neves, and Manuel A. Martins. An exercise on the generation of many-valued dynamic logics. *Journal of Logical and Algebraic Methods in Programming*, 1:1–29, 2016.
14. Alexandre Madeira, Renato Neves, Manuel A. Martins, and Luís Soares Barbosa. A dynamic logic for every season. In Christiano Braga and Narciso Martí-Oliet, editors, *SBMF 2014*, volume 8941 of *LNCS*, pages 130–145. Springer, 2014.
15. Bernhard Möller and Georg Struth. Modal kleene algebra and partial correctness. In Charles Rattray, Savi Maharaj, and Carron Shankland, editors, *AMAST 2004*, volume 3116 of *Lecture Notes in Computer Science*, pages 379–393. Springer, 2004.
16. David J. Pym, Peter W. O'Hearn, and Hongseok Yang. Possible worlds and resources: the semantics of BI. *Theor. Comput. Sci.*, 315(1):257–305, 2004.
17. Mario R. F. Benevides, Bruno Lopes, and Edward Hermann Haeusler. Propositional dynamic logic for Petri nets with iteration. In *ICTAC 2016*, volume 9965 of *LNCS*, pages 441–456, 2016.
18. Kai-Qing Zhou and Azlan Mohd Zain. Fuzzy Petri nets and industrial applications: a review. *Artif. Intell. Rev.*, 45(4):405–446, 2016.