Science of Computer Programming 157 (2018) 17–40

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Industrial applications of the PetriDotNet modelling and analysis tool

András Vörös [a,b,*], Dániel Darvas [a,1], Ákos Hajdu [a,b], Attila Klenik [a], Kristóf Marussy [a], Vince Molnár [a,b], Tamás Bartha [c], István Majzik [a]

[a] Department of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary
[b] MTA-BME Lendület Cyber-Physical Systems Research Group, Budapest, Hungary
[c] Institute for Computer Science and Control, Hungarian Academy of Sciences, Budapest, Hungary

ARTICLE INFO

Article history:
Received 30 December 2016
Received in revised form 5 July 2017
Accepted 11 September 2017
Available online 29 September 2017

Keywords:
Petri nets
Modelling
Simulation
Model checking
Stochastic analysis

ABSTRACT

Since their invention, Petri nets have provided modelling and analysis methods to support the design of correct, reliable and robust systems. This motivated our work to develop PetriDotNet, a Petri net editor and analysis tool. In this paper we overview the supported modelling formalisms and the analysis methods included in PetriDotNet. Next, we present eight different industrial case studies, demonstrating the wide variety of scenarios where Petri nets and PetriDotNet can help the design, development and analysis of industrial systems. Our original goal with PetriDotNet was to provide an educational tool to our students, however our efforts led to a framework being able to serve both academic and industrial needs.

1. Introduction

In the past decades, modelling and model-based analysis have taken a prominent role in the design of computer-based systems. The qualitative and quantitative design-time analysis of models provided early feedback for the developers, allowing them to check the functional correctness of the design, estimate extra-functional characteristics like performance or dependability, and evaluate design alternatives.

In this context, one of the most popular modelling formalisms was Petri nets, since its variants provided simple but powerful representations of the various aspects of the systems under design. The success of Petri nets was due to their straightforward syntax (with intuitive graphical representation) and semantics addressing especially concurrency and causality of events, extended later with the possibility to model (among others) data processing, timing and probabilistic behaviour. The analysis possibilities of basic Petri net models focused on checking the state space of the modelled system, including state reachability, invariants, general safety and liveness properties, as well as temporal logic model checking. Stochastic extensions allowed the evaluation of state probabilities and reward measures representing service quality (e.g. performance and dependability).

As it turned out, Petri nets with an appropriate editor and powerful analysis methods provide tools for the engineers to model, study and analyse various problems in system and software development. The modelling and analysis of event-driven

* Corresponding author at: Department of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary.
E-mail addresses: vori@mit.bme.hu (A. Vörös), darvas@mit.bme.hu (D. Darvas), hajdua@mit.bme.hu (Á. Hajdu), klenik@mit.bme.hu (A. Klenik), molnarv@mit.bme.hu (V. Molnár), tamas.bartha@sztaki.mta.hu (T. Bartha), majzik@mit.bme.hu (I. Majzik).

[1] The author was also affiliated with CERN (European Organization for Nuclear Research), Geneva, Switzerland during this project.

http://dx.doi.org/10.1016/j.scico.2017.09.003
0167-6423/

concurrent systems is well-supported by P/T nets and related analysis techniques: state reachability and temporal logic model checking provide useful means to verify the logical correctness of communicating systems such as control logics or communication protocols. Data processing can be efficiently modelled by coloured Petri nets, this way supporting the design of distributed algorithms and workflows. Invariant analysis, deadlock checking and model checking methods can be used to ensure the correctness of data-dependent behaviour. Stochastic modelling and analysis of state probabilities yield modelling and analysis of service quality properties of critical systems such as reliability, performance or availability.

In this paper we discuss PetriDotNet, a Petri net editor and analysis tool for P/T, coloured and stochastic Petri nets. Besides simulation and simple structural and dynamic analysis methods, the work of the last ten years resulted in many powerful analysis features. In our previous paper [1], we have focused on the educational use of PetriDotNet. Here we demonstrate that Petri nets and PetriDotNet can aid the design and development of industrial systems with its modelling and analysis features.

For this purpose, we define the Petri net variants supported by our tool and we give a short introduction to the main discrete and stochastic analysis features. Then, we showcase a wide variety of industrial use cases from different domains, including the verification of the safety logic of a nuclear power plant, analysis of power consumption of sensor nets, test input generation for communicating robots, analysis of public transport networks, studying railway interlocking systems, hazard rate calculation of a fault-tolerant automotive control system, analysis of an IaaS cloud, and an extensive benchmark on various models. In some applications the modelling features were exploited and in other cases the analysis features were more dominant; some cases demonstrate the usability of P/T nets, some others underline the advantages of coloured or stochastic net representations; some cases were done by the PetriDotNet development team and other applications by independent industrial practitioners.

This paper shows evidence of PetriDotNet being a convenient tool to design Petri net based analysis models and to evaluate them by performing simulation, discrete and stochastic analysis. A wide range of analysis capabilities is supported by PetriDotNet and case studies showed that we could model and verify systems which were not possible to be analysed before. The case studies also emphasise that the tool is competitive with other frameworks in this field.

Structure of the paper. The structure of this paper is the following. Section 2 introduces the modelling and requirement formalisation languages used in the article: different variants of Petri nets and temporal logics. Section 3 focuses on the current functionality of PetriDotNet and describes the discrete and stochastic analysis algorithms included in our tool. Next, we present industrial use cases in Section 4. The related work on industrial usage of Petri nets is presented in Section 5. Finally, Section 6 summarises and concludes the paper.

This article is an extended version of our previous conference paper [1]. The major novel contributions are the extensive description and discussion of analysis algorithms included in PetriDotNet (in Section 3), the detailed description of industrial cases where PetriDotNet was proven to be applicable and useful (Section 4) and the related work on the industrial usage of Petri nets (Section 5).

2. Background

This section introduces the modelling and requirement formalisation languages that are supported by PetriDotNet and are used in the rest of this paper. Section 2.1 describes three variants of Petri nets: P/T nets, coloured Petri nets and stochastic Petri nets. Section 2.2 presents languages for describing temporal requirements, and Section 2.3 is dedicated to quantitative requirements. Although the rest of the paper is more informal and the definitions are less frequently used, we still include them in this section to precisely describe the semantics of the supported modelling and requirement formalisms.

2.1. Variants of Petri nets

This section presents the variants of Petri nets that are supported by PetriDotNet, namely Petri nets (Section 2.1.1), coloured Petri nets (Section 2.1.2) and stochastic Petri nets (Section 2.1.3).

2.1.1. Petri nets

Petri nets are graphical models for describing concurrent, non-deterministic and asynchronous systems, providing both structural and dynamic analysis [2]. Formally, a Petri net (also referred to as Place/Transition net or P/T net[2]) is a tuple $\mathrm{PN} = (P, T, E, W, m_0)$, where

- P is the finite set of *places*,
- T is the finite set of *transitions* with $P \cap T = \emptyset$,
- $E \subseteq (P \times T) \cup (T \times P)$ is the finite set of *arcs*,
- $W : E \mapsto \mathbb{Z}^+$ is the *weight function* assigning weights $w^-(p, t)$ to edges $(p, t) \in E$ and $w^+(p, t)$ to edges $(t, p) \in E$, and
- m_0 is the *initial marking*.

[2] The name "Place/Transition net" is used in the following to emphasise that the Petri net is neither coloured, nor stochastic.

A *marking* of a Petri net is a mapping $m : P \mapsto \mathbb{N}$. A place p contains k *tokens* in a marking m if $m(p) = k$. A transition $t \in T$ is *enabled* in a marking m, if $m(p) \geq w^-(p,t)$ holds for each $p \in P$ with $(p,t) \in E$. An enabled transition t can *fire*, consuming $w^-(p,t)$ tokens from places $p \in P$ with $(p,t) \in E$ and producing $w^+(p,t)$ tokens in places $p \in P$ with $(t,p) \in E$. The firing of a transition t in a marking m is denoted by $m[t\rangle m'$ where m' is the marking after firing t.

A sequence of transitions $\sigma = t_1 t_2 \ldots t_n \in T^*$ is a *firing sequence*. A firing sequence is *realisable* in a marking m, leading to m' (denoted by $m[\sigma\rangle m'$), if $m[t_1\rangle \ldots [t_n\rangle m'$ holds. The *Parikh image* of a firing sequence σ is a vector $\wp(\sigma) : T \mapsto \mathbb{N}$, where $\wp(\sigma)(t_i)$ counts the number of the occurrences of t_i in σ.

The (infinite) set of all markings $m : P \mapsto \mathbb{N}$ is denoted by S. If there exists a realisable firing sequence σ such that $m_0[\sigma\rangle m$, then $m \in S$ is *reachable*. The set of all reachable markings is denoted by $S_R \subseteq S$.

The *incidence matrix* of a Petri net is a matrix C with size $|P| \times |T|$, where $C(p,t) = w^+(p,t) - w^-(p,t)$, i.e. the change in the number of tokens in p when firing t. Given a vector $x \in \mathbb{N}^{|T|}$, Cx denotes the change in the marking if each transition $t \in T$ is fired $x(t)$ times. Given two markings m and m' the *state equation* takes the form $m + Cx = m'$ and it describes the possible trajectories from m to m'. Due to the definition of the incidence matrix, the Parikh image $\wp(\sigma)$ of a realisable firing sequence with $m[\sigma\rangle m'$ fulfils the state equation. On the other hand, not all solutions of the state equation correspond to a realisable firing sequence. A solution x is *realisable* if a realisable firing sequence σ exists with $\wp(\sigma) = x$.

A vector $y \in \mathbb{N}^{|T|}$ is a *T-invariant* if $Cy = 0$ holds, i.e. firing the T-invariant does not change the marking. A vector $y \in \mathbb{N}^{|P|}$ is a *P-invariant* if $yC = 0$ holds, i.e. the weighted sum of tokens in the P-invariant is constant.

Petri nets with inhibitor arcs. Petri nets can be extended with a set $I \subseteq (P \times T)$ of *inhibitor arcs* to become a tuple $PN_I = (PN, I)$. The weight function is also extended to assign weights to inhibitor arcs: $W : (E \cup I) \mapsto \mathbb{Z}^+$. When inhibitor arcs are present, there is an extra condition for a transition $t \in T$ to be enabled: for each $p \in P$, if $(p,t) \in I$ then $m(p) < w^-(p,t)$ must hold. Inhibitor arcs increase the modelling power of Petri nets to be Turing complete [3].

2.1.2. Coloured Petri nets

Coloured Petri nets (CPN) provide a high-level language to develop formal models, yielding compact representations for complex systems. The coloured Petri net formalism enriches P/T nets with complex data structures [4]. There are many types of coloured Petri nets. In this paper a variant of well-formed coloured Petri nets is used that is supported by the PetriDotNet tool. Well-formed coloured Petri nets have the same expressive power as Place/Transition nets.

Coloured Petri nets usually have an associated action language (such as the CPN ML [4]) for defining colour sets (types), declaring variables and constants, and describing initial markings, guards and arc expressions. PetriDotNet supports a subset of the CPN ML language,[3] variables of finite ordered types are supported. Variables can be compared, traditional arithmetic operations (on integers) and also the successor operator can be used. Let *EXPR* denote the set of syntactically correct expressions provided by the language. Given an expression $e \in EXPR$, let $Type[e]$ denote its *type* (i.e. colour set) and $Var[e]$ its *free variables*.

Formally, a coloured Petri net [4] is a tuple $CPN = (P, T, A, \Sigma, V, C, G, E, I, M_0)$, where

- P is the finite set of *places*,
- T is the finite set of *transitions*, with $P \cap T = \emptyset$,
- $A \subseteq (P \times T) \cup (T \times P)$ is the finite set of *arcs*,
- Σ is the finite set of non-empty *colour sets*, i.e. types,
- V is the finite set of typed *variables* such that $\forall v \in V : Type[v] \in \Sigma$,
- $C : P \mapsto \Sigma$ is the *colour set function* assigning a colour set to each place,
- $G : T \mapsto EXPR$ is the *guard function* assigning a guard expression to each transition $t \in T$, with $Type[G(t)] = Bool$,
- $E : A \mapsto EXPR$ is the *arc expression function* assigning an expression to each arc a, with $Type[E(a)] = C(p)$ if a is connected to place p,
- M_0 is the *initial marking*.

A *marking* M is a function mapping each place p into a multiset of values $M(p)$ over the colour set $C(p)$, i.e. $M(p) : C(p) \mapsto \mathbb{N}$, defining the number of *tokens* of each permitted colour for each place.

Firing rules of transitions in P/T nets only depend on the marking. However, in coloured Petri nets the arc and guard expressions have to be considered as well. Let $Var[t]$ denote the variables of a transition $t \in T$: $Var[t] \triangleq Var[G(t)] \cup \bigcup_{p \in P} \{Var[E(a)] \mid a = (p,t) \in A \vee a = (t,p) \in A\}$, i.e. the variables appearing in the guard of t and in expressions on arcs connected to t. A *binding* b of a transition t assigns each variable $v \in Var[t]$ a value $b(v) \in Type[v]$. The set of all bindings for a transition t is denoted by $B(t)$. A *binding element* is a pair (t,b) of a transition t and a binding $b \in B(t)$. The set of all binding elements for a transition t is $BE(t) = \{(t,b) \mid b \in B(t)\}$. Given a binding element (t,b), let $G(t)\langle b\rangle$ denote the result of evaluating a guard in the binding b. Similarly, let $E(p,t)\langle b\rangle$ or $E(t,p)\langle b\rangle$ denote the result of evaluating arc expressions. If no such arcs exist, $E(p,t)\langle b\rangle = \emptyset$ or $E(t,p)\langle b\rangle = \emptyset$, respectively.

[3] The language supported by PetriDotNet is described in its user manual, which is available at http://petridotnet.inf.mit.bme.hu/releases/pdn1_manual.pdf.

A binding element (t, b) is *enabled* in a marking M if (1) $G(t)\langle b\rangle$ evaluates to true and (2) $E(p, t)\langle b\rangle \leq M(p)$ for each $p \in P$. An enabled binding element (t, b) may *fire*, leading to the marking M' defined by $M'(p) = M(p) - E(p, t)\langle b\rangle + E(t, p)\langle b\rangle$ for each $p \in P$, i.e. input arcs remove tokens, while output arcs produce tokens as specified by the arc expressions. Firing sequences and reachability are defined the same way as for P/T nets.

2.1.3. Stochastic Petri nets

Ensuring the correctness of critical systems, including safety-critical, distributed and cloud applications, requires the analysis of not only functional, but also extra-functional requirements. A large class of quantitative extra-functional properties, such as the dependability and performance of the system, can be addressed by stochastic modelling. In PetriDotNet, the *stochastic Petri net* (SPN) formalism is supported for the design and analysis of stochastic models.

Stochastic Petri nets extend P/T nets by assigning exponentially distributed random delays to transitions [5]. After the delay associated with an enabled transition is elapsed the transition fires and transition delays are reset. A stochastic Petri net (SPN) is a pair $\mathrm{SPN} = (\mathrm{PN}, \Lambda)$, where:

- $\mathrm{PN} = (P, T, E, W, m_0)$ is a Petri net. We will only consider the case when the Petri net is bounded, hence its reachable state space S_R is finite.
- $\Lambda\colon T \mapsto \mathbb{R}^+$ is a function that maps the set of transitions to transition rates.

The semantics of SPNs are defined as continuous-time Markov chains (CTMC). A (homogeneous) CTMC over the finite domain $\{0, 1, \ldots, n-1\}$ is a continuous-time stochastic process $X(t) \in \{0, 1, \ldots, n-1\}$, $t \in \mathbb{R}_{\geq 0}$ such that:

- The initial probability distribution vector of the system is $\boldsymbol{\pi}_0 \in [0, 1]^n$, such that $\sum_{x=0}^{n-1} \boldsymbol{\pi}_0[x] = 1$. By convention, boldface vector symbols refer to row vectors unless explicitly noted.
- The infinitesimal generator matrix $Q \in \mathbb{R}^{n\times n}$ describes the state transitions of the system.
  - Off-diagonal elements $q[x, y] > 0, x \neq y$ are the rate parameters of exponentially distributed delays after which the system in state x moves to state y. Zero off-diagonal elements indicate the absence of transitions.
  - The diagonal elements $q[x, x] = -\sum_{y=0, y\neq x}^{n-1} q[x, y] < 0$ are the negatives of the rate parameter of the exponential distribution corresponding to the sojourn time in the state x.
  - Thus $Q\mathbf{1}^{\mathrm{T}} = \mathbf{0}^{\mathrm{T}}$ holds.
- The time evolution of the probability distribution vector $\boldsymbol{\pi}(t) \in \mathbb{R}^n$, i.e. the state probabilities $\boldsymbol{\pi}(t)[x] = \Pr(X(t) = x)$, is described by the initial value problem

$$\boldsymbol{\pi}(0) = \boldsymbol{\pi}_0, \qquad \frac{d\boldsymbol{\pi}}{dt} = \boldsymbol{\pi} Q\,. \tag{1}$$

PetriDotNet implements a variety of analysis algorithms for CTMCs, which are described in Section 3.7.

The stochastic behaviour of the SPN is represented by the CTMC $X(t) \in \{0, 1, \ldots, n-1\}$, where $n = |S_R|$ is the number of reachable markings of the underlying Petri net.

- Transformation between the SPN and the continuous-time Markov chain is established by the bijection $\iota\colon S_R \mapsto \{0, 1, \ldots, n-1\}$, which maps the reachable markings to integers such that whenever the SPN is in the marking m at time t, $X(t) = \iota(m)$ holds. For simplicity we require that $\iota(m_0) = 0$.
- The initial probability distribution is $\pi_0[x] = 1$ if $x = 0 = \iota(m_0)$ and 0 otherwise.
- Off-diagonal elements $q[x, y]$ of the infinitesimal generator matrix Q contain the rate of exponentially distributed transitions from the marking $\iota^{-1}(x)$ to $\iota^{-1}(y)$. That is, Q is defined as

$$q[x, y] = \begin{cases} \sum_{\iota^{-1}(x)[e\rangle\iota^{-1}(y)}^{e\in T} \Lambda(e), & \text{if } x \neq y, \\ -\sum_{z=0, z\neq x}^{n-1} q[x, z], & \text{if } x = y. \end{cases}$$

In the variant of stochastic Petri nets described above, the exponential transition rates $\Lambda\colon T \mapsto \mathbb{R}^+$ are not allowed to depend on the marking of the net. This assumption is made in PetriDotNet to enable the block Kronecker decomposition of CTMCs with large state spaces, which is discussed in Section 3.7.

A further limitation of our tool pertains the analysable firing behaviours of transitions. Generalised Stochastic Petri Nets (GSPNs) extend SPNs by allowing transitions with immediate firing behaviour [6], which necessitates using semi-Markov processes as the semantics of the net. PetriDotNet only implements analysis for CTMCs, thus all transitions must have exponentially timed behaviour, i.e. the model has to be an SPN. Hence we only presented the formal definition of SPNs.

2.2. Temporal logic model checking

The purpose of *model checking* is to determine if the model of a system (a Petri net variant in our case) meets a given specification [7]. Behavioural properties are often expressed in temporal logic formalisms, which are capable of reasoning

about the order of events in a system using a logical time. There are two main categories based on the structure of logical time. Linear Temporal Logic (LTL) can describe a linear sequence of events, while branching time logics such as Computational Tree Logic (CTL) can describe all possible sequences [8]. They have different semantics and expressive power, for example deadlock freedom can be expressed only with CTL, while fairness properties are only expressible with LTL. It is desirable for a model checker to support both formalisms, however only a few of them do so. This allows the user to express and analyse a wide variety of different requirements, ranging from safety through fairness to liveness requirements. The details of the different temporal logic model checking algorithms are discussed together with our contributions, in Sections 3.4 and 3.5.

2.3. Stochastic analysis

Design time analysis of the quantitative aspects of critical systems is an important task in the verification of reliability and performance requirements. The analysis of stochastic properties is challenging as recent real-life systems yield huge state spaces and complex stochastic behaviour. Thus, no single approach or algorithm can handle the various challenges of industrial systems.

Stochastic analysis of complex systems requires the formulation of the engineering model to be studied as a stochastic model and its engineering properties of interest as stochastic reward measures or properties.

In PetriDotNet, stochastic models are continuous-time Markov chains that arise from stochastic Petri nets, which were reviewed in Section 2.1.3. Therefore, in the remainder of this section we will focus our attention to such models.

2.3.1. Rewards

Rewards allow modellers to capture quantitative aspects, such as dependability or performability, of a stochastic model as random variables. Formally, rewards can be described by stochastic reward nets. We also give some examples of the construction of rewards for algebraic expressions and CTL state predicates.

Reward rates. The stochastic reward net (SPN, r) extends the stochastic Petri net $\mathrm{SPN} = (\mathrm{PN}, \Lambda)$ with a reward process $R(t) \in \mathbb{R}$, which is defined by the reward rate function r,

$$r\colon S \mapsto \mathbb{R}, \qquad R(t) = r(\iota^{-1}(X(t))).$$

In other words, the *reward rate* random variable $R(t)$ is the function of the marking $m(t) = \iota^{-1}(X(t))$ of the SPN at time t. By taking the expected value $\mathbb{E}R(t)$, dependability measures, such as reliability and availability can be expressed, as well as performance measures, such as the mean service rate.

In PetriDotNet, the function $r\colon S \mapsto \mathbb{R}$ can be specified by *state rewards*. State rewards can be defined to associate a random variable with the number of tokens on a single place p, where $r_p(M) = M(p)$. We call these rewards *place rewards*.

A more sophisticated approach takes a CTL predicate ψ and an algebraic expression f of the token counts of places, yielding $r_{\psi,f}(M) = f(M)$ if $\psi(M)$, 0 otherwise, which we call *CTL rewards*.

Accumulated rewards. The *accumulated reward* process $Y(t)$ is obtained by the time integration of the reward rate $R(t)$, i.e.

$$Y(t) = \int_0^t R(\tau)\, d\tau.$$

The use cases of the expected accumulated reward $\mathbb{E}Y(t)$ include mean uptime until time t, mean number of serviced requests and mean accumulated cost.

Transition rewards. As an alternative to integrating reward rates, the accumulated reward $Y(t)$ can be defined by an *impulse reward* function $\rho\colon S \times T \mapsto \mathbb{R}$. In this formulation, $Y(0) = 0$ and Y is increased by $\rho(M, t)$ whenever the transition t is fired in marking M. This can be used to determine the number of times a certain action happens in a given time interval, for example, the number of requests successfully processed by a system.

We call the special case where $\rho_t(M, t) = const.$ for some transition $t \in T$, while 0 for all other transitions a *transition reward*.

It is also sometimes useful to define impulse rewards by CTL expression over the source and target states. Consider the CTL predicates ψ, ψ' and the algebraic expression f of the token counts of places. The *CTL event reward* $\rho_{\psi,\psi',f}(M, t)$ is equal to $f(M)$ if $\psi(M)$ and $\psi'(M')$ hold, where M' is the marking obtained after firing t, and is equal to 0 otherwise. Such rewards allow calculating how often a state transition occurs from a marking where ψ holds to a marking where ψ' holds.

2.3.2. Analysis types

Now we review some analysis tasks associated with stochastic reward nets. Solution algorithms for these analyses implemented in PetriDotNet are discussed in Section 3.7.

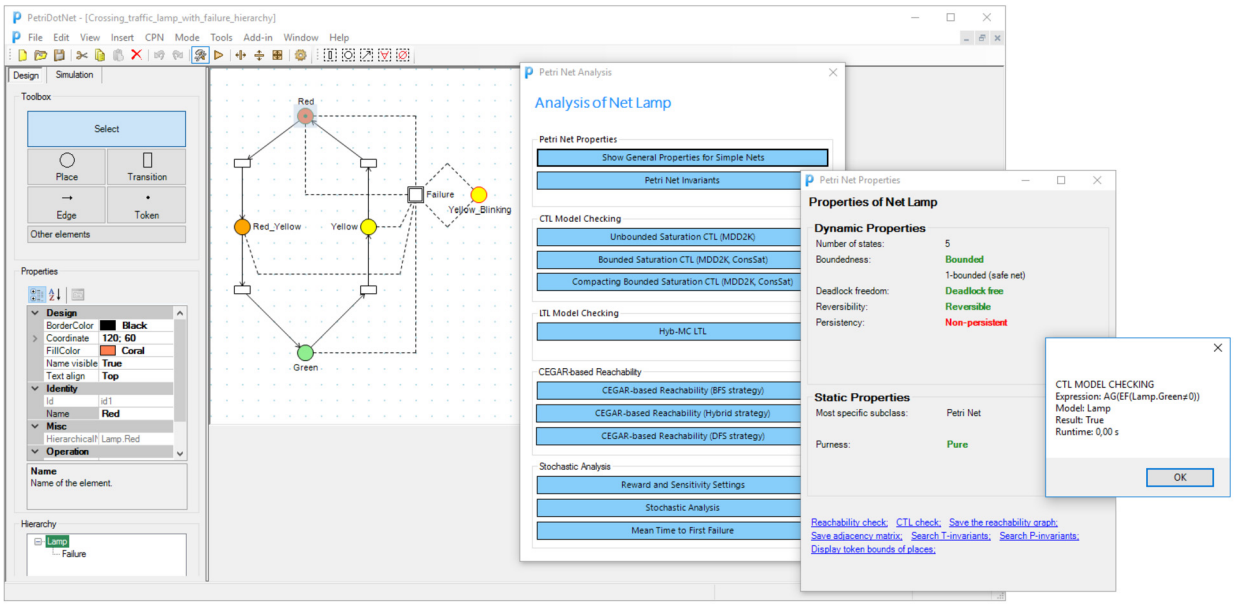

Fig. 1. The editor and some analysis windows of PetriDotNet.

Transient analysis. The *transient* analysis of stochastic models captures the transient behaviours of systems upon starting from an initial state. The expectations $\mathbb{E}R(t) = \boldsymbol{\pi}(t)\mathbf{r}^{\mathrm{T}}$ and $\mathbb{E}Y(t) = \mathbf{L}(t)\mathbf{r}^{\mathrm{T}}$ are calculated for some finite time horizon t after the start. The vector $\mathbf{r}$ enumerates the reward rates $r(M)$ in the order defined by the association between markings and CTMC states, while $\mathbf{L}(t) = \int_0^t \boldsymbol{\pi}(\tau)\,d\tau$ is the time integral of the probability distribution vector $\boldsymbol{\pi}(t)$ [9].

Steady-state analysis. In *steady-state* analysis, the long term-behaviour of the rewards is studied, which requires calculating the steady-state expected reward $\mathbb{E}R = \lim_{t\mapsto\infty} \mathbb{E}R(t)$.

If the steady state probability distribution vector

$$\boldsymbol{\pi} \in [0,1]^n, \quad \boldsymbol{\pi} Q = \mathbf{0}, \quad \boldsymbol{\pi}\mathbf{1}^{\mathrm{T}} = 1 \tag{2}$$

exists independently from the initial distribution $\boldsymbol{\pi}_0$, the mean steady-state reward can be obtained as $\mathbb{E}R = \boldsymbol{\pi}\mathbf{r}^{\mathrm{T}}$ [5].

Mean-time-to-state-partition. Another measure of interest is the time TFF to reach a state partition $D \subsetneq S_R$, which can be used to study the reliability of systems. The mean time to reach a state partition $\mathrm{MTFF} = \mathbb{E}[\mathrm{TFF}]$ can be calculated by noting that TFF has phase-type distribution [10]. In PetriDotNet, state partitions are specified by CTL predicates, similarly to CTL state and event rewards.

Sensitivity analysis. In parametric stochastic Petri nets and continuous-time Markov chains, transition rates may often depend on some vector of model parameters $\boldsymbol{\theta} \in \mathbb{R}^K$. By determining the *sensitivity* of the expected rewards $\mathbb{E}R$ to the parameters, i.e. its partial derivatives $\partial(\mathbb{E}R)/\partial(\theta[k])$, the parameters with the most influence over the behaviour of the model can be determined [11].

3. Features of PetriDotNet

This section overviews the editor and analysis features of the PetriDotNet tool. First, in Section 3.1 we focus on the editor and general features provided by the framework from the users' point of view. Section 3.2 is about the framework features for developers. After, the discrete analysis features are discussed: general Petri net analysis algorithms (Section 3.3), saturation-based CTL model checking (Section 3.4), LTL model checking (Section 3.5) and CEGAR-based reachability analysis (Section 3.6). Section 3.7 is dedicated to the stochastic analysis functionality of PetriDotNet.

3.1. Editor features

First and foremost, PetriDotNet is an editor for Petri nets. It provides graphical editing capabilities (cf. Fig. 1) for both P/T nets and well-formed coloured Petri nets (see Section 2.1.2 for the definition of the supported coloured Petri net variant). The tool also supports Petri nets extended with inhibitor arcs, transition priorities, and places with limited token capacity.

Moreover, the construction of hierarchical Petri nets is supported by allowing coarse transitions that can be refined by a subnet.

The tool provides simulation functionality (*token game*) for Petri nets, where the simulation can be manually conducted or automatically executed. The tool is shipped with a plug-in that can perform large scale simulation, executing thousands or millions of non-deterministic firings, and then present the statistics. These are essential features for the users during the model building phase, as they allow a deeper understanding of the models under development.

To save and load the Petri nets, PetriDotNet supports two formalisms natively. The default format is PNML (Petri Net Markup Language) [12], a standard, XML-based Petri net description format. PNML is supported by various other tools, therefore this is an interface between these tools and PetriDotNet. A binary, custom file format is also supported that provides more efficient persistence for large models.

Export and import features. It is possible to export the constructed Petri nets into other Petri net formalisms, such as to the syntax of the GPenSIM[4] (General Purpose Petri Net Simulator) and the .pnt format of the INA[5] (Integrated Net Analyser) tool. Also, the Petri net models can be translated into the input format of SAL[6] (Symbolic Analysis Laboratory). Furthermore, import is also provided from the .net textual Petri net file format used by the INA/Tina[7] tools, among others. New import or export plug-ins can be developed easily as the internal model representations are simply accessible.

3.2. Framework features for developers

Plug-in features. The functionality of the tool is extensible with plug-ins. Plug-ins can perform simulation tasks, provide analysis features (e.g. model checking) or export/import capabilities. Each plug-in can access the Petri net data models, use the graphical user interface, add new menu items, and call built-in PetriDotNet commands. The architecture of the tool is designed to keep the development of plug-ins simple, in order to help the users to focus on functionality instead of technology. For more details the reader is referred to our previous paper [1].

3.3. General Petri net analysis

The PetriDotNet framework includes essential algorithms for the analysis of general properties of P/T nets, such as the following.

- Dynamic Petri net properties [2] can be calculated based on the state space including *boundedness*, *deadlock-freedom*, *reversibility*, *persistence* and *reachability* of states. These properties can help to answer basic queries about the behaviour of the modelled system, including its safety and liveness.
- *Reachability/coverability graphs* [2] can be generated and exported into image files using a graphical representation. Reachability graphs contain the reachable markings $m \in S_R$ of the Petri net as nodes and transitions $t \in T$ between them as arcs. Coverability graphs are finite over-approximations of reachability graphs involving a special symbol ω to represent infinite number of tokens. These graphs can be used to illustrate and explore all possible behaviours of the modelled system.
- *Invariants* can be calculated and displayed directly on the Petri net. The invariant analysis covers both P and T-invariants based on the well-known Martinez–Silva algorithm [13], and a different algorithm by Cayir and Ucer [14] that computes the bases of invariants. T-invariants correspond to possible cyclic behaviours, whereas P-invariants imply that the weighted sum of tokens is constant (e.g. certain resources are constant in the modelled system).

These algorithms improve the usability of PetriDotNet both in education and for industrial use cases. Note however, that the set of reachable markings (i.e. the state space) can be exponentially or even infinitely large compared to the structural size of the Petri net. Therefore calculating dynamic properties and reachability graphs may not be suitable for industrial applications, but are very useful for educational purposes. More advanced algorithms (presented in the forthcoming sections) are available for efficient analysis of industrial problems.

3.4. Saturation-based CTL model checking algorithms

The CTL model checking algorithms evaluate a CTL expression on a given Petri net with a given initial marking. The main CTL quantifiers such as *all* and *exists* and operators such as *globally*, *future*, *next* and *until* are supported. Predicates of the CTL expression refer to markings of the Petri net. In addition, the model decomposition can also be provided or an invariant-based heuristic can be used. Based on this information the model checking algorithm evaluates if the model satisfies the requirements. In the following the underlying algorithms are discussed in more detail.

[4] http://www.davidrajuh.net/gpensim/.
[5] http://www2.informatik.hu-berlin.de/lehrstuehle/automaten/ina/.
[6] http://sal.csl.sri.com/.
[7] http://projects.laas.fr/tina/.

The traditional algorithm for CTL model checking is based on fixed point computations [15]: the evaluation of each CTL operator is reduced to fixed points reached through the state transitions. Each fixed point computation labels a set of states with a subformula of the CTL expression. Labelling is started from the deepest inner subformula and is continued from subformula to subformula. When the full formula is processed the result is the set of states satisfying the CTL expression. Model checking is then reduced to verifying if the initial states are contained in the result set of the fixed point computations.

Symbolic model checking. Symbolic model checking is a common remedy for the state space explosion problem, it helps to use the computation resources more efficiently than explicit model checking. In symbolic model checking, characteristic functions are used to encode sets of states and decision diagrams can be used for efficient storage. A decision diagram is a directed acyclic graph, representing a function. Various reduction rules ensure that decision diagrams are canonical and compact representation of a given function or set, which makes it a proper means to store sets of states. Traditional symbolic algorithms encode the reachable states and also the next state functions in decision diagrams. State variables are mapped to the variables of the decision diagram and state vectors are stored in the decision diagram as paths.

Saturation. Saturation is a symbolic algorithm for state space exploration and model checking. It relies on a decomposed model, where each Petri net place belongs to exactly one component. Each possible marking of a component is a local state. The global state (encoding a global marking of the net) is a composition of these local states: $s = (s_1, s_2, \ldots, s_K)$. With this decomposition the state space and the next-state function can be stored using multi-valued decision diagrams (MDDs) that are directed acyclic graphs encoding functions $D_1 \times \cdots \times D_K \mapsto \{0, 1\}$, thus can store sets of K-tuples.

The saturation algorithm exploits the locality of the decomposed model, making it especially suitable for models of asynchronous systems. The algorithm uses the following special iteration strategy: it starts with encoding the initial marking of the Petri net in the MDD. Then the nodes of the state space MDD are *saturated* (i.e. a local fixed point is computed w.r.t. the next-state function) in a bottom-up order, by applying saturation recursively whenever a new state is discovered. This helps to avoid the large peak size of the state space MDD [16].

The original saturation algorithm for state space exploration presented in [17] has been extended over the following years. These extensions include e.g. the on-the-fly exploration of local state spaces [18], the CTL model checking based on saturation [19], or the constrained saturation to make CTL model checking more efficient [20].

Our contributions. In PetriDotNet, various algorithms provide model checking based on saturation [21–23]. The CTL model checking approaches are based on the work of Zhao and Ciardo [20] and the bounded model checking approach is the extension of the method of Yu et al. [24]. Our research resulted in significant extensions and improvements in order to provide more efficient CTL model checking algorithms. This way PetriDotNet currently supports novel analysis algorithms as follows.

- CTL model checking of P/T nets and coloured Petri nets based on traditional and extended versions of saturation [16, 25]. For this, various improvements were developed on top of the original saturation algorithms to improve their performance. Furthermore, the saturation algorithm was extended to support CPNs, which turned out to be quite useful in practice (Section 4.1). This work led to the so-called lazy saturation, that improves the performance of the state space exploration of CPNs by reducing the size of the decision diagram representations.
- Bounded CTL model checking based on a novel saturation-based algorithm, with various search strategies [26,27]. This required the integration of the bounded state space exploration [24] with the CTL model checking algorithms. Further improvements were done in order to reduce the performance overhead of the bounded saturation [27].

3.5. On-the-fly hybrid model checking for LTL properties

Our LTL model checking module is used and parametrised similarly to the formerly introduced CTL model checking algorithm: the LTL model checker evaluates an LTL expression on a given Petri net and initial marking. The main LTL operators such as *globally*, *future*, *next* and *until* are supported. The algorithm also handles the *release* operator as the dual of *until*, as well as *weak next* as the dual of *next* [28], which provides the semantics for evaluating finite traces (*weak next* evaluates to *true* at the end of a finite trace, while *next* will be *false*). Predicates of the LTL expression refer to the markings of the Petri net. Saturation requires that the model has to be decomposed: a model decomposition can be provided by the user or an invariant-based heuristic can be used. In our measurements (cf. Section 4.8) we used model decompositions computed by external tools. Based on this information the model checking algorithm evaluates if the model satisfies the requirement. In the following the underlying algorithms are discussed in more detail.

The traditional steps to perform LTL model checking are summarised in Fig. 2 [29]. First, the negation of the desired property (given as an LTL expression) is translated into a Büchi automaton that accepts the execution traces violating the original property. In the next step, the product of this automaton and the state space of the model is computed to describe the set of violating traces that can actually emerge as the behaviour of the model. If the language of the product automaton is empty, it means that there is no behaviour in the model that can produce a violating trace, otherwise any such trace is a counterexample.

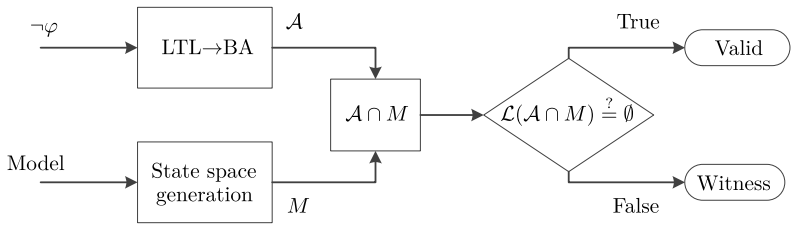

Fig. 2. Steps of automaton-based LTL model checking.

One of our most significant contributions to saturation-based model checking algorithms has been a novel algorithm for linear temporal model checking introduced in [30] and defined in detail in [31]. The motivation behind the new approach emerged from the need to combine the efficiency of saturation with the favourable properties of explicit LTL model checking – the on-the-fly or forward search, which enables the early termination of the algorithm as soon as a counterexample is found (see e.g. [32]).

Symbolic approaches usually follow the workflow described in Fig. 2 directly, i.e. they compute the set of reachable states in the product automaton [33,34] and apply least fixed point computations to obtain the reachable and accepting strongly connected components (SCCs) that imply the presence of counterexamples [35]. While this approach can handle large state spaces due to the symbolic techniques applied, it requires the complete exploration of the state space and thus suffers from state space explosion even if a simple counterexample exists. In contrast, on-the-fly explicit algorithms perform the state space generation, product computation, and emptiness check together in an incremental way.

To combine the benefits of both worlds, the following techniques were designed and applied to create a hybrid LTL model checking algorithm that looks for counterexamples on the fly. All of the following paragraphs describe original contributions that have not been defined or used in this way previously.

Product computation. As in the case of CTL model checking, we used saturation as a base algorithm. However, in case of LTL model checking, it is applied to explore the combined state space of the model and the specification automaton. It is not straightforward to keep the benefits of the algorithm, as the synchronised transitions typically lose locality, which is one of the most important properties that saturation strives to exploit. Our solution builds on the ideas of constrained saturation [20] to decompose the transition relation and to compute the synchronised transitions on the fly. A detailed description of the algorithm can be found in Section 4 of [31].

Incremental symbolic SCC computation. The key idea of our on-the-fly model checking approach is to stop the exploration periodically and execute an SCC computation to find accepting cycles quickly. With the existing algorithms, this approach would have an unacceptable overhead, therefore we designed a variant of the classic symbolic SCC-hull computation approaches that can reuse previous results, making it mostly incremental and greatly reducing the computational needs of consequent emptiness checks. The algorithm is designed for the iteration strategy of saturation, running every time when a node gets saturated (dividing the analysis into a series of alternating *exploration and SCC computation phases*). More details can be found in Section 5 of [31].

Recurring states heuristic. The repeated SCC computation is still costly even with the incremental symbolic algorithm, so it is worth to detect the situations when there is no chance of finding an accepting SCC. One of our heuristics is based on the following observation: any full traversal of a cycle will eventually find a state that has already been discovered – we call these states *recurring states*. We have modified the saturation algorithm to collect recurring states and use them as a cheap indicator of potential cycles: if no new recurring states were found since the last exploration phase, executing the symbolic SCC computation algorithm is not necessary.

Abstraction and local search heuristic. A more sophisticated heuristic uses abstraction to quickly decide if a set of states encoded by a decision diagram node may contain an SCC. The key idea is to project the system behaviour (transitions) to the currently processed component (see the componentwise processing of saturation), with the local states read from the latest saturated node. This approach is similar to those introduced in [36] and [37]. The resulting graph is sufficiently small to run explicit SCC computation algorithms on it, and the abstraction is defined such that if there is no cycle in the abstract graph, then there is no *new* SCC in the set of states encoded by the saturated node. The word *new* refers to the assumption that all child nodes have been analysed previously, thus the method is not concerned with any SCCs that could have been found in the previous phases. Details about the heuristics can be found in Section 6 of [31].

The complete workflow combining the new algorithms and heuristics is shown in Fig. 3. Experimental data shows that both heuristics are efficient in quickly computing the "no SCC" cases. In addition, they are orthogonal in the sense that there are cases for each when the other one fails to prove the lack of SCCs. The combined algorithm has the advantages of symbolic approaches in terms of compact state space representation and efficient traversal, as well as the ability of traditional explicit approaches to look for counterexamples on the fly, enabling early termination if a simple proof is available.

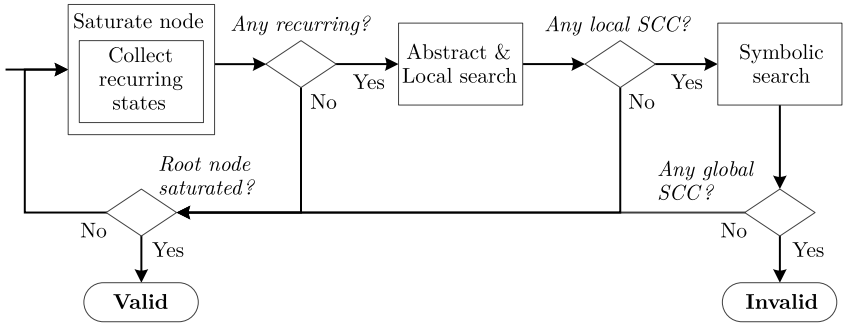

Fig. 3. SCC computation workflow with heuristics [31].

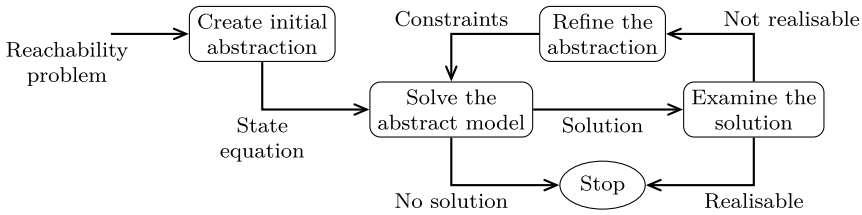

Fig. 4. The CEGAR workflow for Petri nets.

3.6. CEGAR-based reachability algorithms

Abstraction [38] is a general technique for reducing the complexity of a problem by hiding information that is not relevant for the solution. The main challenge in abstraction-based methods is to find the proper precision of abstraction that is coarse enough to tackle the complexity, but fine enough to be able to solve the problem. Counterexample-Guided Abstraction Refinement (CEGAR) [39] is an automatic algorithm that starts with an initial, coarse abstraction and refines it iteratively until the proper precision is obtained. CEGAR-based reachability algorithms can check if a given marking m' or a marking satisfying a set of linear constraints (e.g. $m(p_1) + 2m(p_2) \leq 5$) is reachable from the initial marking m_0 of a P/T net *PN*, possibly extended with inhibitor arcs. If the target marking is reachable, a firing sequence σ with $m_0[\sigma\rangle m'$ is also generated as a witness. These algorithms are incomplete (i.e. they may leave a problem undecided), but are also capable of handling nets with infinite state space in certain cases due to the nature of abstraction.

The CEGAR algorithm for Petri nets was first introduced in [40] and it served as the basis of our work [41,42]. This section presents the main concept of the original algorithm and our contributions as well. Fig. 4 presents an overview of the algorithm. The steps of the workflow are detailed below.

Initial abstraction. The input of the algorithm is a reachability problem, i.e. deciding whether some marking m' is reachable from the initial marking m_0 of a Petri net *PN*. The initial abstraction is the state equation in the form $m_0 + Cx = m'$. The state equation has to be solved for the vector $x \in \mathbb{N}^{|T|}$, where $x(t)$ denotes the number of times each transition $t \in T$ must fire in order to reach m' from m_0.

Solving the abstract model. Solving the abstract model (i.e. the state equation) is an integer linear programming (ILP) problem. The feasibility of the state equation is a necessary, but not sufficient condition for reachability, therefore if no solution exists, the target marking is not reachable. Otherwise, the solution must be checked whether it corresponds to a firing sequence in the original model (i.e. the Petri net).

Examining the solution. The solution of the state equation determines the number of times a transition $t \in T$ must fire in order to reach m' from m_0. However, x does not contain any information regarding the order of transitions or whether they are enabled. Therefore, the algorithm has to explore the state space of the Petri net with a limitation that each transition $t \in T$ can fire at most $x(t)$ times along a path. If m' can be reached with this limitation, x is called realisable and it is an obvious proof (witness) of reachability. In other words, the solution x is a Parikh image and the algorithm tries to find a corresponding realisable firing sequence σ with $\wp(\sigma) = x$.

Refining the abstraction. If a solution cannot be realised, the ILP solver is forced to generate a different solution, since there may be other, possibly realisable solutions of the state equation. This can be done by extending the state equation with additional *constraints* (i.e. linear inequalities over transitions) that rule out the non-realisable solution. The solution space is semi-linear, which means that each solution can be written as a sum of a *base solution* and the linear combination of T-invariants (solutions of the homogeneous part $Cx = 0$). Therefore, there are two types of constraints.

- *Jump constraints* have the form $x(t_i) < n$, where $n \in \mathbb{N}, t_i \in T$. Jump constraints can be used to obtain different base solutions, exploiting their pairwise incomparability.
- *Increment constraints* have the form $\sum n_i x(t_i) \geq n$, where $n_i \in \mathbb{Z}, n \in \mathbb{N}$ and $t_i \in T$. Increment constraints can be used to reach non-base solutions. This means that a new solution $x + y$ is obtained, where y is a T-invariant.

After adding the new constraint, the state equation either becomes infeasible or a new solution is obtained. At each non-realisable solution, multiple jump and/or increment constraints may be applied. Therefore, the algorithm traverses the solution space until a realisable solution is found (thus the reachability property is satisfied) or the state equation becomes infeasible and there are no more possibilities to backtrack (thus the reachability property is not satisfied).

Our contributions. From the theoretical point of view, we examined the algorithm [40] and proved that the heuristic calculation of new constraints is unsound by proposing an example net where the algorithm states that a reachable marking is unreachable. We suggested a fix for the heuristic by detecting the unsound situations [41]. We also investigated the completeness of the algorithm and presented some subclasses of nets where the algorithm could not decide the problem (yielding inconclusive answer or non-termination) due to its iteration strategy [41,42]. We introduced the concept of *distant invariants* [42] and some search space pruning techniques [41] to overcome this deficiency for some of the undecidable subclasses. We also investigated the theoretical limits of our new approaches [42].

From a practical point of view, we extended the algorithm to be able to handle new types of problems. An extension for *submarking coverability* allows us to define arbitrary linear constraints over the target marking [41]. Another extension allows us to handle *inhibitor arcs* [41]. However, as inhibitor arcs lift the modelling power of Petri nets to the level of Turing machines, this extension is obviously incomplete. Furthermore, our implementation in PetriDotNet includes various search strategies, adapted to the characteristics of the different models [42].

3.7. Stochastic analysis algorithms

Recently the tool was extended to support the modelling and analysis of stochastic Petri net models. The goal was to provide a configurable stochastic analysis framework where various state space exploration, matrix representation and numerical analysis algorithms can be combined [43].

The Markovian stochastic Petri nets (Section 2.1.3) supported in PetriDotNet have exponentially distributed transition firing rates that may optionally depend on model parameters. Measures of interest that can be expressed are state and impulse reward expressions. Impulse rewards are transformed into reward rates for mean reward calculation.

Our tool implements a general workflow that consists of:

1. Exploration of the possible behaviours of the system to construct its state space;
2. Construction of the generator matrix of a continuous-time Markov chain (CTMC) based on the state space and the exponential firing times of the model;
3. Numerical solution of the linear equation systems and differential equations arising from the CTMC;
4. Calculation of the specified reward measures based on the numerical solutions and reward expressions specified by the user.

Reward measures (see Section 2.3.1 for formal definitions) can be calculated as a sum of elementary rate and impulse rewards specified by CTL expressions and algebraic expressions of model parameters and the current marking. For simple problems, constant *place* rewards per token of a selected place per unit time are available as syntactic sugar, as well as constant *transition* rewards per firing of a selected transition.

Impulse rewards $\rho(M, t)$ are emulated as rate rewards $r(M)$ by setting the reward rate

$$r(M) = \sum_{t \in T,\, t \text{ is enabled in } M} \Lambda(t)\rho(M, t).$$

Therefore, solution algorithms were only implemented for rate rewards. Note that this simplification does not necessarily hold in the semi-Markov case. Hence immediate transitions are not supported and only stochastic Petri nets with exponential distributions can be analysed.

The analysis workflow can calculate with steady-state and transient analysis techniques (see Section 2.3.2 for the description of analyses) the mean reward rates $\mathbb{E}R$ in steady-state and $\mathbb{E}R(t)$ at time t. In addition, the mean accumulated transient reward $\mathbb{E}Y(t)$ is also computable. The partial derivatives of mean steady-state reward rates with respect to model parameters can be determined by sensitivity analysis. As a convenience, *complex* rewards, which are algebraic expressions of mean reward measures can be saved, e.g. to quickly calculate the ratio of two mean values.

In mean-time-to-state-partition analysis, state partitions can be specified by CTL expressions which is a convenient and expressive approach. The result of the analysis is the mean time to reach any of the state partitions, as well as the probability of reaching each partition first. This analysis is useful when state partitions represent failure modes of a modelled system and thus the mean time to first failure (MTFF) is to be computed. Calculation of sensitivity values to model the parameter changes is also incorporated.

The analyses described above can be run either standalone or as a parameter study. In parameter studies, the analysis can be run on a grid of model parameter values.

State space exploration and representation. Exploration of state spaces of stochastic models in PetriDotNet is provided by the symbolic analysis facilities of our tool. Evaluation of CTL expressions as part of reward expressions is facilitated by the tight integration between the stochastic and symbolic analysis capabilities of PetriDotNet. Explicit state space exploration is also available.

When symbolic state space exploration is selected, the set of reachable markings of the stochastic Petri net is represented as a Multivalued Decision Diagram. In addition, the mapping ι between markings and probability vector indices is represented as an Edge-Valued Additive Multivalued Decision Diagram (EV^+MDD).

Markov chain representation. The generator matrix of the continuous-time Markov chain associated with the stochastic Petri net may be stored as a dense or a sparse matrix. In addition, the block Kronecker decomposition form [44] is also available, which may achieve significant memory savings with little run time overhead if the model is appropriately partitioned.

Block Kronecker decomposition expresses the generator matrix Q as the sum of its off diagonal-part, which is a block matrix Q_O, and a diagonal matrix $\operatorname{diag}\mathbf{q}_D$,

$$Q = Q_O + \operatorname{diag}\mathbf{q}_D, \qquad Q_O = \left(Q_O[\tilde{x}, \tilde{y}]\right)_{\tilde{x},\tilde{y}=0}^{\tilde{n}-1,\tilde{n}-1}.$$

The indices $\tilde{x}$ and $\tilde{y}$ refer to partitions of the reachable state space that are called *macro states*.

Each block is a linear combination of Kronecker products

$$Q_O[\tilde{x}, \tilde{y}] = \sum_t \left(\lambda_t \cdot \bigotimes_j Q_t^{(j)}[\tilde{x}, \tilde{y}]\right),$$

where the factors are either sparse or identity matrices. Multiplication of a vector by a matrix in this form can be performed by the Shuffle algorithm [45].

The construction of dense and sparse matrices from the explicit or symbolic state space and the stochastic behaviours of the model is straightforward. On the other side, block Kronecker matrices first require state space decomposition. This state space decomposition, as well as mapping between the original and decomposed state indices of the model, is supported for both explicit [44] and symbolic [46] state space representations.

A case study on large models of IaaS clouds and the application of state space and Markov chain decompositions is presented in Section 4.7.

Numerical algorithms. Several numerical solvers are available in our framework. Implementations were done from scratch to support the expression tree-based generator matrices.

Steady state measures, which are the solutions to eq. (2), as well as and mean times to reach state partitions re calculated by solving usually sparse systems of linear equations.

Suitable solution algorithms in our framework are:

- Direct solution of the linear equation system by *LU decomposition*;
- Basic *power, Jacobi* and *Gauss–Seidel* stationary iterative solvers with optional overrelaxation;
- *Group* versions of *Jacobi* and *Gauss–Seidel* methods, in which subproblems arising from blocks along the diagonal of the generator matrix are solved by an inner linear equation solver; and
- The *BiCGSTAB* algorithm, which is a popular Krylov subspace method that offers a good compromise between convergence criteria, run time and memory requirements.

For a description of these solvers, we refer the reader to Chapters 4 and 7 of [47].

Transient solutions of CTMCs for immediate and accumulated distribution vectors are provided by the well-known *uniformisation* (or *randomisation*) algorithm as a solution to the initial value problem (1). In addition, the *TR-BDF2* implicit integrator is also part of the framework to handle *stiff* Markov chains that may take many iterations to solve with uniformisation [9]. Being an implicit method, TR-BDF2 requires the selection of a linear equation solver for the arising subproblems.

The built-in algorithms and configuration options allow the user to customise the analysis according to the model to be studied and the available computational resources. However, the outcome of the customisation, i.e. the performance of the constructed configuration relies on the developers experience and knowledge about the models/algorithms.

We also adapted the Krylov subspace solver IDR(s)STAB(ℓ) to our framework. By choosing the parameters s and ℓ appropriately, IDR(s)STAB(ℓ) becomes a common generalisation of several other methods, including BiCGSTAB(ℓ) and IDR(s) [48]. Thus, the run time and memory trade-off can be tuned according to the user's preference. However, despite our modifications to the algorithm in [49], convergence behaviour with CTMC generator matrices remained unsatisfactory, as their rank deficiency seems to pose an obstacle for the algorithm.

A case study on the applications of steady-state and mean-time-to-state-partition analysis to hazard rate calculations in automotive system is presented in Section 4.6. Moreover, we refer to [49] for additional benchmarks of the various solution algorithms.

Symbolic evaluation. As an extension of the expression tree approach for matrices, we added support for storing *elements* of the matrices and vectors involved in the analysis as algebraic expression trees. Instead of fixed values, mean reward measures are obtained as symbolic functions of model parameters.

In contrast with methods specific to parametric Markov chains, such as [50], in principle any analysis involving linear equation solvers can be adapted. Our tool symbolically computes the algebraic expressions of the model parameters as a solution for mean steady-state reward values and mean times to reach state partitions.

Unfortunately, in the symbolic setting, linear equation solvers can only depend on the nonzero structure of the matrices, as the numerical values of model parameters and hence the matrix elements themselves must be treated as unknowns. Therefore, only direct linear equation methods are available and no pivoting can be used. Therefore, in our tool, solving symbolically for algebraic expressions of the model parameters is carried out by LU decomposition.

To reduce memory requirements, expression trees need to be simplified by algebraic manipulations and constant folding during analysis. The user can select at which points this simplification should happen.

High-precision evaluation. Because pivoting and other techniques to improve numerical stability are unavailable in symbolic evaluation, catastrophic cancellation and underflow may occur during constant folding. To alleviate these problems, constants in expression trees can be represented by a .NET arbitrary-precision floating-point library[8] in addition to double-precision floating-point values.

For use in specialised scenarios, high-precision arithmetic is also supported in purely numerical evaluation. In this mode, matrix and vector elements are arbitrary-precision numbers and any linear equation solver can be run. Perhaps surprisingly, we found no improvement of the convergence behaviour of numerical algorithms when ported to the arbitrary-precision data structures compared to ordinary double-precision arithmetic. Therefore, this mode of operation is only of limited use in problems where many significant digits of mean reward values are desired.

Our contributions. Our contributions are twofold. From the practical point of view, we have extended the framework to provide configurable stochastic analysis. PetriDotNet provides a huge number of possible combinations of the various stochastic analysis algorithms.

From the theoretical point of view, we have bridged the gap between the symbolic state space representation and the various matrix representations of the stochastic behaviour. In addition, we compared the various configurations of the algorithms and we also experimented with the extended algorithm of IDR(s)STAB(ℓ). These findings were summarised in [49].

4. Industrial case studies

This section is dedicated to an overview of the various case studies where PetriDotNet was used as a modelling or analysis tool. In this paper we focus on the *industrial* applications of PetriDotNet. Information about the different educational uses of PetriDotNet can be found in [1]. Each use case is described briefly, and the reader is referred to the related publications for more information.

In the rest of the section, the following use cases are discussed:

- describes the application of PetriDotNet to model and formally verify a safety logic of a nuclear power plant using saturation-based CTL model checking;
- Section 4.2 presents a usage of PetriDotNet to model and simulate the power consumption of sensor nets;
- Section 4.3 reports about the R3-COP project,[9] where PetriDotNet was used to generate test input sequences for testing the robustness of communicating robots;
- Section 4.4 discusses a project in which urban public transport networks were modelled and analysed using Petri nets and PetriDotNet;
- Section 4.5 shows a case study where PetriDotNet was applied to model and study railway interlocking systems;
- Section 4.6 is dedicated to a case study of hazard rate calculations in a fault-tolerant automotive control system using the stochastic analysis capabilities of PetriDotNet;
- Section 4.7 describes the transformation of a complex stochastic model of an IaaS cloud for analysis with PetriDotNet; and
- Section 4.8 presents the performance of the LTL model checking algorithm of Section 3.5 on models of the Model Checking Contest (MCC), which includes not only benchmark models, but some industrial problems as well (such as a complex workflow model from IBM).

[8] https://peteroupc.github.io/Numbers/.
[9] http://www.r3-cop.eu/.

The models for the case studies in Sections 4.2, 4.5 and 4.8 were taken from the cited publications (or from their authors). All other models were constructed by the authors and their colleagues.

4.1. Verification of a safety procedure in a nuclear power plant

Background. In [51,52], Németh et al. described the modelling and verification of a safety procedure implemented for the Paks Nuclear Power Plant, Hungary. This safety procedure detects the so-called *primary-to-secondary leaking* (PRISE) event and initiates a safety response. This is a major fault of the nuclear reactor that initiates a reactor trip (emergency shutdown of the reactor) [51]. In addition, to avoid the release of contaminated water to the environment, the safety procedure activates specifically designed safety valves to drain the faulty circuits and prevent the release of contaminants. As the draining has a major impact on the plant, the safety procedure should initiate this action if and only if the PRISE event occurs. These serious consequences and the complexity of the logic requires a well-founded verification to make sure its adequate design and implementation.

The implementation of this safety procedure (PRISE safety logic) is included in the software of the reactor protection system [52]. The PRISE safety logic is described as a function block diagram (FBD), similar to the FBD formalism specified in IEC 61131-3. This description consists of interconnected blocks (function blocks, FBs) representing logic operations (AND, OR, NOT gates), stateful elements (e.g. RS flip-flops) and timer modules (TP pulse modules and TON delay module, as defined in IEC 61131-3). The authors of [52] have chosen coloured Petri nets as the modelling formalism. Their experience is that the "transformation [from FBD to CPN] was not straightforward, as the semantics of the FBs have a heterogeneous, semi-formal description consisting of truth tables, timing diagrams, together with textual information" [52]. Various analyses were used to check if there can be any false positive (spurious activation) or false negative (masked actuation) PRISE events indicated by the safety logic. However, due to the complexity of the logic, the analyses relying on the complete state space were not successful. Verifying the correctness of the safety logic was evaluated on a simplified model and using special analysis techniques manually.

Later work [53] compared various modelling and analysis tools (Design/CPN, UPPAAL, SAL) for this case. The authors report that Design/CPN and UPPAAL were not able to handle the complete state space. In case of SAL the verification was successful, but the modelling was difficult and needed various manual transformations, therefore it was much more error-prone than the other approaches.

Modelling in PetriDotNet. Based on the modelling principles defined in [52], we have provided a CPN model for the PRISE safety logic using PetriDotNet [16]. Modelling the safety logic was less convenient in PetriDotNet than using Design/CPN, as the latter is a much more mature modelling tool. However, the CPN formalism supported by PetriDotNet is expressive enough for this case. The CPN model follows the same structure as the original FBD model, therefore the model is easy to understand and maintain.

CTL model checking in PetriDotNet. The saturation-based CTL model checking algorithms extended to CPNs were able to explore and store the complete state space of the PRISE safety logic model [16]. The CTL formalism was rich enough to express the desired safety and liveness requirements and to show their satisfaction. The state space exploration of the model took a considerable amount of time, about 15 minutes. However, after the exploration phase, the evaluation of each CTL expression took less than 10 seconds. The size of the model varies depending on the chosen parameter values. The model used for the mentioned measurements contains 5×10^{12} states; 56 places, 62 transitions, 264 edges and 7 different colour sets.

Later, further improvements were made to the saturation algorithms targeting CPNs. Various minor improvements were made to the implementation of the algorithm to improve its efficiency. One of the weak points of the saturation applied to CPNs is the efficient construction of the next-state relation. In [25] we have proposed the so-called *lazy saturation* that allows the decomposition of the next-state relation into smaller, disjunct relations. It allows to reduce the overhead of the on-the-fly local state space exploration, i.e. the overhead caused by the frequent updates of the next-state relation, which is required every time a new local state is found. For more details of the lazy saturation we refer the reader to [25].

These improvements have successfully reduced the state space exploration time and memory consumption compared to the previous results reported in [16], to about 4 minutes from the previous 15 minutes.

Summary. The experiments with the verification of the PRISE safety logic have demonstrated that (1) CPNs are useful for modelling industrial modules, (2) the PetriDotNet tool is capable of modelling industrial modules, and (3) the novel saturation-based CTL model checking algorithms for CPNs make the verification of complex, industrial modules feasible.

4.2. Analysis of energy consumption in sensor networks

Background. Sensor networks are often considered as delay-tolerant networks. In these cases, the fastest possible transmission is not the topmost priority. Therefore, multiple messages can be aggregated, reducing the total amount of data to be transmitted (i.e. only one header is needed for the aggregated message, saving $n - 1$ headers in case of aggregating n messages). This improves the energy-efficiency, which is a major concern for the devices often operated by small-scale energy

sources. The goal of the work of Milánkovich et al. [54] is to find an optimal aggregation setting that results in minimal energy consumption, but satisfies the given delay requirements.[10]

Modelling and analysis in PetriDotNet. The paper [54] models a sensor network (chain of sensors) and analyses the effect of aggregation on energy consumption. For this purpose the stochastic modelling features of PetriDotNet were used. To determine the energy consumption, the authors calculate the stationary distribution of the model using the large-scale simulation feature of PetriDotNet. The authors state that "[t]he determination of the equilibrium distribution was carried out by the previously presented PetriDotNet software, because according to our tests, it was the fastest and it generated the best output results" [54].

To find the best configuration, Milánkovich et al. compared models with different aggregation number vectors. The elements of these vectors determine for each node in the sensor network the amount of data to be aggregated. For each aggregation vector a different SPN model is needed. Generating the permutations of aggregation number vectors and the corresponding SPN models, then executing PetriDotNet and processing the results were automated using Matlab. In addition, the authors used a C++ simulation to determine the total delay time to exclude the aggregation number vectors leading to unacceptable overall delays.

The authors of [54] report the summary of 3125 different aggregation settings in a given sensor network. Using the method described above they were able to find the aggregation numbers with the required delay and energy consumption characteristics. It was also shown that the same delay can be achieved by different aggregation settings having widely varying energy consumptions.

Summary. This case study demonstrates that (1) PetriDotNet can be used as an underlying analysis engine (in this case integrated into a workflow implemented in Matlab), and that (2) the simulation capabilities of PetriDotNet may provide useful information about characteristics of a (stochastic) model.

4.3. Test input sequence generation for laser-guided vehicles

Background. The PetriDotNet tool was also used in the Resilient Reasoning Robotic Co-operating Systems (R3-COP) project that aimed to "enable the production of advanced robust and safe cognitive, reasoning autonomously and co-operative robotic systems at reduced cost."[11] PetriDotNet and the underlying analysis methods were used to generate test input sequences for testing the robustness of communicating robots.

A demonstrator application in R3-COP used laser-guided vehicles (LGVs). LGVs are operated by a central computer that assigns distinct paths to them and ensures that the LGVs do not collide, provided that they follow the assigned paths. The goal of this part of the project was to test the communication protocols between the LGVs and the central computer.

Modelling and analysis in PetriDotNet. In one of the applied testing approaches we have developed a CPN model of the communication protocol in order to generate test input sequences: traces of the model (lists of events) that satisfy given requirements (formalised in CTL or as Boolean predicates). Some of the CPN models had simple structures: they had only 10–12 places and transitions, however they used complex colour sets, typically with two- or three-dimensions (i.e. the colour sets assigned to the places were cross products of two or three simple colour sets). In other cases the model size was more significant (with approximately hundred nodes), but the colour sets were simpler.

Three main scenarios were considered [55]:

- generation of a trace (and thus the corresponding test sequence) to a state satisfying a given requirement R,
- generation of a trace which goes through states satisfying requirements $R_1, R_2, \ldots, R_n$ in the given order,
- generation of a trace which goes through states satisfying requirements $R_1, R_2, \ldots, R_n$ in any order.

The first scenario allows the generation of a trace that reaches a specified protocol state defined by R. This is useful to test the functional correctness of the implementation by covering the relevant states defined by the test engineer.

The second scenario was used to force some specific states to occur in the trace: these specific states represented by $R_1, R_2, \ldots, R_n$ belonged to extreme behaviour with respect to the protocol operation, especially resulting from the sequence of processing invalid input data or boundary values. This way the robustness of the protocol implementation was tested.

The third scenario allows the specification of reaching various internal error states in the trace, this way testing the correctness of error handling and error processing in the protocol implementation.

In [55] we have proposed solutions for all three scenarios based on the bounded saturation-based CTL model checking algorithm. As the first scenario is a special case of the other two, in the following we assume that the trace shall go through multiple states satisfying multiple requirements.

[10] This section summarises the work of Milánkovich et al. based on [54]. The authors of the current paper were not involved in the development of this case study.

[11] http://www.r3-cop.eu/wp-content/uploads/2012/07/2012.07.04.Banner-R3-COP.pdf [Online, accessed: 07 Oct 2016].

The idea of these algorithms is to find a path from the initial state s_0 to a state s_{i_1} that satisfies R_1 (in the ordered case) or any requirement R_i (in the unordered case). The bounded CTL model checking algorithm is used to find a state that satisfies R_1 or some (one or more) expressions R_i. When such state is found, it is possible to extract the path (intermediate states, or required transition firings) between s_0 and s_{i_1} from the decision diagram representation of the state space. This is then performed iteratively, while there are more requirements to be satisfied by the trace: the trace generation continues from the last s_{i_j} state and looks for the next requirement to be satisfied (or for any requirement that is not satisfied yet, in the unordered case).

This is a greedy algorithm that does not necessarily find the optimal (shortest) trace. However in practice it was possible to generate traces satisfying the given requirements with acceptable lengths. Furthermore, the simple greedy algorithm was fast: in one of the cases reported in [55], the generation of a trace for 5 requirements (consisting of 63 states in total) on a model with 2.4×10^9 states required only 13 seconds. The generated traces (list of events) were later used as test input sequences.

Summary. This case study demonstrated that the combination and extension of pre-existing algorithms (e.g. bounded CTL model checking based on saturation, and saturation for CPNs) allow the application of PetriDotNet to solve novel verification tasks. In this case, the saturation algorithms implemented in PetriDotNet allowed to generate complex test sequences satisfying various requirements, making the testing of LGVs more efficient. The development of the presented algorithms was motivated by the R3-COP project, however they are now part of the PetriDotNet framework and can be reused for other use cases.

4.4. Modelling and analysis of public transportation networks

Background. The goal of an internship project at the company *evopro Innovation LLC* was to examine the possibility of Petri net-based modelling and analysis of public transportation networks. The main purpose of the analysis was to determine whether a network with a given topology and capacity can fulfil the travelling needs of the passengers. Modelling was done using the editor of PetriDotNet, while analysis was conducted using the CEGAR plug-in (Section 3.6).

Modelling in PetriDotNet. The basic concepts that had to be modelled were passengers, stops, lines and vehicles. In the first basic model, passengers were modelled by tokens and stops by places for the waiting passengers. Thus, the initial and target locations of the passengers could be represented by the marking of the Petri net. A vehicle was modelled by places representing its free capacity and places/transitions representing its route. A line was a collection of vehicles with the same route.

Moving on from the basic model, there were many modelling questions for more advanced, real-life concepts. For example, stops usually have two directions, vehicles have different types of places (e.g. sitting, standing). These problems could be solved in many different ways using the modelling possibilities of Petri nets. There was however, an interesting question related to vehicles and lines. As mentioned before, vehicles for the same line were copies of each other. A straightforward request was to be able to define the line only once and to instantiate it as many times as needed. Although PetriDotNet only supports hierarchical Petri nets to a limited extent by the means of *coarse* transitions, this construct provided a suitable solution. A line was modelled with a coarse transition and vehicles on that line were instantiations of the transition.

Analysis in PetriDotNet. The main goal of the analysis was to determine if the network modelled by the Petri net could fulfil the needs of the passengers, which were described by markings. Therefore, the most straightforward technique was to use reachability analysis. A further advantage of reachability analysis was that more advanced metrics could be calculated based on the firing sequence between the initial and target markings by a simple post-processing application. These metrics include for example the distance covered by vehicles, the minimal/maximal/average occupancy, the passenger traffic at each stop and the number of transfers between lines.

We created a sample model with 3 lines, 9 stops and 3 possibilities for changing lines, requiring around 50 places and 50 transitions. Explicit state space exploration algorithms could not handle the state space explosion problem, therefore we applied the CEGAR-based reachability algorithm. The algorithm was executed for various problems with different complexities. Unfortunately, the CEGAR-based algorithm could only solve the simpler instances within a reasonable amount of time. However, for a future work we identified some optimisation possibilities that could improve the performance of the algorithm based on certain domain-specific properties of the models.

Summary. Although the analysis results were partly negative, the case study still demonstrated that Petri nets may be capable of modelling public transportation networks with industrial relevance, especially with some of the advanced features of the PetriDotNet editor, such as coarse transitions. The case study also revealed that reachability analysis can provide useful metrics about the network, however, analysing networks with real-life size would require additional optimisations in the algorithms, motivating the need for improvement.

4.5. Modelling railway interlocking systems

Background. Various interlocking systems are in use to ensure the safe operation of railways. They mainly ensure that "a movement authority is given only when it is safe to do so", "points and other movable infrastructure are set to and locked in the correct position for any movements which are authorised over them or for which they provide protection" and the "interlocking and route locking associated with a movement authority is only released when it is safe to do so" [56].

In the Hungarian practice interlocking systems are mainly developed based on two principles: *tabular* (route-related) interlocking and *geographical* interlocking. The route-related principle uses the topology of the interlocking area (station or junction) and it builds on a given list of all possible routes and their dependencies. Based on this list, the elements of the route are locked together, at once. The geographical principle relies directly on the railway objects (signals, points, track sections, etc.) and their individual functionality. Each railway object representation is connected to its neighbours, and the communication between the start and end-point of a desired movement will build up the safe, locked route (in a "distributed" manner).

To compare these two principles, a group of domain experts[12] has developed Petri net models of interlocking systems for a selected station. These models include the main functionalities of the interlocking systems, such as setting and locking points; selecting, locking and releasing routes; setting signals; and shunting.

Modelling in PetriDotNet. Two Petri net models were developed using PetriDotNet, one for each interlocking system principle. As at this stage the modelling was done manually, only a small station was modelled. Therefore the resulting models contain approximately 100 nodes (places and transitions) each. According to the feedback of the domain experts, it was easy to learn and use PetriDotNet for the efficient modelling of the interlocking systems. The support for hierarchical models helped to structure the models, which may allow building models of larger interlocking areas in the future. The simulation (token game) feature provided useful feedback about the behaviour of the models.

Analysis in PetriDotNet. The key features of interlocking systems were checked on both models, for each possible route. For this purpose, the requirements were formalised in CTL. These requirements include the following examples: "A signal can only be cleared (set to a state permitting train movements) if all corresponding points are set and locked in correct position." or "While a signal is cleared, the corresponding points cannot be released or set to a different direction." These requirements were verified using the saturation-based CTL model checking included in PetriDotNet. The developers reported that the support for the full CTL formalism helps the formalisation of requirements. This is an advantage compared to UPPAAL (that supports only a CTL subset), however UPPAAL provides more help to the users in case of unsatisfied requirements, when an interactive counterexample is provided visually.

It was also reported that the performance of the saturation-based algorithms highly depend on the decomposition of the model. Without knowing the principles of the underlying model checker algorithms, it is difficult to provide a nearly optimal decomposition for the model. Currently only basic heuristics are provided for automated model decomposition in PetriDotNet. This underlines the need for better, more robust heuristics for model decomposition that would help the non-expert users and which is a planned future work.

A similar work was carried out and published independently by Cseh et al. in [57], which also used PetriDotNet as modelling and analysis tool. In that work the analysis relied on checking the structural and dynamic properties of the Petri nets, instead of using CTL model checking.

4.6. Hazard rate calculations in a fault-tolerant automotive control system

Background. The stochastic analysis capabilities of PetriDotNet were used in collaboration with an industrial partner to derive reliability measures for an automotive subsystem. The design of the system employs dual modular redundancy and self-checking mechanisms for fault tolerance.

Modelling in PetriDotNet. The subsystem was modelled using the SPN formalism. The possible events that can occur during operation, e.g. component faults, were mapped to transitions with exponentially distributed firing rates. Specification of event rates heavily relied on the model parameter support of our framework, which facilitated the quick reconfiguration of the model.

Critical sets of states $\{D_1, D_2, \ldots\}$ were specified by CTL expressions. These sets correspond to hazards with different severities. Upon reaching a hazard state the system is reset to its initial state. Measures of interest were the *hazard rates*, which are the frequencies of reaching the critical state partitions in the steady-state of the system according to system resets.

Analysis in PetriDotNet. Calculation of the hazard rates is possible by two of the stochastic analysis tools provided by PetriDotNet, which were described in Section 3.7. These approaches, which we used to determine hazard rates for the automotive system, can be applied to other models by the users of our tool.

[12] The modelling was performed by B. Farkas and G. Lukács.

Firstly, direct computation can be performed by steady-state analysis. This approach assigns an impulse reward to transitions that reset the system from the critical states D_i to the initial state. The expected steady-state value of this reward is equal to the hazard rate of reaching D_i.

Exploring and storing the state space and stochastic behaviour of the acquired model was feasible due to its modest size of 34 states. However, as some differences between the transition rates are several orders of magnitude, the obtained model is very *stiff*. Numerical solution of the steady-state problem was difficult, as well as approximation by transient analysis with a long time interval.

To reduce these problems, mean-time-to-state-partitions analysis was used instead, which was added to PetriDotNet after we face difficulties with steady-state hazard rate calculation. Because the model is reset to its initial state upon reaching any critical state $D = D_1 \cup D_2 \cup \cdots$, the mean time of reaching D can be interpreted as the common measure Mean Time to First Failure D (denoted as $MTFF(D)$), thus it is the expected value of a phase-type distributed random variable that can be derived from the model.

Let $HR(D_i)$ denote the hazard rate associated with the critical states D_i and $p(D_i|D)$ denote the conditional probability of residing in the state partition D_i upon reaching any critical state from D. The probability $p(D_i|D)$ can be calculated from the parameters phase-type distribution associated with the model and rates of transitions from noncritical states to D_i. As the hazard rate can be expressed as $HR(D_i) = \frac{MTFF(D)}{p(D_i|D)}$, calculation of the measure of interest is possible by mean-time-to-state-partition analysis instead of steady-state analysis. For models where this simplification is possible, mean-time-to-state-partitions analysis gives users a tool for hazard rate calculations even in stiff models.

The linear equation systems associated with the mean-time-to-state-partition analysis were simpler and had better numerical properties than the equivalent steady-state problems, which allowed analysis of the system with a wider range of parameters. Moreover, the modified analysis allowed simplified modelling by treating the critical states as deadlocks instead of adding reset transitions to the model. Resets were treated implicitly as part of the mean-time-to-state-partition analysis instead.

Parameter study and visualisation. The original hazard rate analysis was conceived as a parameter study. Around one million measure calculations in a predefined subspace of the parameter space were performed to find configurations with acceptable reliability measures.

The acquired results were used to visualise and analyse the safety metrics of the subsystem in an interactive web browser interface, which was created in R [58] with the Shiny web application framework.[13]

Inspired by this parameter study, additional parametric analysis features were added to PetriDotNet. Symbolic evaluation allows extracting parametric formulas of measures of interest from the model instead of numeric values. For the smallest models created for hazard rate calculation the resulting formulas can be used instead of repeated calculations for each parameter binding to obtain hazard rate values. However, for larger models, symbolic evaluation quickly became intractable as the size of the formulas grew.

Summary. This case study demonstrated that extensions to stochastic analysis, such as mean-time-to-state-partition calculation and symbolic evaluation, allow more robust analysis of extra-functional requirements and easier modelling of fault-tolerant systems. However, it also revealed scalability issues in steady-state analysis, hence symbolic evaluation will have to be addressed in a future work.

4.7. IaaS cloud scalability modelling

Background. Ghosh et al. [59] studied the performability, availability and resilience of an IaaS cloud system by generalised stochastic Petri nets (GSPNs). These models were stochastic Petri nets extended with immediate transitions, guards and state-dependent transition rates.

Instead of creating a monolithic GSPN, Ghosh employed the interacting sub-models approach to model an IaaS cloud, which is an approximation method offering considerable memory savings. Each sub-model was expressed as a parametric Markov chain depending on reward measures that can be extracted from the solutions of other sub-models. Approximate solutions to the monolithic model can be obtained by iteratively solving the parametric sub-models until a fixed point is reached.

Modelling in PetriDotNet. To validate and benchmark the stochastic analysis capabilities of PetriDotNet, we attempted to model and analyse the IaaS performability GSPN from [59]. As immediate transitions and guards are not available in the current version of PetriDotNet for stochastic analysis, a two-step process was needed to create compatible models.

First, the monolithic GSPN was translated into a textual domain-specific language created within the Eclipse Xtext[14] environment. Because not all features of GSPNs were implemented in the DSL, this step is unfortunately manual. For example, there were several restrictions of the immediate transitions and transition guards.

[13] http://www.rstudio.com/shiny/ [Online, Accessed 01 Nov 2016].
[14] http://www.eclipse.org/Xtext/ [Online, Accessed 01 Nov 2016].

After the manual translation of the model into a simpler form, a PNML file containing an ordinary stochastic Petri net was generated from the DSL. This model was loaded into PetriDotNet and analysed.

Analysis in PetriDotNet. The resulting model could serve as a large-scale benchmark of our stochastic analysis module, since the state space of the monolithic GSPN increased dramatically with the size of the modelled IaaS cloud. Additionally, it provided insights into the combination of model transformation techniques with stochastic analysis.

The IaaS cloud scalability model can be parameterised with the number of servers and the number of virtual machines per server, which affect the size of its state space. Due to the memory efficient representations of the state space and the stochastic behaviour, the analysis could run with huge state spaces with up to 2×10^7 states. State space decomposition could be applied to models up to 8×10^8 states, however, in those cases, numerical solution timed out.

In addition, this model revealed that by properly decomposing the model, block Kronecker representation can not only save memory, but can also reduce computation time.

However, the experiment also revealed several shortcomings of our stochastic analysis module. The lack of support for the GSPN formalism required manual intervention for model import. Moreover, even with the two-stage import process, state-dependent rates were lost from the original monolithic GSPN. Thus, the numerical results were not representative of the actual performance of the IaaS cloud. We are currently developing support for GSPNs with guards and state-dependent rates to alleviate these problems and increase the expressiveness of the models supported by our tool.

4.8. Model checking contest – performance evaluation of LTL model checking

Since the implementation of the LTL model checking algorithm is relatively new, it has not been used in any of our industrial projects so far. Nevertheless, we have evaluated its performance on a selection of models from the 2014 edition of the Model Checking Contest[15] (MCC), which includes some industrial problems among the many benchmark models (e.g. IBMB2S565S3960, the workflow model of a problem described by IBM). Since at that time, there were no verified correct solutions in the LTL track (and no tools competed in 2015 either), the implementation in PetriDotNet was compared to the following three competitor tools.

- NuSMV 2 [60] is a well-established tool that has a large variety of implemented algorithms, including BDD-based traditional SCC-hull algorithms. Before 2014, it was a popular choice for a reliable and mature tool for symbolic model checking. Measurements were conducted using the `check_ltlspec` command with flags `-df` (avoids the computation of all reachable states) and `-dcx` (disables the computation of a counterexample).
- nuXmv [61] is the successor of NuSMV. It was freshly released at the time of our experiments with some of the latest algorithms. It has been used with the K-liveness algorithm implemented on top of its IC3 engine (command `check_ltlspec_klive`), again without counterexample generation.
- ITS-LTL [37] is a powerful model checking tool using set decision diagrams (SDDs) [62] and saturation along with various model checking algorithms and optimisations. Back in 2014, it was the only tool combining saturation with LTL model checking, therefore it was selected as the main competitor.

All tools were provided with the same static and flat variable ordering (except nuXmv where the variable ordering is not applicable). The hierarchical ordering of ITS-LTL was not used in order to study the SCC computation capabilities of all tools under the same circumstances. Models of the MCC were given in the PNML format [63], which is not supported directly in any of the tools apart from PetriDotNet. Conversion to the CAMI format (used by the tool CPN-AMI) suitable for ITS-LTL was performed with a standard tool available online,[16] while the transformation from Petri nets to the SMV format supported by NuSMV and nuXmv was implemented based on the ideas described in [64].

At the time of the experiment, 27 scalable models were available from the MCC website (excluding those that had a coloured Petri net model only), yielding a total of 157 model instances. The SPOT toolset [65] was used to generate 50 LTL expressions for each model, producing a total of 7 850 benchmark cases. More details about the benchmark setting can be found in [31].

Timeout was set to 600 seconds for each individual measurement. In addition to model checking with the four tools, a simple state space generation using saturation was also executed for every model to provide an insight into the overhead and benefits of the on-the-fly model checking algorithm presented in Section 3.5. Execution times were measured by every tool individually. The results were visualised on scatter plots, which can be seen in Fig. 5.

The four plots compare the execution time of PetriDotNet to that of state space generation with saturation, ITS-LTL, NuSMV 2 and nuXmv, respectively. Each dot represents a benchmark case: the horizontal axis implying the execution time of PetriDotNet and the vertical axis belonging to the other tool or state space generation. This way, a dot above the diagonal line represents a case which PetriDotNet solved faster than its competitor. The borders of the diagram represent cases which could not be solved in 600 seconds (all in all, 1 474 cases remained unsolved by every tool).

[15] http://mcc.lip6.fr/2014/.
[16] http://pnml.lip6.fr/camipnml/.

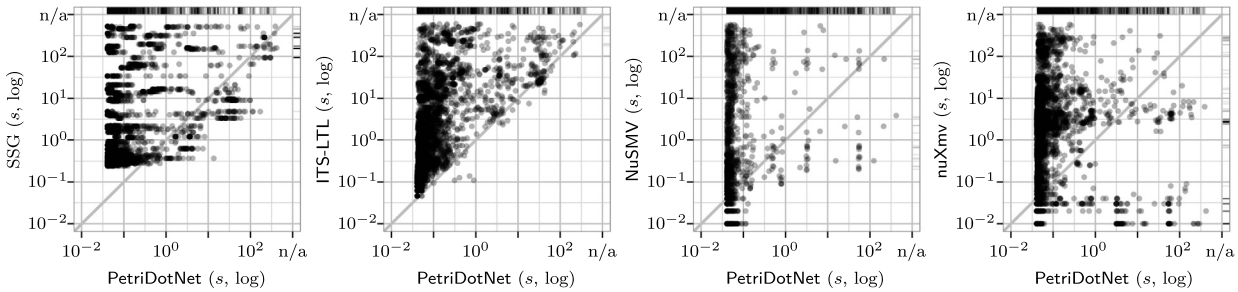

Fig. 5. Comparison of execution times for 3 tools and saturation-based exhaustive state space exploration on a selection of MCC models and randomly generated LTL expressions [31].

As the leftmost plot suggests, model checking with PetriDotNet is usually faster than the execution of state space generation only, mostly due to the efficiency of the incremental and on-the-fly algorithm that can be stopped in case a counterexample is found. Sometimes, state space generation was successful, but model checking was not: this shows that incremental operation and the optimisations presented in Section 3.5 cannot always compensate the overhead of model checking complex properties.

In terms of the other model checking tools, the vast majority of cases show the competitiveness of PetriDotNet. The scales of the axes are logarithmic, thus the distance of a dot from the diagonal indicate an exponential difference in the execution times of the tools. ITS-LTL showed the best performance among the competitors, but it could outperform PetriDotNet in very few cases only. Since it is also based on saturation and uses abstraction to perform on-the-fly model checking, these results are the most significant. Although NuSMV was usually the last to finish the computation, in some cases it was faster than PetriDotNet, even more significantly than ITS-LTL. This suggests that the advanced algorithms of ITS-LTL and PetriDotNet sometimes have a trade-off – "traditional" algorithms are still more suitable for certain problems. Results of nuXmv are very different from the results of other tools. This is due to the SAT-based algorithm it uses that has different techniques and strengths. It has outperformed PetriDotNet in most of the cases, even solving many that caused a timeout in PetriDotNet.

More details about the results can be found in [30,31]. The results and the benchmark package can be downloaded from our website.[17] All in all, the results were convincing and demonstrated the competitiveness of PetriDotNet in the model checking of LTL properties. Since that time, the LTL track of the Model Checking Contest has been populated by a number of tools, which offers an opportunity to participate with PetriDotNet and obtain neutral results – something to be expected in 2018.[18]

5. Related work – Petri nets in industrial case studies

Petri nets and their variants have been widely used to model and analyse real-life, industrial systems in various domains ranging from business process modelling to aeronautics. Many of these examples can be found in survey papers such as [66] and [67] or on the websites of the tools (e.g. CPnets[19]). The main purpose of our paper is to present our industrial applications of Petri nets in particular with the PetriDotNet tool. Therefore, as a related work we describe some recent industrial case studies conducted by other teams and tools. We present related case studies grouped by the type of Petri nets they use. We refer to similarities in our case studies and we also discuss if PetriDotNet could have been used in the same setting. However, we do not try to reproduce and compare the case studies as most of them lack details and artefacts are not available.

Place/Transition nets. Industrial case studies show that P/T nets already provide useful modelling and analysis possibilities. Stahl et al. transformed globally asynchronous locally synchronous circuits to Petri nets and reduced hazard detection to a reachability problem solved by the LoLa tool [68]. Talcott and Dill used Petri nets to answer reachability queries in biological processes interactively [69]. They implemented a custom tool (named Pathway Logic Assistant), which relies on LoLa. PetriDotNet also supports reachability analysis with its discrete analysis algorithms (Section 3). For example, in our case studies the modelling of public transportation networks (Section 4.4) targeted reachability and the verification of a safety procedure in a nuclear power plant (Section 4.1) also involved such queries. Fahland et al. checked a large number of industrial business process models for soundness (absence of deadlock, lack of synchronisation) using three different approaches, one of them based on Petri net CTL model checking of the LoLa tool [70]. They concluded that such checks can be performed in a few milliseconds, enabling a tight integration into the modelling process. PetriDotNet also supports efficient CTL model checking

[17] http://inf.mit.bme.hu/en/tacas15.
[18] Note that participation so far has been hindered by technical difficulties, as PetriDotNet currently runs on Windows only.
[19] http://cs.au.dk/cpnets/industrial-use/.

of P/T nets with the saturation algorithm (Section 3.4), as demonstrated in the railway interlocking case study (Section 4.5). Mendes et al. described services of a service-oriented architecture with Petri nets and proposed formal composition techniques to model more complex services [71]. Their experiment on a flexible transport system revealed that modelling the individual components could be done before specifying the control layout. Manyari et al. transformed discrete event system (DES) models of an oil production plant to labelled Petri nets and created a diagnoser that could recognise faults [72]. The previously discussed two case studies could not be solved with the base functionality of PetriDotNet. However, PetriDotNet makes the extension of its features possible by providing an easy-to-use plug-in interface. With some specific extensions PetriDotNet could be made suitable for such tasks.

Timed, hybrid, stochastic Petri nets. Timed, hybrid and stochastic extensions of Petri nets allow the analysis of non-functional and quantitative aspects. Bicchierai et al. integrated formal methods into an industrial software process by translating UML-MARTE (Unified Modelling Language Profile for Modelling and Analysis of Real-Time and Embedded Systems) specifications into pre-emptive time Petri nets [73]. They verified several non-functional requirements through state-space enumeration, including best/worst case completion time, deadlines and minimal laxity. Gaudel et al. used hybrid particle Petri nets (HPPN) to monitor the health status of the NASA K11 rover [74]. Based on the HPPN model, a diagnoser provided a distribution of belief over the possible health statuses even in an uncertain environment. Currently PetriDotNet does not support hybrid or timed nets.

Rogge and Weske proposed a method to predict remaining execution time of business processes using stochastic Petri nets with generally distributed transitions (GDT SPN) [75]. They applied their method at a Dutch logistics provider and showed that it was more accurate than earlier methods. Horváth used deterministic and stochastic Petri nets (DSPN) to analyse the production at a Hungarian window manufacturing company and to determine the throughput and bottleneck of the process [76]. Wan et al. built stochastic Petri net models of an integrated modular avionics (IMA) architecture and studied its health monitoring and fault management scheme [77]. They considered two performance metrics, namely availability and mean system response time. Amodeo et al. combined a genetic algorithm with a stochastic Petri net-based simulator tool to find an optimal inventory management policy under multiple objectives [78].

In general, these case studies rely on extensions to stochastic Petri nets. In the future, we plan to elaborate such complex semantics also in PetriDotNet.

Coloured Petri nets. Bergenthum and Shick used coloured Petri nets to reveal faulty processes of a medical laboratory based on log files [79]. They formed words (sequences) from events belonging to the same patients and split the language (i.e. the set of possible sequences) to valid and faulty words with the help of domain experts. Then, a coloured Petri net model was constructed where only valid words are executable. The CPN model was transformed into PL/SQL code in order to check for faulty processes directly in the log database. Gottschalk et al. proposed transformations from Protos – a popular tool for business process modelling – to coloured Petri nets [80]. Although Protos has a built-in simulation engine, the user cannot interact or inspect the simulations. The transformed models in contrast, can be executed by CPN Tools, using its extensive simulation and measurement features. State space exploration in coloured Petri nets is also often used for the analysis of internet protocols, for example the datagram congestion control protocol [81] and the dynamic on-demand routing protocol for mobile ad-hoc networks [82]. PetriDotNet was applied in case studies using coloured Petri nets (Section 4.1 and Section 4.3), demonstrating its modelling, simulation and analysis capabilities.

6. Summary and conclusion

This paper demonstrated that Petri nets and their variants, together with powerful, state-of-the-art analysis methods can help the design, development and analysis of various industrial systems. We presented various use cases, ranging from sensor nets, through transport systems, to cloud scalability, where Petri nets were found to be appropriate and helpful modelling solutions. With our new and improved analysis algorithms we were able to push the limits of the current state-of-the-art in formal methods and we could analyse systems which were too large or too complex to handle before. The use cases also demonstrated that PetriDotNet integrates the modelling power of Petri nets along with the analysis methods into a powerful Petri net editor and analysis tool.

Acknowledgements

The authors would like to thank all contributors who have worked on the implementation of PetriDotNet and the mentioned algorithms. We are especially grateful for the work of B. Szilvási, A. Jámbor, Z. Mártonka, D. Segesdi, and T. Szabó. We thank M. Telek for the assistance in the design of stochastic analysis algorithms.

This paper is partially supported by the MTA-BME Lendület 2015 Research Group on Cyber-Physical Systems and by the ARTEMIS JU and the Hungarian National Research, Development and Innovation Fund in the frame of the R5-COP (ARTEMIS-2013-1 nr. 621447) and R3-COP projects. Supported by the UNKP-16-2-I, New National Excellence Program of the Ministry of Human Capacities.

References

[1] A. Vörös, D. Darvas, V. Molnár, A. Klenik, A. Hajdu, A. Jámbor, T. Bartha, I. Majzik, PetriDotNet 1.5: extensible Petri net editor and analyser for education and research, in: F. Kordon, D. Moldt (Eds.), Application and Theory of Petri Nets and Concurrency, in: Lecture Notes in Computer Science, vol. 9698, Springer, 2016, pp. 123–132.

[2] T. Murata, Petri nets: properties, analysis and applications, Proc. IEEE 77 (4) (1989) 541–580, http://dx.doi.org/10.1109/5.24143.

[3] J.L. Peterson, Petri Net Theory and the Modeling of Systems, Prentice Hall, 1981.

[4] K. Jensen, L.M. Kristensen, Coloured Petri Nets: Modelling and Validation of Concurrent Systems, Springer, 2009.

[5] M.A. Marsan, Stochastic Petri nets: an elementary introduction, in: Advances in Petri Nets 1989, in: Lecture Notes in Computer Science, vol. 424, 1988, pp. 1–29.

[6] E. Teruel, G. Franceschinis, M. De Pierro, Well-defined generalized stochastic Petri nets: a net-level method to specify priorities, IEEE Trans. Softw. Eng. 29 (11) (2003) 962–973, http://dx.doi.org/10.1109/TSE.2003.1245298.

[7] E.M. Clarke, The birth of model checking, in: O. Grumberg, H. Veith (Eds.), 25 Years of Model Checking, in: Lecture Notes in Computer Science, vol. 5000, Springer, 2008, pp. 1–26.

[8] E.A. Emerson, J.Y. Halpern, "Sometimes" and "not never" revisited: on branching versus linear time temporal logic, J. ACM 33 (1) (1986) 151–178, http://dx.doi.org/10.1145/4904.4999.

[9] A.L. Reibman, R. Smith, K.S. Trivedi, Markov and Markov reward model transient analysis: an overview of numerical approaches, Eur. J. Oper. Res. 40 (2) (1989) 257–267, http://dx.doi.org/10.1016/0377-2217(89)90335-4.

[10] M.F. Neuts, Probability distributions of phase type, in: Liber Amicorum Prof. Emeritus H. Florin, University of Louvain, 1975, pp. 173–206.

[11] J.T. Blake, A.L. Reibman, K.S. Trivedi, Sensitivity analysis of reliability and performability measures for multiprocessor systems, in: Proc. of the 1988 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, 1988, pp. 177–186.

[12] ISO/IEC ISO/IEC, 15909-2 Systems and software engineering – High-level Petri nets – Part 2: Transfer format, 2011.

[13] J. Martínez, M. Silva, A simple and fast algorithm to obtain all invariants of a generalised Petri net, in: C. Girault, W. Reisig (Eds.), Application and Theory of Petri Nets, in: Informatik-Fachberichte, vol. 52, Springer, 1982, pp. 301–310.

[14] S. Cayir, M. Ucer, An algorithm to compute a basis of Petri net invariants, in: Proc. of the 4th Int. Conf. on Electrical and Electronics Engineering, 2005.

[15] E.M. Clarke, E.A. Emerson, Design and synthesis of synchronization skeletons using branching-time temporal logic, in: D. Kozen (Ed.), Logics of Programs, in: Lecture Notes in Computer Science, vol. 131, Springer, 1982, pp. 52–71.

[16] T. Bartha, A. Vörös, A. Jámbor, D. Darvas, Verification of an industrial safety function using coloured Petri nets and model checking, in: E. Ilie-Zudor, Z. Kemény, L. Monostori (Eds.), Proc. of the 14th Int. Conf. on Modern Information Technology in the Innovation Processes of the Industrial Enterprises, MITIP 2012, Hungarian Academy of Sciences, Computer and Automation Research Institute, 2012, pp. 472–485.

[17] G. Ciardo, G. Lüttgen, R. Siminiceanu, Saturation: an efficient iteration strategy for symbolic state-space generation, in: T. Margaria, W. Yi (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, in: Lecture Notes in Computer Science, vol. 2031, Springer, 2001, pp. 328–342.

[18] G. Ciardo, R. Marmorstein, R. Siminiceanu, Saturation unbound, in: H. Garavel, J. Hatcliff (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, in: Lecture Notes in Computer Science, vol. 2619, Springer, 2003, pp. 379–393.

[19] G. Ciardo, R. Siminiceanu, Structural symbolic CTL model checking of asynchronous systems, in: W.A. Hunt Jr., F. Somenzi (Eds.), Computer Aided Verification, in: Lecture Notes in Computer Science, vol. 2725, Springer, 2003, pp. 40–53.

[20] Y. Zhao, G. Ciardo, Symbolic CTL model checking of asynchronous systems using constrained saturation, in: Z. Liu, A.P. Ravn (Eds.), Automated Technology for Verification and Analysis, in: Lecture Notes in Computer Science, vol. 5799, Springer, 2009, pp. 368–381.

[21] G. Ciardo, R. Marmorstein, R. Siminiceanu, The saturation algorithm for symbolic state-space exploration, Int. J. Softw. Tools Technol. Transf. 8 (1) (2006) 4–25, http://dx.doi.org/10.1007/s10009-005-0188-7.

[22] G. Ciardo, A.J. Yu, Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning, in: D. Borrione, W. Paul (Eds.), Correct Hardware Design and Verification Methods, in: Lecture Notes in Computer Science, vol. 3725, Springer, 2005, pp. 146–161.

[23] G. Ciardo, Y. Zhao, X. Jin, Ten years of saturation: a Petri net perspective, in: K. Jensen, S. Donatelli, J. Kleijn (Eds.), Transactions on Petri Nets and Other Models of Concurrency V, in: Lecture Notes in Computer Science, vol. 6900, Springer, 2012, pp. 51–95.

[24] A.J. Yu, G. Ciardo, G. Lüttgen, Decision-diagram-based techniques for bounded reachability checking of asynchronous systems, Int. J. Softw. Tools Technol. Transf. 11 (2) (2009) 117–131, http://dx.doi.org/10.1007/s10009-009-0099-0.

[25] A. Vörös, D. Darvas, A. Jámbor, T. Bartha, Advanced saturation-based model checking of well-formed coloured Petri nets, Period. Polytech. Electr. Eng. Comput. Sci. 58 (1) (2014) 3–13, http://dx.doi.org/10.3311/PPee.2080.

[26] A. Vörös, D. Darvas, T. Bartha, Bounded saturation-based CTL model checking, Proc. Est. Acad. Sci. 62 (1) (2013) 59–70, http://dx.doi.org/10.3176/proc.2013.1.07.

[27] D. Darvas, A. Vörös, T. Bartha, Improving saturation-based bounded model checking, Acta Cybern. 22 (3) (2016) 573–589, http://dx.doi.org/10.14232/actacyb.22.3.2016.2.

[28] Z. Manna, A. Pnueli, Temporal Verification of Reactive Systems: Safety, Springer, 1995.

[29] M.Y. Vardi, An automata-theoretic approach to linear temporal logic, in: F. Moller, G. Birtwistle (Eds.), Logics for Concurrency, in: Lecture Notes in Computer Science, vol. 1043, Springer, 1996, pp. 238–266.

[30] V. Molnár, D. Darvas, A. Vörös, T. Bartha, Saturation-based incremental LTL model checking with inductive proofs, in: C. Baier, C. Tinelli (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, in: Lecture Notes in Computer Science, vol. 9035, Springer, 2015, pp. 643–657.

[31] V. Molnár, A. Vörös, D. Darvas, T. Bartha, I. Majzik, Component-wise incremental LTL model checking, Form. Asp. Comput. 28 (3) (2016) 345–379, http://dx.doi.org/10.1007/s00165-015-0347-x.

[32] G.J. Holzmann, D. Peled, M. Yannakakis, On nested depth first search, in: J. Grégoire, G.J. Holzmann, D. Peled (Eds.), The Spin Verification System, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 32, AMS, 1997, pp. 81–89.

[33] E.M. Clarke, O. Grumberg, K. Hamaguchi, Another look at LTL model checking, Form. Methods Syst. Des. 10 (1) (1997) 47–71, http://dx.doi.org/10.1023/A:1008615614281.

[34] R. Sebastiani, S. Tonetta, M.Y. Vardi, Symbolic systems, explicit properties: on hybrid approaches for LTL symbolic model checking, in: K. Etessami, S.K. Rajamani (Eds.), Computer Aided Verification, in: Lecture Notes in Computer Science, vol. 3576, Springer, 2005, pp. 350–363.

[35] F. Somenzi, K. Ravi, R. Bloem, Analysis of symbolic SCC hull algorithms, in: M.D. Aagaard, J.W. O'Leary (Eds.), Formal Methods in Computer-Aided Design, in: Lecture Notes in Computer Science, vol. 2517, Springer, 2002, pp. 88–105.

[36] C. Wang, R. Bloem, G.D. Hachtel, K. Ravi, F. Somenzi, Compositional SCC analysis for language emptiness, Form. Methods Syst. Des. 28 (1) (2006) 5–36, http://dx.doi.org/10.1007/s10703-006-4617-3.

[37] A. Duret-Lutz, K. Klai, D. Poitrenaud, Y. Thierry-Mieg, Self-loop aggregation product – a new hybrid approach to on-the-fly LTL model checking, in: T. Bultan, P. Hsiung (Eds.), Automated Technology for Verification and Analysis, in: Lecture Notes in Computer Science, vol. 6996, Springer, 2011, pp. 336–350.

[38] E.M. Clarke, O. Grumberg, D.E. Long, Model checking and abstraction, ACM Trans. Program. Lang. Syst. 16 (5) (1994) 1512–1542, http://dx.doi.org/10.1145/186025.186051.

[39] E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement for symbolic model checking, J. ACM 50 (5) (2003) 752–794, http://dx.doi.org/10.1145/876638.876643.
[40] H. Wimmel, K. Wolf, Applying CEGAR to the Petri net state equation, in: P.A. Abdulla, K.R.M. Leino (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, in: Lecture Notes in Computer Science, vol. 6605, Springer, 2011, pp. 224–238.
[41] A. Hajdu, A. Vörös, T. Bartha, Z. Mártonka, Extensions to the CEGAR approach on Petri nets, Acta Cybern. 21 (3) (2014) 401–417, http://dx.doi.org/10.14232/actacyb.21.3.2014.8.
[42] A. Hajdu, A. Vörös, T. Bartha, New search strategies for the Petri net CEGAR approach, in: R. Devillers, A. Valmari (Eds.), Application and Theory of Petri Nets and Concurrency, in: Lecture Notes in Computer Science, vol. 9115, Springer, 2015, pp. 309–328.
[43] A. Klenik, K. Marussy, Configurable Stochastic Analysis Framework for Asynchronous Systems, Scientific Students' Associations Report, Budapest University of Technology and Economics, 2015, http://petridotnet.inf.mit.bme.hu/publications/TDK2015_KlenikMarussy.pdf.
[44] P. Buchholz, Hierarchical structuring of superposed GSPNs, IEEE Trans. Softw. Eng. 25 (2) (1999) 166–181, http://dx.doi.org/10.1109/32.761443.
[45] P. Buchholz, G. Ciardo, S. Donatelli, P. Kemper, Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models, INFORMS J. Comput. 12 (3) (2000) 203–222, http://dx.doi.org/10.1287/ijoc.12.3.203.12634.
[46] K. Marussy, A. Klenik, V. Molnár, A. Vörös, I. Majzik, M. Telek, Efficient decomposition algorithm for stationary analysis of complex stochastic Petri net models, in: F. Kordon, D. Moldt (Eds.), Application and Theory of Petri Nets and Concurrency, in: Lecture Notes in Computer Science, vol. 9698, 2016, pp. 281–300.
[47] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, 2003.
[48] G.L. Sleijpen, M.B. van Gijzen, Exploiting BiCGstab(ℓ) strategies to induce dimension reduction, SIAM J. Sci. Comput. 32 (5) (2010) 2687–2709, http://dx.doi.org/10.1137/090752341.
[49] K. Marussy, A. Klenik, V. Molnár, A. Vörös, M. Telek, I. Majzik, Configurable numerical analysis for stochastic systems, in: E. Ábrahám, S. Bogomolov (Eds.), 2016 Int. Workshop on Symbolic and Numerical Methods for Reachability Analysis, SNR, IEEE, 2016.
[50] E.M. Hahn, H. Hermanns, L. Zhang, Probabilistic reachability for parametric Markov models, Int. J. Softw. Tools Technol. Transf. 13 (1) (2011) 3–19, http://dx.doi.org/10.1007/s10009-010-0146-x.
[51] E. Németh, T. Bartha, C. Fazekas, K.M. Hangos, Verification of a primary-to-secondary leaking safety procedure in a nuclear power plant using coloured Petri nets, Reliab. Eng. Syst. Saf. 94 (5) (2009) 942–953, http://dx.doi.org/10.1016/j.ress.2008.10.012.
[52] E. Németh, T. Bartha, Formal verification of safety functions by reinterpretation of functional block based specifications, in: D. Cofer, A. Fantechi (Eds.), Formal Methods for Industrial Critical Systems, in: Lecture Notes in Computer Science, vol. 5596, Springer, 2009, pp. 199–214.
[53] Z. Tóth Heinemann, Modelling and Verification of Discrete Industrial Control Systems Using Formal Methods [in Hungarian, original title: Diszkrét ipari irányítórendszerek modellezése és ellenőrzése formális módszerekkel], Master's thesis, Budapest University of Technology and Economics, 2009.
[54] A. Milánkovich, G. Ill, K. Lendvai, S. Imre, S. Szabó, Evaluation of energy efficiency of aggregation in WSNs using Petri nets, in: O. Postolache, M. van Sinderen, F. Ali, C. Benavente-Peces (Eds.), Proc. of the 3rd Int. Conf. on Sensor Networks, Science and Technology Publications, 2014, pp. 289–297.
[55] D. Darvas, A. Vörös, Saturation-based test input generation using coloured Petri nets [in Hungarian, original title: Szaturációalapú tesztbemenet-generálás színezett Petri-hálókkal], in: Mesterpróba 2013, 2013, pp. 48–51.
[56] Rail Safety and Standards Board, GK/RT0060 – Interlocking principles (2003). http://www.rssb.co.uk/rgs/standards/GKRT0060%20Iss%204.pdf.
[57] A. Cseh, G. Tarnai, B. Sághi, Petri net modelling of signalling systems [in Hungarian, original title: Biztosítóberendezések modellezése Petri-hálókkal], Vezetékek Világa XIX (1) (2014) 14–17.
[58] R Development Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, 2008.
[59] R. Ghosh, Scalable Stochastic Models for Cloud Services, Ph.D. thesis, Duke University, 2012, http://hdl.handle.net/10161/6110.
[60] A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV 2: an opensource tool for symbolic model checking, in: E. Brinksma, K.G. Larsen (Eds.), Computer Aided Verification, in: Lecture Notes in Computer Science, vol. 2404, Springer, 2002, pp. 359–364.
[61] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, M. Mariotti, A. Micheli, S. Mover, M. Roveri, S. Tonetta, The NuXmv Symbolic Model Checker, Tech. rep., Fondazione Bruno Kessler, 2014.
[62] A. Hamez, Y. Thierry-Mieg, F. Kordon, Hierarchical set decision diagrams and automatic saturation, in: K.M. van Hee, R. Valk (Eds.), Applications and Theory of Petri Nets, in: Lecture Notes in Computer Science, vol. 5062, Springer, 2008, pp. 211–230.
[63] L.M. Hillah, E. Kindler, F. Kordon, L. Petrucci, N. Treves, A primer on the Petri net markup language and ISO/IEC 15909-2, Petri Net Newsl. 76 (2009) 9–28, http://www.pnml.org/papers/pnnl76.pdf.
[64] M. Szpyrka, A. Biernacka, B. Jerzy, Methods of translation of Petri nets to NuSMV language, in: L. Popova-Zeugmann (Ed.), Concurrency, Specification and Programming, in: CEUR Workshop Proceedings, vol. 1269, 2014, pp. 245–256, http://ceur-ws.org/Vol-1269/paper245.pdf.
[65] A. Duret-Lutz, D. Poitrenaud, SPOT: an extensible model checking library using transition-based generalized Büchi automata, in: Proc. of the IEEE Int. Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, IEEE, 2004, pp. 76–83.
[66] R. Zurawski, M.C. Zhou, Petri nets and industrial applications: a tutorial, IEEE Trans. Ind. Electron. 41 (6) (1994) 567–583, http://dx.doi.org/10.1109/41.334574.
[67] X. Zhang, Q. Lu, T. Wu, Petri-net based applications for supply chain management: an overview, Int. J. Prod. Res. 49 (13) (2011) 3939–3961, http://dx.doi.org/10.1080/00207543.2010.492800.
[68] C. Stahl, W. Reisig, M. Krstic, Hazard detection in a GALS wrapper: a case study, in: J. Desel, Y. Watanabe (Eds.), Fifth Int. Conf. on Application of Concurrency to System Design, IEEE, 2005, pp. 234–243.
[69] C. Talcott, D.L. Dill, The pathway logic assistant, in: Third Int. Workshop on Computational Methods in Systems Biology, vol. 3, University of Edinburgh, 2005, pp. 228–239.
[70] D. Fahland, C. Favre, J. Koehler, N. Lohmann, H. Völzer, K. Wolf, Analysis on demand: instantaneous soundness checking of industrial business process models, Data Knowl. Eng. 70 (5) (2011) 448–466, http://dx.doi.org/10.1016/j.datak.2011.01.004.
[71] J.M. Mendes, P. Leitão, F. Restivo, A.W. Colombo, Composition of Petri nets models in service-oriented industrial automation, in: 8th IEEE Int. Conf. on Industrial Informatics, IEEE, 2010, pp. 578–583.
[72] M. Manyari-Rivera, J.C. Basilio, A. Bhaya, Integrated fault diagnosis based on Petri net models, in: 2007 IEEE Int. Conf. on Control Applications, 2007, pp. 958–963.
[73] I. Bicchierai, G. Bucci, L. Carnevali, E. Vicario, Combining UML-MARTE and preemptive time Petri nets: an industrial case study, IEEE Trans. Ind. Inform. 9 (4) (2013) 1806–1818, http://dx.doi.org/10.1109/TII.2012.2205399.
[74] Q. Gaudel, P. Ribot, E. Chanthery, M.J. Daigle, Health monitoring of a planetary rover using hybrid particle Petri nets, in: F. Kordon, D. Moldt (Eds.), Applications and Theory of Petri Nets and Concurrency, in: Lecture Notes in Computer Science, vol. 9698, Springer, 2016, pp. 196–215.
[75] A. Rogge-Solti, M. Weske, Prediction of remaining service execution time using stochastic Petri nets with arbitrary firing delays, in: S. Basu, C. Pautasso, L. Zhang, X. Fu (Eds.), Service-Oriented Computing, in: Lecture Notes in Computer Science, vol. 8274, Springer, 2013, pp. 389–403.
[76] Á. Horváth, Usability of deterministic and stochastic Petri nets in the wood industry: a case study, in: T. Do van, H.A.L. Thi, N.T. Nguyen (Eds.), Advances in Intelligent Systems and Computing, in: Advanced Computational Methods for Knowledge Engineering, vol. 282, Springer, 2014, pp. 119–127.
[77] J. Wan, X. Xiang, X. Bai, C. Lin, X. Kong, J. Li, Performability analysis of avionics system with multilayer HM/FM using stochastic Petri nets, Chin. J. Aeronaut. 26 (2) (2013) 363–377, http://dx.doi.org/10.1016/j.cja.2013.02.014.

[78] L. Amodeo, H. Chen, A. El Hadji, Supply chain inventory optimisation with multiple objectives: an industrial case study, in: A. Fink, F. Rothlauf (Eds.), Advances in Computational Intelligence in Transport, Logistics, and Supply Chain Management, in: Stud. Comput. Intell., vol. 144, Springer, 2008, pp. 211–230.
[79] R. Bergenthum, J. Schick, Verification of logs – revealing faulty processes of a medical laboratory, in: M. Koutny, J. Desel, S. Haddad (Eds.), Transactions on Petri Nets and Other Models of Concurrency X, in: Lecture Notes in Computer Science, vol. 9410, Springer, 2015, pp. 1–18.
[80] F. Gottschalk, W.M. van der Aalst, M.H. Jansen-Vullers, H. Verbeek, Protos2CPN: using colored Petri nets for configuring and testing business processes, Int. J. Softw. Tools Technol. Transf. 10 (1) (2008) 95–110, http://dx.doi.org/10.1007/s10009-007-0055-9.
[81] S. Vanit-Anunchai, J. Billington, G.E. Gallasch, Analysis of the datagram congestion control protocol's connection management procedures using the sweep-line method, Int. J. Softw. Tools Technol. Transf. 10 (1) (2008) 29–56, http://dx.doi.org/10.1007/s10009-007-0050-1.
[82] K.L. Espensen, M.K. Kjeldsen, L.M. Kristensen, Modelling and initial validation of the DYMO routing protocol for mobile ad-hoc networks, in: K.M. van Hee, R. Valk (Eds.), Applications and Theory of Petri Nets, in: Lecture Notes in Computer Science, vol. 5062, Springer, 2008, pp. 152–170.