# HYPEG: Statistical Model Checking for hybrid Petri nets

Tool Paper

Carina Pilch
Westfälische Wilhelms-Universität
Münster, Germany
carina.pilch@uni-muenster.de

Fabian Edenfeld
Westfälische Wilhelms-Universität
Münster, Germany
f.edenfeld@uni-muenster.de

Anne Remke
Westfälische Wilhelms-Universität
Münster, Germany
anne.remke@uni-muenster.de

## ABSTRACT

We present the tool HYPEG, i.e., a simulator for hybrid Petri nets with general transitions, which uses discrete-event simulation and Statistical Model Checking techniques. The implementation of HYPEG is discussed and several hypothesis tests as well as approaches for the computation of confidence intervals are discussed. Using a simple case study on a cooling tower of a generating plant, we compare the results and performances of the different methods.

## KEYWORDS

Tool, Hybrid Petri nets, discrete-event simulation, Statistical Model Checking, confidence intervals, hypothesis tests

**ACM Reference format:**
Carina Pilch, Fabian Edenfeld, and Anne Remke. 2017. HYPEG: Statistical Model Checking for hybrid Petri nets. In *Proceedings of 11th EAI International Conference on Performance Evaluation Methodologies and Tools, Venice, Italy, December 5–7, 2017 (VALUETOOLS 2017),* 6 pages.
https://doi.org/10.1145/3150928.3150956

## 1 INTRODUCTION

The model formalism of hybrid Petri nets with general transitions is well-suited for modeling critical infrastructures like power systems and gas networks, since it combines discrete, continuous and stochastic components. Statistical Model Checking for hybrid Petri nets with general transitions (HPnGs) has been presented in [15].

This tool paper presents its implementation within a Java tool, which allows the simulation of hybrid Petri nets with multiple stochastic firings and model checking complex properties on this model. The tool called HYPEG is available on Github[1]. We are using well-known Statistical Model Checking (SMC) techniques for an automated verification of properties. HYPEG is able to calculate confidence intervals (CIs) by the standard approach, Wald CIs, Clopper-Pearson CIs and Score CIs. Implemented hypothesis tests are the Sequential Probability Ratio Test, the Gauss SSP test, the Gauss-CI test, the Chow-Robbins test and the Azuma test. The core idea is to simulate the behavior of the model for a finite number of simulation runs and then apply one of these methods to check if properties are fulfilled or not. Simulation provides an estimation of the probability measures and allows evaluating the system behavior in different situations. However, only probabilistic guarantees are given and accuracy depends on the number of samples.

In this work, we discuss the implementation of HYPEG and extend the work from [15] with additional methods for the calculation of confidence intervals and for hypothesis tests. We present a small case study on an attack on a cooling tower of a generating plant, in which we compare the results and run time of these methods.

### 1.1 Related Work

HPnGs have been introduced in [9]. As their analysis is mostly restricted in the number of general transition firings, [15] introduced Statistical Model Checking for HPnGs.

A number of tools exist for hybrid stochastic systems. The **Modest Toolset**[2] [10] provides analytical model checking algorithms and Statistical Model Checking methods for Stochastic Hybrid Automata (SHAs). **COSMOS**[3] [3] is a statistical model checker for Discrete Event Stochastic Processes (DESPs), which are described in terms of Generalized Stochastic Petri Nets (GSPNs), and it implements the Chow-Robbins test. The tool **UPPAAL**[4] focuses on (networks of) timed automata (TAs) and its extension **UPPAAL SMC** [4] provides Statistical Model Checking for a stochastic extension of those TAs, using the Sequential Probability Ratio Test for qualitative statements and the Chow-Robbins test for quantitative purposes. **Möbius**[5] [7] supports modeling formalisms like Stochastic Petri Nets (SPNs), Stochastic Automata Networks (SANs), Markov Chains and Queuing Networks. It mainly uses discrete-event simulation. These tools mainly differ in the considered modeling formalisms and the evaluation approaches. HYPEG allows choosing between different confidence intervals and hypothesis tests for models with discrete and continuous behavior and a large number random variables, which represent stochastic firing times that follow arbitrary distributions.

Several approaches for calculating confidence intervals for binomial distributions are summarized in [1], with a focus on the Wald CI, the Clopper-Pearson CI and the Score CI. The authors determine the actual coverage probability for each interval type. They conclude that the Score CI gives narrow intervals with an actual coverage probability close to the confidence level. Several hypothesis tests are compared in [17], some use a fixed sample size while others are sequential tests. The tests are divided into three classes. The first contains tests with a relatively high risk of drawing an incorrect conclusion. Tests in the second class may

[1] https://github.com/jannikhuels/libhpng


*VALUETOOLS 2017, December 5–7, 2017, Venice, Italy*

ACM ISBN 978-1-4503-6346-4/17/12.
https://doi.org/10.1145/3150928.3150956

[2] http://www.modestchecker.net/
[3] http://www.lsv.fr/Software/cosmos/
[4] http://www.uppaal.org/
[5] https://www.mobius.illinois.edu/

draw no conclusion at all. The third class contains sequential tests that may require an extremely large number of samples.

### 1.2 Outline

We provide a short overview on HPnGs in Section 2. In Section 3, we demonstrate the main aspects of the implementation. The techniques available in HYPEG are briefly discussed in Section 4. Using a small case study we compare the results and performance of these techniques in Section 5. The paper concludes in Section 6.

## 2 THE MODELING FORMALISM

Hybrid Petri nets with general transitions [9] provide a modeling formalism for systems with discrete, continuous and stochastic components. Section 2.1 gives an overview on HPnGs. The so-called *Stochastic Time Logic* [8] for the specification of properties is discussed in Section 2.2.

### 2.1 The HPnG model

For this tool paper, we restrict the discussion of HPnGs to the relevant features for the case study. We do not aim at a formal definition, but refer to [15] for the adopted modeling formalism.

A hybrid Petri nets consists of places, which hold a discrete or continuous marking. This marking determines the state of the system. Transitions can transfer discrete or continuous units from the marking of the places they are connected to via arcs. Discrete tokens are transferred upon the firing of the connected transition, whereas continuous units are transferred continuously with a predefined rate between continuous places. Three kinds of discrete transitions exist: (i) immediate transitions fire instantly when enabled, (ii) deterministic transitions fire after a predefined time and (iii) general transitions fire according to a specified probability distributions. Both, discrete and continuous transitions can be connected to and controlled by test or inhibitor arcs that enable or disable transitions based on the current marking of the place they are connected to.

The behavior of an HPnG is defined by several functions that assign boundaries, rates, clocks, weights and priorities to the components, as explained in [15]. Furthermore, we adopt concession and enabling rules, the definition of the evolution of continuous variables, clocks and enabling times as well as the concept for fluid share and rate adaption from [9]. The graphical representation of the components of HPnGs is shown in Figure 1.

### 2.2 Stochastic Time Logic

For expressing state- and path-based properties on HPnGs, the *Stochastic Time Logic (STL)* has been introduced in [8]. In [15], we have extended it to obtain a branching time logic with one probabilistic operator, which embraces linear path-based properties. We define an STL formula as follows:

$$\varphi ::= P_{\bowtie\Theta}(\Psi),$$
$$\Psi ::= tt \mid AP \mid \neg\Psi \mid \Psi \wedge \Psi \mid \Psi \mathcal{U}^{[t_1,t_2]} \Psi,$$

with comparison operator $\bowtie \in \{<, >\leq, \geq\}$, probability bound $\Theta \in [0, 1]$ and time points $t_1, t_2 \in \mathbb{R}_0^+$ with $t_1 \leq t_2$.

For the formal satisfaction relation, we refer to [15]. In the given syntax, $tt$ means *always true* and $AP$ stands for atomic properties that relate to state properties like the fluid level (i.e. the marking)

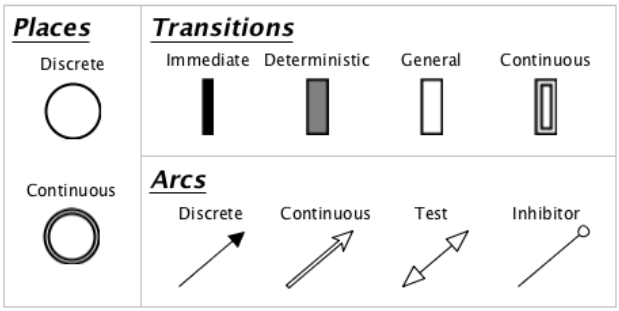


**Figure 1: Graphical representation of HPnGs components.**

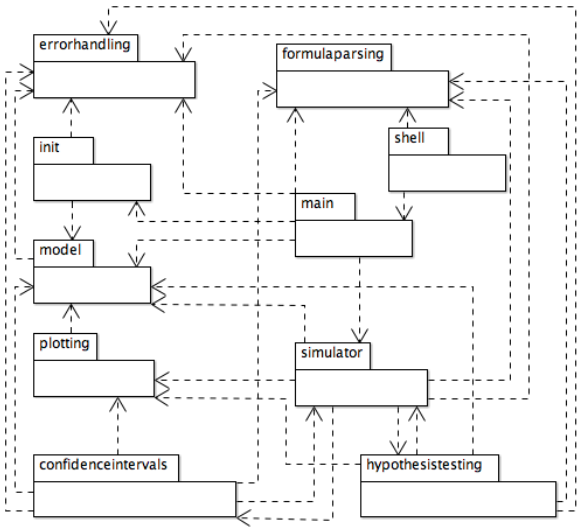


**Figure 2: UML diagram about package structure of HYPEG.**

of a continuous place, tokens (the marking) of a discrete place, clock values, drifts (derivatives of fluid levels), enabling times and enabling status, which are compared to specified thresholds. Furthermore, the inner formula $\Psi$ can be a combined property like negation, conjunction or an *until* property. whereas nested formulas are possible. An *until* property $\Psi_1 \mathcal{U}^{[t_1,t_2]} \Psi_2$ is fulfilled at time $t$ only if $\Psi_2$ holds at some time point $t' \in [t + t_1, t + t_2]$, and if in addition $\Psi_1$ holds consistently from $t$ up to this point $t'$.

## 3 IMPLEMENTATION

HYPEG is a Java library with an object-oriented code, which consists of a total number of 82 class files. For a better overview over these files, they are organized into several packages for the model implementation, the import of model files, the simulation itself, the calculation of confidence intervals, hypothesis testing, parsing of STL expressions, graphical plotting, error handling, a shell and for the main class. An UML diagram of these packages is shown in Figure 2, where an arrow points from package A to package B only if classes of A import classes of B. Since their functionalities are closely interrelated, there exist many interdependencies between the packages.

Within a case study presented in [15], we compared the results of HYPEG with other evaluation tools and demonstrated that HYPEG gives correct results. Incorrect input files and STL formulas are detected and communicated to the user. A detailed exception handling catches a large number of possible errors and prevents an abortion of the tool. Regarding usability, a specific background knowledge about HPnGs and STL is required for using HYPEG. The simple shell provides a manageable amount of commands while providing proper user guidance to keep the usage easy and intuitive. The run time of the different functions of HYPEG highly depends on the input files and chosen parameters.

We provide an overview on the input and output data of HYPEG in Section 3.1 and explain the formula parsing in Section 3.2. The simulation approach of HYPEG is discussed in Section 3.3. In Section 3.4, we present the front-end of the tool. Further instructions on the use of HYPEG are available in our Wiki[6].

## 3.1 Input and output data

The HPnG formalism, i.e. all kinds of places, transitions and arcs and the rules on their behavior, is implemented within a Java object structure, created by an XML file reader. Within an XML input file, each component of the corresponding model is specified. HYPEG uses *JAXB*[7] and a predefined mapping between XML elements and the Java objects to import an input file. For validation, the file is checked against a predefined XMLSchema. Furthermore, connections between the components are set and the objects are initialized with their initial states. Finally, the model includes a main object, which contains lists of the places, transitions and arcs. In addition, the simulation parameters are imported from a configuration file and the user can enter an STL formula, as explained in Section 3.2.

HYPEG outputs the model checking results, i.e. a confidence interval or the result of a hypothesis test, the number of simulation runs and the computation time. The output is written into a log-file and also printed on the console. Another functionality provided by the tool is plotting the evolution of the continuous places in a graph with the simulation time on the x-axis and the fluid level on the y-axis, in which the mean value and the confidence interval for each continuous place are depicted.

## 3.2 Formula parsing

The user can enter STL properties in a predefined textual format via the console. The input has to include the point in time $t$ and a probability operator asking either whether the probability that a specific STL formula $\Psi$ holds fulfills a certain threshold ($\bowtie \Theta$) or asking for a confidence interval for the probability that this formula holds. The former is expressed as $\mathbf{P}_{\bowtie}\,\boldsymbol{\Theta}(\boldsymbol{\Psi})$ (with $\bowtie \in \{<, >\leq, \geq\}$ and $\Theta \in [0, 1]$) and the latter is expressed, following the notation in PRISM [12], as **P=?(Ψ)**. The inner property $\Psi$ has to be one of the following:

- **tt** for true,
- $\mathbf{!\Psi_1}$ for negation,
- $\mathbf{AND(\Psi_1, \Psi_2)}$ for conjunction,
- $\mathbf{U[t_1, t_2](\Psi_1, \Psi_2)}$ for an *until* property with $t_1, t_2 \in \mathbb{R}_0^+$,
- an atomic property,

whereas $\Psi_1$ and $\Psi_2$ are inner properties again, so nested properties are possible. For example, the atomic property **fluidlevel('*id*')>1.0** requires that the fluid level of the continuous place *id* is greater than one. For parsing STL properties, the compiler and pre-processor library *JavaCC/JJTree*[8] has been integrated into HYPEG. It provides the possibility to transform the textual input into a tree structure. In case of an incorrect input, a notification is given to the user.

## 3.3 Simulation

A simulation experiment consists of several simulation runs, which simulate single paths of the given HPnG model. Before each run, the simulation time is set to zero and the HPnG model is set to its initial state. In HYPEG, we have implemented a simulation handler, which starts and controls individual simulation runs. The markings and events of each run are stored for each point in time, where an event occurs. When the run is finished, it is checked if the considered property $\Psi$ is fulfilled. The number of runs in total and the number of runs fulfilling $\Psi$ are counted and based on that, the simulation handler is able to calculate confidence intervals or to take a decision on hypotheses.

The main approach that we are using is discrete-event simulation: At the beginning of each simulation run, the current simulation time *currentTime* is initialized as zero. Afterwards, the following loop is executed:

```
while (currentTime <= maxTime)
  currentTime = getAndCompleteNextEvent(currentTime);
```

Within this instruction, the function *getAndCompleteNextEvent()* is repeatedly called for any event until the maximum time *maxTime* is reached. In contrast to time-based simulation, the step size of discrete-event simulation has optimal length, which makes this approach very efficient. The system model is updated whenever an event occurs. This way, dependent events are handled one-by-one [11]. Within the *getAndCompleteNextEvent()* function, the occurrence of the particular next event is determined. For identifying the next event, the algorithm iterates through all enabled transitions, continuous places and test or inhibitor arcs connected to continuous places. These components cause events, when (i) a transitions fires, (ii) a continuous places reaches its upper or lower boundary and (iii) the validity of the criteria of a connected test or inhibitor arc changes. When the time point of an event has been determined, the state of the model is updated to the time of the event by updating the continuous variables. Then, the event is executed and again the state is updated according to the effects caused by the event. Finally, the function returns the simulation time point, at which the event has occurred.

In an HPnG model, general transitions fire according to predefined distributions. For each firing of a general transition, a random variable represents the time between the enabling of the transition and the corresponding firing. The simulator handles these random variables by sampling firing times from the corresponding distribution using the inversion method, which is provided by the library *SSJ*[9]. When a new simulation experiment is started in HYPEG, the random stream is initialized by setting seeds randomly. The firing times are sampled at the beginning of every simulation run and

[6] https://github.com/jannikhuels/libhpng/wiki
[7] https://jaxb.java.net
[8] https://javacc.org/
[9] http://simul.iro.umontreal.ca/ssj/indexe.html

after every firing, respectively. For further details regarding the core of the simulation, we refer to [14, 15].

### 3.4 The front-end

So far, HYPEG is a shell application and provides user interactions via a console. The current shell has been implemented using the library *Spring Shell*[10]. Next to commands for changing several settings and simulation parameters, the main commands are:

- **read –p**, followed by a file path, reads an XML input file (see 3.1).
- **plot –t**, followed by the simulation time, creates the graphical plot for the continuous places.
- **parse** reads an STL formula and outputs the resulting parse tree (see 3.2).
- **check** imports an STL formula and calls the parser and simulation handler (see 3.2 and 3.3).

## 4 STATISTICAL MODEL CHECKING

When model checking STL properties, we distinguish between the $P_{=?}[\Psi, t]$ operator and the $P_{\bowtie\Theta}[\Psi, t]$ operator. When we consider a property within the $P_{=?}[\Psi, t]$ operator, the simulation data is used to estimate the exact probability that the property $\Psi$ is fulfilled at time point $t$. For properties inside the operator $P_{\bowtie\Theta}[\Psi, t]$, the tool aims to decide whether the property holds with a probability that matches the threshold $\bowtie \Theta$ .

The property $\Psi$ can express an atomic property (either discrete or continuous) or it can be nested using conjunction and / or negation. Discrete atomic properties can easily be model checked by taking the state variable at the time point of the last event before $t$, since their validity does not change in between events. Continuous variables evolve with a constant derivative in between events and though HYPEG updates the continuous state variable to point $t$ when checking the property. Combined properties with conjunction and negation are model checked by recursively evaluating the validity of the sub properties.

A so-called *until* property with the syntax $\Psi_1 \mathcal{U}^{[t_1,t_2]} \Psi_2$ holds at some time point $t$ if $\Psi_1$ holds consistently until $\Psi_2$ is fulfilled within the interval $[t + t_1, t + t_2]$. When model checking an *until* property, HYPEG first checks if $\Psi_1$ holds consistently within the interval $[t, t + t_1]$. If it does, it searches within $[t + t_1, t + t_2]$ for a point in time $t_{\Psi_2}$, at which $\Psi_2$ is fulfilled. Finally, $\Psi_1$ has to hold until this point $t_{\Psi_2}$. Only then, the *until* property is fulfilled.

Being able to model check a property $\Psi$ for a single simulation run, we use statistical techniques for evaluating the whole set of runs. Several methods for calculating a confidence interval for the probability that $\Psi$ holds are presented in Section 4.1. We give an overview about some hypothesis tests in Section 4.2.

### 4.1 Calculation of confidence intervals

In the following, we consider a random variable $X_i \sim \text{Bernoulli}(p)$ with its realizations $x_1, ..., x_n$ for $n$ simulation runs with $1 \leq i \leq n$, which evaluates to one if a property $\Psi$ holds for the i-th simulation run. Confidence intervals for the probability that $\Psi$ holds are computed, which represent the quality of the simulation. For all of these confidence intervals, a maximum width $2w$ with $w \in (0, 1]$ of the interval and a level of confidence $\lambda \in [0, 1]$ are required, so that in $(100 \cdot (1 - \lambda))\%$ of the times the real probability lies within the confidence interval.

In [15], we have presented the general approach for the calculation of a confidence interval (CI) with unknown standard deviation. In the following, we call this interval the *Standard Confidence Interval* [13]. Since we are considering independent Bernoulli random variables, we are able to use interval estimation approaches for binomial distributions, as presented in [1]. The Wald CI is based on the asymptotic normality of the sample proportion and it estimates the standard error. The *Clopper-Pearson Confidence Interval* [6] avoids approximation and guarantees a coverage probability of at least $1 - \lambda$, though it is also called the *exact* confidence interval in [1]. It is based on an equal-tailed binomial test, in which a null hypothesis $H_0 : p = \Theta$ is tested against an alternative hypothesis $H_1 : p \neq \Theta$ for a binomial distributed $p$. The Clopper-Pearson CI basically determines a set of all values $\Theta$, for which the probability that $H_0$ is not rejected is $\leq \lambda$.

Another approach is the *Score Confidence Interval* [19], which tends to have a high performance. Like the Clopper-Pearson CI, it is based on inverting a hypothesis test, the so-called score test. This test is based on determining the logarithmic likelihood for $\Theta$ from the null hypothesis $H_0 : p = \Theta$.

### 4.2 Hypothesis testing

Using a hypothesis test, HYPEG decides whether the probability that a property $\Psi$ holds at time point $t$ fulfills $\bowtie \Theta$, having $\bowtie \in \{<, \leq, >, \geq\}$ and $0 \leq \Theta \leq 1$. In general, we look at the null hypothesis $H_0 : p = \Theta$ and two alternative hypotheses $H_{+1} : p > \Theta$ and $H_{+1} : p < \Theta$. On the one hand, the probability for a *type-1-error* (*false positives*), which means that $H_{+1}$ resp. $H_{-1}$ is accepted even though it does not hold, is required to be less or equal to a value $\alpha$. On the other hand, we expect the probability for a *type-2-error* (*false negatives*), which is about not accepting $H_{+1}$ resp. $H_{-1}$ when it does hold, to be less or equal to a value $\beta$.

Recall from Section 1.1 that there are three classes of hypothesis tests. The first class of tests consists of tests with a high risk of drawing a wrong conclusion for small values of $|p - \Theta|$. In opposite to tests of the other classes, the tests of the first class consider the hypotheses $H'_{+1} : p > \Theta + \delta$ and $H'_{-1} : p < \Theta - \delta$. The parameter $\delta > 0$ is called the correctness-indifference level and it has to be chosen, such that we can safely exclude that the real value of $p$ lies within the interval $[\Theta-\delta, \Theta+\delta]$. Only if this holds, the error bounds $\alpha$ and $\beta$ are valid. The *Sequential Probability Ratio Test (SPRT)* [18] works sequentially, which means it does not have a fixed sample size. For $n$ simulation runs, a likelihood ratio $L$ is calculated and compared to $A = \frac{1-\alpha}{\alpha}$ and $B = \frac{\alpha}{1-\alpha}$. If $L < B$, we accept $H'_{+1}$ and $L > A$, $H_{-1}$ is accepted. Otherwise, another realization $x_{n+1} \in X_i$, i.e. another simulation run, is required. The *Gauss SSP* test [20] can be seen as a fixed sample size version of the SPRT. The number of simulation runs depends on $\delta$, as presented in [1]. Within the Gauss SSP, we determine a test statistic $Z_n = \sum_{i=1}^{n} X_i - n\Theta$ and take a decision upon whether it is greater or lower zero.

Hypothesis tests from the seconds class are based on confidence intervals and have a relatively high risk of terminating without

[10] https://projects.spring.io/spring-shell/

result for a small value of $|p - \Theta|$. The minimum number of runs that is required to be conclusive, depends on the so-called power indifference level $\zeta$, such that the error bounds are guaranteed for $|p - \Theta| > \zeta$. Within the *Gauss-CI* test [16], an approximation of a Wald CI with confidence level $\lambda = 1 - \alpha$, whose midpoint is shifted towards zero, is calculated. If the test statistic $Z_n$ lies within the interval, the test finishes without result. Another test of this class is the *Chow-Robbins* test [5], which is sequential. It also calculates a Wald CI and executes simulation runs until this interval has a desired width, which is dependent on $\zeta$.

The *Azuma* test [16] from Class III is sequential and always leads to a result. However, it has a high risk of taking a large run time. Its performance depends on the quality of the input parameter *guess*, denoted by $\gamma$, which represents the expectation of $|p - \Theta|$. This test is characterized by the functions $u(n)$ and $l(n)$, which depend on $\gamma$. After every run $n_i$, the test statistic $Z_n$ is determined and compared to the function values of $u(n_i)$ and $l(n_i)$. If no hypothesis can be accepted, the simulation is continued for another run.

## 5 CASE STUDY

To compare the different methods for the computation of confidence intervals and the different hypothesis tests, we carried out a small case study. The model, which abstracts the process of converting hot water to steam in a cooling tower of a generating plant, is presented in Section 5.1. We model check if the water level in the tower reaches its capacity limit within the first five minutes.

The different confidence intervals are compared in Section 5.2 and the comparison of the hypothesis tests is discussed in Section 5.3. The tests were executed using the environment *Eclipse Neon* (version 4.6.1) on a 64-Bit macOS Sierra system (version 10.12.5).

### 5.1 HPnG model for a cooling tower

The HPnG model of our case study is shown in Figure 3. The cooling tower is modeled by the continuous place $P_{\text{tower}}$ with a capacity of 50 tons of water. The continuous transition $T_{\text{in}}$ models the inflow of hot water from the generating plant into the tower with a constant fluid rate of half a ton per second. The conversion of water into steam is modeled as outflow governed by transition $T_{\text{out}}$ with a rate of 1.5 tons per second. The inflow is always enabled. The outflow starts when the tower holds a water level of at least 30 tons. At this point, the deterministic transition $T_{\text{filled}}$ fires and moves a token from place $P_{\text{inflow}}$ to place $P_{\text{outflow}}$. Then $T_{\text{out}}$ becomes enabled and fires after 3 seconds. When the fluid level of $P_{\text{tower}}$ is below 5 tons, $T_{\text{finished}}$ fires after 3 seconds, to ensure a minimum stock.

This HPnG also models the random duration of a cyber-attack on the system, which disables the outflow process. We assumed that the attacker aims to prevent the start of the steaming process *just-in-time*, when the water level is at its highest point after 30 seconds. To illustrate the capability of the tool, we model a simplified attack which introduces random variables. Following [2], our model includes the duration of a potential cyber-attack, as a general transition $T_{\text{attack}}$ following a folded normal distribution with a mean of 30 seconds and a standard deviation of 2 seconds. If it fires, a token is moved from $P_{\text{safe}}$ to $P_{\text{unsafe}}$, so that $T_{\text{filled}}$ becomes disabled. Hence, the outflow transition $T_{\text{out}}$ remains disabled, too. So, the cyber-attack is able to prevent the conversion of water into steam, which leads to a safety issue of having an inflow of hot water, but no outflow.

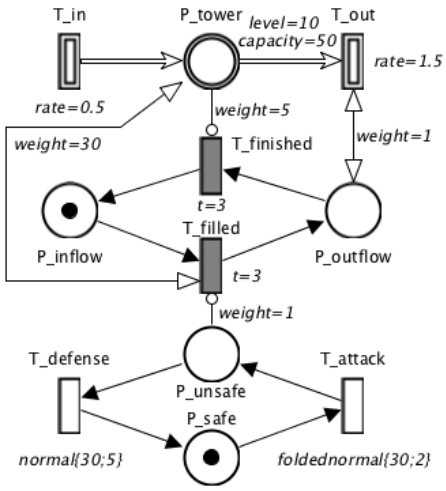


**Figure 3: HPnG model of a cooling tower.**

However, we assume that the system has some kind of intrusion detection and is able to recognize the attack and take counter-measures quickly. This mechanism is modeled by the general transition $T_{\text{defense}}$, which returns the critical token back to the place $P_{\text{safe}}$. We assumed that the defense mechanism of the system needs about 30 seconds to detect the attack and take counter-measures. However, the duration of the defense has a higher variance than the duration of the attack. Hence, $T_{\text{defense}}$ follows a normal distribution with a mean of 30 seconds and a standard deviation of 5 seconds.

### 5.2 Comparison of confidence intervals

We compared the different confidence intervals by checking if the water level in the tower reaches its upper bound of 50 tons within the first 5 minutes (300 seconds). This can be expressed by the following *until* property:

$$P_{=?}[(tt)\ \mathcal{U}^{[0,120]}\ (x_{P\_\text{tower}} >= 50), 0]. \tag{1}$$

The confidence intervals were required to have a width of $2w = 2 \cdot 0.01$ and a confidence level of $\lambda = 0.95$. The simulation was set to a minimum number of 100 and maximum of 100000 simulation runs. For each method, one hundred intervals were determined.

Table 1 shows the results of the case study for the different confidence intervals and their respective run times. The second to fourth column show the average value over all midpoints (i.e., the point in the center of an interval with the same distance to both bounds), and over the lower bounds and upper bounds. The results over all methods had an average midpoint around 0.080718. The fifth column of the table shows the percentage of intervals, in which this average midpoint lies, called *Coverage*. Furthermore, the table shows the mean, minimum and maximum number of simulation runs, the total computation time and the average time per run.

In our case study, the Score CI had the highest coverage and the average midpoint of the Score intervals lies closest to the value

**Table 1: Case study results for confidence interval methods**

| Method | Midpoint | Lower B. | Upper B. | Coverage | #Runs | Min #Runs | Max #Runs | Time | Time/Run |
|---|---|---|---|---|---|---|---|---|---|
| Standard | 0.080079 | 0.070081 | 0.090078 | 93% | 2833 | 2339 | 3795 | 53, 12 s | 18, 75 ms |
| Wald | 0.080300 | 0.069519 | 0.089316 | 94% | 2836 | 2395 | 3289 | 72, 01 s | 25, 39 ms |
| Clopper-Pearson | 0.081392 | 0.070630 | 0.090427 | 95% | 2958 | 2553 | 3377 | 79, 62 s | 26, 92 ms |
| Score | 0.081100 | 0.070488 | 0.090285 | 97% | 2850 | 2532 | 3236 | 64, 65 s | 22, 68 ms |

**Table 2: Case study results for hypothesis tests**

| Test | Class | Correct | Incorrect | Inconclusive | #Runs | min #Runs | max #Runs | Time | Time/Run |
|---|---|---|---|---|---|---|---|---|---|
| SPRT | I | 97% | 3% | – | 880 | 100 | 2927 | 12, 65 s | 14, 37 ms |
| Gauss-SSP | I | 95% | 5% | – | 1992 | *fixed size* | *fixed size* | 29, 33 s | 14, 72 ms |
| Gauss-CI | II | 98% | – | 2% | 7499 | *fixed size* | *fixed size* | 125, 51 s | 16, 74 ms |
| Chow-Robbins | II | 100% | – | – | 8124 | 7521 | 11631 | 171, 46 s | 21, 10 ms |
| Azuma | III | 100% | – | – | 34576 | 13810 | 61018 | 615, 39 s | 17, 80 ms |

of 0.080718. The required confidence level was 0.95, which relates to a coverage of 95%. Our experiments show that the coverage of the Standard CI and the Wald CI lie slightly below this parameter. The mean, minimum and maximum number of simulation runs are similar when comparing all methods. The method of the Standard CI proved to be the quickest in total and per single run.

### 5.3 Comparison of hypothesis tests

For comparing the hypothesis tests, we considered the same *until* property as before, but checked if the associated probability is greater than $\Theta = 0.07$. We set both error bounds $\alpha$ and $\beta$ to 0.05. As an estimate for $|p - \Theta| \approx |0.08 - 0.07|$, we set the correctness indifference level $\delta$, the power indifference level $\zeta$ and the *guess* $\gamma$ to 0.01, so that all tests are comparable. With the same bounds on the number of simulation runs as before, one hundred hypothesis tests of each kind were executed.

The results are shown in Table 2, which presents the percentages of correct, incorrect and inconclusive tests. The Chow-Robbins test and the Azuma test both gave 100% correct test results, whereas the tests of the first class gave a few incorrect results and the Gauss-CI test had two inconclusive results. The Class I tests, especially the SPRT, need much less time than the other tests, whereas the Azuma test takes on average nearly 40 times as many runs as the SPRT.

## 6 CONCLUSION

We presented our tool HYPEG and discussed its implementation with a focus on additional statistical techniques for the calculation of confidence intervals and hypothesis tests, which we compared using a simple case study. The results showed that the tests have similar strengths and weaknesses as mentioned in [1] and [17]. Obviously, the choice of techniques mainly depends on the requirements on the accuracy of the results. Concluding, we developed a tool that comprises all the discussed techniques and enables Statistical Model Checking for hybrid Petri nets with general transitions. In future works, we will investigate how we can reduce the dependency of the run time on the number of simulation runs by using multi-threading. Another goal is to improve the attractiveness of HYPEG with a web-based front-end, while retaining its simplicity.

## REFERENCES


[1] A. Agresti and B. Coull. 1998. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician* 52 (1998), 119–126.

[2] F. Arnold, H. Hermanns, R. Pulungan, and M. Stoelinga. 2014. Time-Dependent Analysis of Attacks. In *3rd Int. Conf. on Principles of Security and Trust*. Springer, 285–305.

[3] P. Ballarini, H. Djafri, M. Duflot, S. Haddad, and N. Pekergin. 2011. COSMOS: A Statistical Model Checker for the Hybrid Automata Stochastic Logic. In *8th. Int Conf. on Quantitative Evaluation of Systems*. IEEE, 143–144.

[4] P. Bulychev, A. David, K. Gulstrand Larsen, M. Mikučionis, D. Bøgsted Poulsen, A. Legay, and Z. Wang. 2012. UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata. *ArXiv e-prints* (2012). arXiv:cs.LO/1207.1272

[5] Y. Chow and H. Robbins. 1965. On the Asymptotic Theory of Fixed-Width Sequential Confidence Intervals for the Mean. *The Annals of Mathematical Statistics* 36, 2 (04 1965), 457–462.

[6] C. Clopper and E. Pearson. 1934. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika* (1934), 404–413.

[7] D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. Doyle, W. Sanders, and P. Webster. 2002. The Mobius framework and its implementation. *Transactions on Software Engineering* 28, 10 (2002), 956–969.

[8] H. Ghasemieh, A. Remke, and B. Haverkort. 2013. Survivability Evaluation of Fluid Critical Infrastructures Using Hybrid Petri Nets. In *19th Pacific Rim Int. Symp. on Dependable Computing*. IEEE, 152–161.

[9] M. Gribaudo and A. Remke. 2016. Hybrid Petri nets with general one-shot transitions. *Performance Evaluation* 105 (2016), 22–50.

[10] A. Hartmanns and H. Hermanns. 2014. The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification. In *20th Int. Conf. on Tools and Algorithms for Construction and Analysis of Systems*. Springer, 593–598.

[11] Boudewijn Haverkort. 1998. *Performance of Computer Communication Systems: A Model-Based Approach*. John Wiley & Sons.

[12] M. Kwiatkowska, G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *23rd Int. Conf. on Computer Aided Verification*. Springer, 585–591.

[13] J. Neyman. 1937. Outline of a theory of statistical estimation based on the classical theory of probability. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences* 236, 767 (1937), 333–380.

[14] C. Pilch. 2016. *Development of an event-based simulator for model checking hybrid Petri nets with random variables*. Master's thesis. University of Münster, Germany. http://go.wwu.de/1yiqr

[15] C. Pilch and A. Remke. 2017. Statistical Model Checking for hybrid Petri nets with multiple general transitions. In *47th IEEE/IFIP Int. Conf. on Dependable Systems and Networks*. IEEE, 475–486.

[16] D. Reijsbergen. 2013. *Efficient Simulation Techniques for Stochastic Model Checking*. Ph.D. Dissertation. Univ. of Twente, Netherlands.

[17] D. Reijsbergen, P.-T. de Boer, W. Scheinhardt, and B. Haverkort. 2015. On hypothesis testing for statistical model checking. *Int. Journal on Software Tools for Technology Transfer* 17, 4 (2015), 377–395.

[18] A. Wald. 1945. Sequential Tests of Statistical Hypotheses. *The Annals of Mathematical Statistics* 16, 2 (06 1945), 117–186.

[19] E. Wilson. 1927. Probable inference, the law of succession, and statistical inference. *J. Amer. Statist. Assoc.* 22, 158 (1927), 209–212.

[20] H. Younes and R. Simmons. 2006. Statistical probabilistic model checking with a focus on time-bounded properties. *Information and Computation* 204, 9 (2006), 1368 – 1409.