Assessment of Maintenance Policies for Smart Buildings: Application of Formal Methods to Fault Maintenance Trees

Alessandro Abate[1], Carlos E. Budde[2], Nathalie Cauchi[3], Khaza Anuarul Hoque[4], and Mariëlle Stoelinga[5]

[1,3] *Department of Computer Science, University of Oxford, Oxford, OX1 2JD, United Kingdom*
alessandro.abate@cs.ox.ac.uk
nathalie.cauchi@cs.ox.ac.uk

[2,5] *Formal Methods and Tools group, University of Twente, Enschede, 7522 NB, The Netherlands*
c.e.budde@utwente.nl
m.i.a.stoelinga@utwente.nl

[4] *Department of Electrical Engineering and Computer Science, University of Missouri, Columbia, 65211, USA*
hoquek@missouri.edu

ABSTRACT

Cyber-physical systems must meet high RAMS—reliability, availability, maintainability, and safety—standards. It is of essence to implement robust maintenance policies that decrease system downtime in a cost-effective way. Power plants and smart buildings are prominent examples where the cost of periodic inspections is high, and should be mitigated without compromising system reliability and availability. Fault Maintenance Trees (FMTs), a novel extension in fault tree analysis, can be used to assess system resilience: FMTs allow reasoning about failures in the presence of maintenance strategies, by encoding fault modes in a comprehensible and "maintenance-friendly" manner. A main concern is how to build a concrete model from the FMT, in order to compute the relevant RAMS metrics via (ideally automatic) analyses. Formal methods offer automated and trustworthy techniques to tackle with such task. In this work, we apply quantitative model checking—a well established formal verification technique—to analyse the FMT of a Heating, Ventilation and Air-Conditioning unit from a smart building. More specifically, we model the FMT in terms of continuous-time Markov chains and priced time automata, which we respectively analyse using probabilistic and statistical model checking. In this way we are capable of automatically estimating the reliability, availability, expected number of failures, and differentiated costs of the FMT model for various time horizons and maintenance policies. We further contrast the two approaches we use, and identify their advantages and drawbacks.

Alessandro Abate et al. This is an open-access article distributed under the terms of the Creative Commons Attribution 3.0 United States License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

ACRONYMS

AHU	Air Handling Unit
BE	Basic Event
CSL	Continuous Stochastic Logic
CTMC	Continuous Time Markov Chain
DFT	Dynamic Fault Tree
EBE	Extended Basic Event
ENF	Expected Number of Failures
FMT	Fault Maintenance Tree
HVAC	Heating, Ventilation and Air-Conditioning
IE	Intermediate Event
IM	Inspection Module
KPI	Key Performance Indicator
MTTF	Mean Time To Failure
PMC	Probabilistic Model Checking
PTA	Priced Time Automata
RAMS	Reliability, Availability, Maintainability, Safety
RM	Repair Module
SMC	Statistical Model Checking
TA	Time Automata
TLE	Top Level Event

1. INTRODUCTION

The high standards imposed upon many cyber-physical systems are usually met with inspection-intensive, heavily resilient maintenance policies which are carefully devised to diminish the chances of downtime. In spite of the best endeavours, such policies are prone to become costly, and the

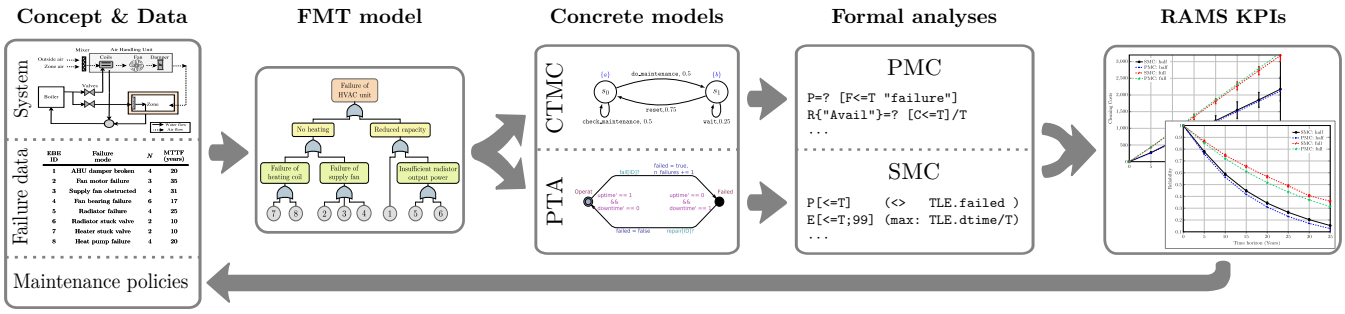

Figure 1. Overall work flow, from HVAC to computation of RAMS metrics

compromise between maintenance budget and the probability of system failure is, more often than not, a multi-variate problem of high complexity. Modern edifices are a prominent example, where the cost of periodic inspections is high, and should be mitigated at the same time that the fault resilience of the system is kept at the desired level.

All of this is inserted in a modern world with a steadily increasing interest in making things "smart." In the building sector in particular, engineers and researchers have been focused for years in developing (ever more) *smart buildings*, which feature a supervised control system for lighting, ventilation, surveillance, heating, etc. This enables a more efficient use of resources aimed at guaranteeing all basic functionality. To better steer decision making at the individual level, myriads of present-day works focus on detecting occupancy in rooms (Zhao et al., 2016; Dong, Lam, & Neuman, 2011), synthesising optimal control policies (Lindelöf et al., 2015; Haesaert, Cauchi, & Abate, 2017), and modelling the internal zone temperature and air quality (R. Volk, Stengel, & Schultmann, 2014; Han, Gao, & Fan, 2012). Among other features, smart buildings are characterised by a swarm of monitoring sensors: their constant reading of the internal environmental conditions is intended to deliver all necessary services to ensure comfort and productivity of the inhabitants, e.g. proper thermal stability and good air quality. These services must be ubiquitous, compliant with RAMS—reliability, availability, maintainability, and safety—requirements, and cost effective. The correct and dependable operation of the premises is subject to the availability of such services which, in turn, require all components in the building to work at sufficient capacity.

A crucial component in smart buildings is the Heat, Ventilation, and Air-Conditioning unit (HVAC) that regulates temperature and air circulation. The lifespan and reliability of the HVAC can be improved by coupling early fault detection with maintenance actions. Maintenance can be optimised for each specific setting studied: many methodologies to approach such optimisation can be found in the literature, e.g. condition-based maintenance modelling (Alaswad & Xiang, 2017), predictive maintenance (Macek, Endel, Cauchi, & Abate, 2017), and so on; the reader is referred to Nicolai and Dekker (2008) for a review. A novel technique involves decomposing the various fault modes of the system in a so called *Fault Maintenance Tree* (FMT, Ruijters, Guck, Drolenga, & Stoelinga, 2016; Ruijters, Guck, Drolenga, Peters, & Stoelinga, 2016). FMTs extend Fault Trees (which are commonly deployed in RAMS analyses) by introducing maintenance driven concepts like inspections and partial degradation of components. FMT analyses enable an in-depth study of the relevant RAMS metrics, e.g. system reliability and total costs incurred, which then serve as platform for planning concrete improvements on the implemented maintenance policy.

However, an FMT is a high-level concept. This means that, although descriptive and convenient for reasoning about system health decay, an FMT needs to be instantiated in order to allow a concrete computation of metrics[1]. For that purpose, formal methods in computer science offer a variety of modelling formalisms that can capture the behaviours underlying an FMT description. Among the main appeals of these methods we highlight (i) the formal guarantees of correctness for the results computed, and (ii) the automation of the computing procedures, once the model and property queries have been specified. For these reasons, together with their adequacy to the structured nature of FMTs, formal methods are a promising choice when looking for mechanisms to derive RAMS metrics of a system.

In this work we use two well known modelling formalisms, namely *continuous-time Markov chains* (Aziz, Sanwal, Singhal, & Brayton, 2000) and *priced time automata* (Behrmann, Larsen, & Rasmussen, 2004), to instantiate our FMT of an HVAC unit. This means that Markov chains and time automata are implemented in computer software tools, yielding concrete (independent) representations of the higher-level FMT. The "concrete models" resulting from such implementations can be analysed via formal methods (e.g. quantitative model checking), to estimate values for all relevant RAMS metrics. We employ two alternative approaches to do this: for the continuous-time Markov chain model we use *probabilistic model checking*, and for the priced time automata model we use *statistical model checking*. In particular, we

[1] Technically speaking the FMT is *given semantics* as a (formal) model.

measure the reliability, availability, expected number of failures, and differentiated costs, which characterise the FMT HVAC model. Fig. 1 presents a schematic overview of the whole analysis process.

In this manner we aim to address two main challenges:

- motivate the use of formal methods in computer science for RAMS assessment of systems in smart buildings, and
- contrast the capabilities of the two model checking techniques chosen for the task.

To illustrate the potential and adequacy of our approach, we compute the aforementioned metrics for various time horizons and using different maintenance policies. Furthermore, we discuss the benefits and drawbacks of the two specific analysis techniques chosen. In this way we substantiate our claim that formal methods, in combination with FMT modelling, offer a rich framework to reason about the RAMS properties of a system, thus facilitating the selection of the maintenance policies that suit best a specific purpose.

This article is structured as follows: Sec. 2 presents the HVAC conceptualization chosen as central case study; Sec. 3 briefly introduces the fundamental theoretical concepts upon which our analyses are constructed; Sec. 4 describes in detail our application of formal methods to the case study; Sec. 5 presents and discusses the results yielded by the analyses described in Sec. 4; and Sec. 6 concludes this work, also mentioning some intended lines for future research.

2. HVAC CASE STUDY

The specific HVAC set-up considered in this work is depicted in Fig. 2, and corresponds to the system found in the "smart buildings" laboratory at the Department of Computer Science, University of Oxford, first examined by Cauchi, Hoque, Abate, and Stoelinga (2017). It also corresponds to HVAC units found commonly in medium-sized buildings (Kim & Katipamula, 2017). The set-up is composed of two circuits: one for water circulation and the other for air circulation. In the water circuit, the gas boiler warms up the supply water which is then transferred to the air handling unit (AHU), the heating coils, and the *zone* (or room) radiators. The rate of water flow is controlled via valves in the heating coil and in the radiators. In the air circuit, outside air is pumped into the AHU and mixed with zone air, until a homogeneous temperature is achieved in the mixer. The mixed air is then warmed up to the required temperature using the heating coil. A fan pushes this air into the zone at a rate controlled by the AHU dampers. On the other hand, the radiator is directly connected to the water circuitry and transfers heat from the supply water to the zone. The return water from the heating coil and all radiators is then gathered by the collector and pumped back to the boiler, where the heating cycle is restarted.

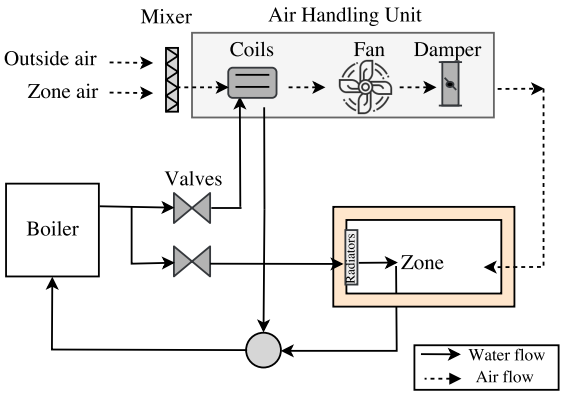

Figure 2. Schematic depiction of the HVAC set-up

The correct and continuous operation of the HVAC depends on the proper operation of its subsystems. Usually, all individual components from these subsystems have different expected lifetimes, as well as different sources of degradation and failures. For instance, the valves can get stuck due to the gradual increase in deposits from the water flow. This limits the rate of water flow in the heating system, consequently reducing the heat exchange rate. The supply fan gradually decreases its efficiency as the lifetime of the fan motor increases, while it may cease to work if it breaks due to a failure of the fan bearings or the motor itself. The AHU damper may break due to build up of fouling (Trojanova, Vass, Macek, Rojíček, & Stluka, 2009). The radiator may fail due to rust and leak inhibitors, which form solids that collect in the radiator cooling system and restrict the water flow. The boiler heating capacity reduces with age, due to corrosion and the accumulation of dust and dirt in the internal heat exchanger and flue (Durkin, 2006).

The aforementioned issues are a subset of possible failure causes that may lead to downtime of the HVAC unit from Fig. 2. When maintenance actions are being planned, all such (foreseeable) events must be considered to ensure compliance with the RAMS requirements. However and in spite of the best efforts, the operation of the individual components—and thus the whole system—degrades as time passes. This leads to a reduction in system performance and an increase in the operational and maintenance costs. To mitigate these expenses, service and maintenance strategies should be carefully designed and assessed. The main question is how to increase the performance of the whole system, while minimising the total number of expected failures and operational costs. In this work, we address such challenge by presenting techniques to assess the performance of an HVAC under different maintenance policies. These analysis techniques are automatic and enable a comparison of policies from different angles. Thus, following the work flow described in Fig. 1, any maintenance scheme can be updated and refined until the desired RAMS criteria is met.

3. THEORETICAL BACKGROUND

In order to tune a maintenance policy to the specific failure modes of an HVAC unit, and achieve the desired level of system resilience at reasonable costs, methodical and thorough analyses must be carried out. The modular structure of HVACs permits reasoning about operation and performance via fault maintenance trees. In particular this allows us to estimate and contrast a selection of RAMS metrics, based on which we can study the trade-off between system performance and money investment in maintenance actions.

In this section we give a brief introduction to FMTs, providing the necessary background to comprehend our model of the (fault modes of the) HVAC unit set-up presented in Sec. 2. We also formalise all metrics studied, and summarise the model checking concepts and techniques employed to estimate the metrics from our FMT.

3.1. Fault maintenance trees

Fault trees (Vesely, Goldberg, Roberts, & Haasl, 1981) are directed acyclic graphs consisting of two types of nodes: events and gates. An event is an occurrence within the system, e.g. the failure of a subsystem down to an individual component. Events can be divided into *basic events* (BEs) and *intermediate events* (IEs). BEs correspond to the leaves in fault trees and denote atomic failures in the system, typically modelled via random variables following the exponential distribution. IEs correspond to internal events which are caused by one or more other events. The event at the top of the tree, called the *top event* (TLE), represents the main event being analysed, i.e. a failure of the whole system under consideration.

The internal nodes of the graph are called *gates* and describe how failures in basic events and lower level gates interact, as they propagate towards the TLE. Each gate has one output and one or more inputs. Gates in (standard) fault trees are *static*, in the sense that their output at any point in time depends solely on the configuration of their inputs in that moment. Different gates model different logical interactions. These include (i) OR gates, which require only one child to fail in order to propagate a failure to the next level, (ii) AND gates, which require all children to fail, and (iii) VOT$_k$ gates, which fail if $\geqslant k$ children fail (Vesely et al., 2002). In standard fault trees a closed-form solution exists for many RAMS metrics, provided the distribution parameters of the basic events is known. Fig. 3 shows a fault tree instance.

Dynamic fault trees (DFTs) are a proper superset of fault trees, extended with gates that exhibit time- or order-dependent behaviour. This adds a level of complexity that rules out analytical approaches for DFTs (M. Volk, Junges, & Katoen, 2018), that is, no closed-form solution exists to compute most RAMS metrics from general DFTs. Fault maintenance trees are a superset of DFTs enriched with maintenance concepts (Ruijters, Guck, Drolenga, et al., 2016). This is achieved by the introduction of:

1. *Extended Basic Events* (EBEs), which are basic events whose failures are Erlang rather than exponentially distributed. The subsequent stepwise degradation as a function of time allows e.g. identifying lightly degraded components, whose health can be restored before an actual failure (that may trigger a TLE) occurs;
2. *Maintenance modules*, which handle timely inspections and repairs. The repair module (RM) performs cleaning, repair, or replacement actions on EBEs. These actions can be carried out either using fixed time schedules or when enabled by the inspection module (IM). The RM can also perform periodic maintenance actions independently of the IM. The IM performs periodic inspections and, when components degradation rises above a predefined threshold, a maintenance action is triggered by the IM and carried out by the RM, outside of the periodic maintenance cycle of the RM.

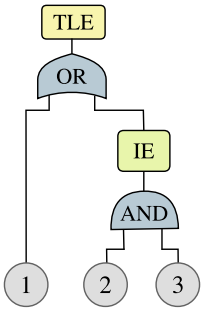

Figure 3. Fault tree with three basic events: BEs 2 and 3 are connected to an AND gate which triggers an IE. The IE is connected to an OR gate together with BE 1 and propagates failures to the TLE.

3.2. Metrics

Several RAMS requirements can be formalised as *Key Performance Indicators* (KPIs), i.e. a set of metrics which are specified over a time horizon $T > 0$, and quantify failures in the time window $[0, T]$. Some of the most relevant KPIs are:

- *Reliability* - the probability of the system not failing (i.e. observing a TLE) in the time window $[0, T]$;
- *Availability* - the expected fraction of time in the window $[0, T]$ that the system is operational;
- *Expected number of failures* (ENF) - the expected number of times a TLE is observed in the time window $[0, T]$;
- *Expected costs* - the total expected costs incurred in the time window $[0, T]$, including operational costs, inspection costs, costs of maintenance actions triggered, and costs associated to system failures.

We include the expected costs because maintenance is essentially a cost-driven concept. The costs accrued during system (in)operation, and also those incurred in the various service

actions and inspections, provide further insight on how well the HVAC is performing.

3.3. Model checking

FMT offers an abstract description of the fault modes of a system. This enables reasoning about the failure mechanisms but, to compute the metrics listed in Sec. 3.2 and to assess a particular maintenance policy, it is necessary to provide semantics to (viz. instantiate) the FMT. For this purpose, one can choose from a wide variety of modelling formalisms developed in the active area of formal methods in computer science. In particular we use continuous-time Markov chains (CTMC) and priced time automata (PTA). These are widely used formalisms that have rich tool support, and whose expressiveness meets our modelling requirements.

For both the CTMC and the PTA semantics of the FMT model, the KPIs of interest can be quantified via *quantitative model checking* (Clarke, Emerson, & Sistla, 1986). This is a well-established formal verification technique used to verify the correctness of finite-state automata. In general, model checking algorithms take two inputs: (i) a *formal model* of the system to be analysed, e.g. a description of the HVAC FMT as a CTMC, and (ii) a *property query* to be verified, usually following the syntax of some temporal logic. To check whether the query is valid in the model, these algorithms explore exhaustively and automatically all possible system configurations (*states*). In quantitative model checking applied to stochastic models, the likelihood of satisfiability of a property can be quantified, e.g. what is the probability of observing a system failure in a given time window. Furthermore, states and transitions can be augmented with *rewards*—real numbers associated with certain states or transitions of the model. This allows the computation of the expected value achieved by taking certain transitions or visiting particular states. For instance, one can encode the act of performing an inspection by associating a reward C_I to the corresponding model transition, where C_I is the cost of performing an inspection in the HVAC. Then the expected value computed for such transition in the time window $[0, T]$, is the expected cost incurred by HVAC inspections in that time span of system operation.

Quantitative model checking is a broad field that comes in different flavours (Clarke, Grumberg, & Peled, 1999). *Probabilistic model checking* (PMC, Kwiatkowska, Norman, & Parker, 2018) and Monte Carlo simulation techniques like *Statistical model checking* (SMC, Legay, Delahaye, & Bensalem, 2010) stand among its most prominent examples. In this work we perform analyses using both techniques, and present a comparison between the two.

3.3.1. Probabilistic model checking

PMC deals with systems that exhibit stochastic behaviour and is based on the construction and analysis of a probabilistic model of the system. It refers to a state space analysis technique for probabilistic finite state automata. The system is usually specified as a state transition model with probability as rates of transitions and labels on both the transitions and the states. A *probabilistic model checker* can calculate the probabilities of reaching different states in a model or compute the expected reward given a time horizon. For our state transition model we make use of CTMCs with labels in both transition and states.

Definition 3.1. The tuple $C = (S, s_{init}, \text{TL}, \text{AP}, L, \mathbf{R})$ defines a CTMC which is composed of:

- a set of states S,
- the initial state $s_{init} \in S$,
- a finite set of transition labels TL,
- a finite set of atomic propositions AP,
- a labelling function $L : S \to 2^{AP}$, and
- the transition rate matrix $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$.

The rate $\mathbf{R}(s, s')$ defines the delay before which a transition between states s and s' takes place. If $\mathbf{R}(s, s') \neq 0$ then the probability that a transition between the states s and s' is defined as $1 - e^{-\mathbf{R}(s,s')t}$, where t is time. No transitions can trigger if $\mathbf{R}(s, s') = 0$.

Fig. 4 presents a CTMC described using the tuple C where $S = \{s_0, s_1\}$, $s_{init} = s_0$, TL = {`do_maintenance`, `check_maintenance`, `reset`, `wait`}, AP $= \{a, b\}$, $L(s_0) = a$, $L(s_1) = b$ and

$$\mathbf{R} = \begin{bmatrix} 0.5 & 0.5 \\ 0.25 & 0.75 \end{bmatrix}.$$

Note, a system can be modelled using multiple CTMCs representing different subcomponents. Transition labels are then used to synchronise the individual CTMCs representing different parts of a system and in turn obtain the full CTMC representing the whole system.

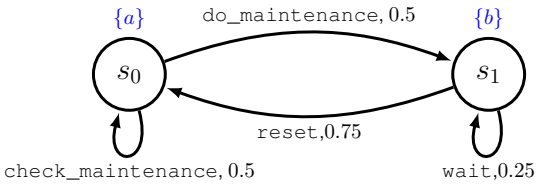

Figure 4. Example of a CTMC

To compute KPI metrics using PMC, we express properties in Continuous Stochastic Logic (CSL), which includes reward formulae. See Kwiatkowska, Norman, and Parker (2007) for the syntax and semantics of CSL. States (or particular transitions) can be associated with a real-valued reward, which allows us to compute reward based queries such as the expected cost of an action in a time window. Examples of CSL properties with its natural language translation are:

- $\mathbb{P}_{\geq 0.9}[\mathrm{F}\ b]$ - "The probability of the system eventually reaching the state labelled with 'b' is at least 0.9."
- $\mathrm{R}_{=?}[\mathrm{F}\ a]$ - "What is the expected reward accumulated until the system reaches the state labelled with 'a'?"

3.3.2. Statistical model checking

SMC solves basically the same questions that PMC does, but from the perspective of Monte Carlo simulation and statistical analysis. Rather than an exhaustive exploration of the state space, SMC samples finitely many (independent and random) runs of model behaviour, typically in the shape of execution traces. These traces are sequences of states and transitions that follow the semantics of the model, i.e. simulations over a specified finite time horizon. For any specified query, e.g. "can the model fail in the time window $[0, T]$?," or "how much do inspections cost?," each trace yields a concrete qualitative or quantitative answer, e.g. "on *this trace* the model *did not fail* within the time window $[0, T]$," or "over *this trace this much money* was spent in inspections." The more (independent and random) traces are generated, the more information of the stochastic behaviour of the model is gathered, and thus the more precisely and confidently can the query be answered in general.

The outcomes of these simulations can be regarded as a random sample of model behaviour. SMC performs statistical analyses on such random sample, in order to estimate an answer to the property query: the result will be e.g. a confidence interval, which contains the true value of the property with a certain level of confidence typically predefined by the user. Following traditional statistical concepts, the probability of SMC converging to an incorrect answer can be arbitrarily bounded (Younes & Simmons, 2002). This contrasts to PMC, where result correctness is formally guaranteed. Another difference with PMC is that SMC may take a longer (computation) time to build a sufficiently large random sample, viz. sample enough execution traces, in order to build a confidence interval with the desired confidence and precision. On the other hand, memory consumption by SMC is usually constant and thus negligible. This allows its use in many large models where, due to the hardware computing capabilities, PMC cannot deal with the amount of states involved.

With SMC we estimate KPI metrics from the FMT modelled as a priced timed automaton. PTA are an extension of *timed automata* (TA, Bengtsson & Yi, 2003) enriched with costs on *locations* (aka states) and *actions*. TA are transition systems using real-valued clocks and time invariants to specify deadlines. Invariants decorate locations, and actions (viz. enabling guards and variables updates) decorate edges (viz. transitions between locations). In PTA, fixed costs can be incurred when taking a transition, or proportionally to the time spent in a certain location. Furthermore, in modular frameworks, Input/Output broadcast synchronisation can take place between the various modules composing a system model.

Definition 3.2. A PTA is a tuple $A = (L, l_{init}, X, \Sigma, E, \mathbf{R}, I)$ where:

- L is a finite set of locations,
- $l_{init} \in L$ the initial location,
- X is a finite set of clocks,
- Σ is a finite set of actions,
- $E \subseteq L \times \mathcal{L} \times \Sigma \times 2^X$ is a finite set of edges, with $\mathcal{L}$ representing the set of clock constraints,
- $\mathbf{R} : L \to \mathbb{N}^X$ assigns a rate vector to each location, and
- I assigns an invariant to each location.

Fig. 5 shows an example of a PTA. It begins in the left location, and waits until either the time of clock x reaches T, or a "threshold" broadcast signal (thresh[id]) is received from another PTA module. If T time units elapse before a signal is received, the total cost C_total is incremented by C units, and the clock is reset (x:=0). If a threshold signal is received, the module takes the transition to the right location, where it waits until x == T. When that happens the PTA moves back to the left location, and in doing so it outputs a broadcast signal (force[a_id], to which other PTA modules may react), resets clock x , and increments the total cost by C.

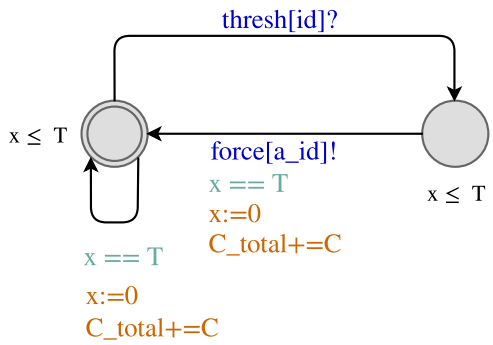

Figure 5. Example of a PTA

To compute the KPI metrics using SMC, a variant of (time bounded) CSL is used, which includes information to determine convergence of the estimation (Bengtsson & Yi, 2003; David, Larsen, Legay, Mikučionis, & Poulsen, 2015). To build a confidence interval for some chosen confidence level, statistical convergence can be determined in two ways: either the number of simulations is predefined, in which case the width of the interval is a random variable; or the width of the interval is predefined, in which case the number of simulations is a random variable. In our studies we use both criteria, the first to compute system reliability, and the second to compute all other metrics. Two examples of property queries (for some arbitrarily fixed confidence level) are:

- $\mathbb{P}^{\leqslant T}_{\leqslant 0.01}[b]$ - "For the time window $[0, T]$, compute a confidence interval of width $\leqslant 0.01$, for the probability of reaching the state labelled b,"
- $\mathrm{R}^{\leqslant T}_{9999}[a]$ - "For the time window $[0, T]$, compute a confidence interval using 9999 simulations, for the expected reward accumulated in states labelled a."

In the first case, increasingly many simulations will be run, until the statistical evidence is enough to build a confidence interval of width $\leqslant 0.01$. In the second case, 9999 simulations will be run, and the width of the resulting confidence interval cannot be known a priori. Notice that in the first case, the time for SMC to converge is unknown, whereas in the second case the run time can be estimated from the the time it takes to execute a single simulation run in that time window.

As a final note for this section, we highlight that it would suffice to choose a single modelling formalism and analysis technique, e.g. CTMC and PMC, and derive all KPI metrics for the FMT of the HVAC from that choice. However, one of the motivations of this work is to push the capabilities of formal methods when applied to FMT analysis. In particular we are keen on demonstrating the soundness of this approach. Consequently, we show that the outcomes produced coincide—to the expected degree—regardless of the particular formalism and analysis technique selected.

4. FORMAL ANALYSIS OF THE CASE STUDY

In Sec. 2, several causes leading to a general HVAC failure are listed. These affect at least one subsystem component from the set-up depicted in Fig. 2, namely a heating coil, a supply fan, or a radiator. In Fig. 6 we present an FMT decomposition of the corresponding fault modes, derived from analyses presented in Cauchi et al. (2017) for this case study.

In the tree, due to the use of OR gates, a failure of any leaf at the down most level results in a TLE. Such leaves are EBEs whose health decays in a stepwise manner following an Erlang distribution. In essence this means that each EBE starts in an initial phase which represents a component in perfect operational condition, aka "as good as new." As time passes the component will eventually *degrade* by internally moving to the next phase, which represents a decay in the health of the component. This process repeats until the component ultimately reaches its final phase; when this happens the EBE *fails* and sends a signal that propagates through the FMT.

The time to move from one degradation phase to the next is described by an exponentially distributed random variable. For each EBE, all random variables describing the jumps from the initial to the failed phase are independent and have the same rate parameter. The number of phases and mean time to failure of EBEs is typically derived from measurements or manuals, and uniquely determines the degradation behaviour of the component, viz. the rate of the exponential random variables. All EBEs in Fig. 6 are derived from Table 1, originally obtained from ASHRAE (1996); Faisal and Mahmoud (2003). In the table, N is the number of degradation phases and MTTF is the mean time to failure: for instance, EBE 2 models a failure of the supply fan motor via a (random variable with distribution) $\mathrm{Erlang}(3, {}^{3}/_{35})$.

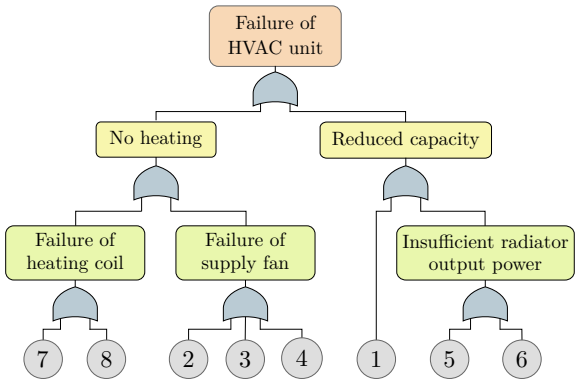

Figure 6. FMT model of the HVAC from Fig. 2

To allow differentiated maintenance actions, for all EBEs we label the degradation phases, i.e. we label states in CTMCs and locations in PTA. With `new` we label the initial phase of an EBE, corresponding to a new component that starts operation in perfect condition. With `failed` we label the last phase, e.g. phase 3 for EBE 2, corresponding to a failure in the component that will propagate in the tree to cause a TLE. With `thresh` we label all other phases to indicate a degraded—but still functional—condition of the EBE.

EBE ID	Failure mode	N	MTTF (years)
1	AHU damper broken	4	20
2	Fan motor failure	3	35
3	Supply fan obstructed	4	31
4	Fan bearing failure	6	17
5	Radiator failure	4	25
6	Radiator stuck valve	2	10
7	Heater stuck valve	2	10
8	Heat pump failure	4	20

Table 1. Details of the EBEs from Fig. 6

4.1. Maintenance policies

In Cauchi et al. (2017), maintenance schemes distinguish between *inspections*, *repair checks*, and *overhauls*, which in our setting take place every half, two, and fifteen years respectively. We term this policy "full maintenance," and experiment also with "half maintenance," where the time periods are respectively one, four, and thirty years for inspections, repair checks, and overhauls.

Each of these periodical service activities has a clear and distinct restoration purpose:

- Overhauls enforce a *replace* action that renews the whole HVAC, sending all EBEs back to their `new` phase.
  - When triggered, replace actions take one week (i.e. seven days) to complete.
  - This incurs in a cost of €5000 and is the highest point-wise investment of the whole policy.
- Repair checks can trigger a *repair* action, but will do so if and only if some EBE is in its `failed` phase at the moment when the check takes place. The intuition is that an expert technician visits the premises regularly, fixing any component he finds in a `failed` state, namely broken.
  - When triggered, repair actions take two days.
  - All broken EBEs are repaired in those two days, and thus the repair time is independent of the number of broken components. This modelling assumption can be easily generalised; however, we choose to follow the scheme from Cauchi et al. (2017).
  - Repaired EBEs have their health significantly restored: once the action is completed they are sent back $N-2$ degradation phases. The resulting phase for the component is the first one labelled `thresh` after the `new` phase, i.e. a repair leaves the component "almost—but not quite—new."
  - A repair costs €800; notice nonetheless that, if no component is broken when the repair check takes place, no repair action is effectively triggered and thus the cost is not incurred.
- Inspections can trigger a *clean* action, but will do so if and only if some EBE is in a `thresh` phase (aka degraded but operational) at the moment when the inspection takes place.
  - Every inspection incurs a cost of €5.
  - When triggered, clean actions take a day.
  - All degraded EBEs are cleaned in that day, following the same scheme implemented for repairs.
  - When an EBE is cleaned, its health is restored by one degradation phase, which makes cleaning the lightest among all maintenance actions.
  - A clean costs €100; however notice that, if no component is degraded when the inspections occurs, no clean is performed and this cost is not incurred.

These maintenance procedures are not explicit in the FMT from Fig. 6, but are implicitly implemented in the corresponding repair module (RM) of the model. Moreover, all costs described above account for maintenance checks/actions. We also consider operational costs: €1 is accrued per day of system uptime, and €4 per day of system downtime (Cauchi et al., 2017; Macek et al., 2017).

4.2. PMC via PRISM

As described in Secs. 1 and 3, the FMT must be instantiated in order to compute KPI metrics. We use CTMCs as modelling formalism to carry out analyses via PMC. We employ the PRISM model checker (Kwiatkowska, Norman, & Parker, 2011) to encode the model and perform the queries. This software tool has been thoroughly tested and stands among the most commonly used probabilistic model checkers in the formal methods community of computer science. Modelling in PRISM is done by means of a text format, where states are encoded as (discrete and bounded) variables valuations, and transition rates are written on the same line than the enabling guard and the produced action.

```
1  ctmc
2  const int YEAR = 364;
3  const int DEG_N1 = 5;
4  const double DEG_MTTF1 = 20.0*YEAR;
5  module EBE1
6      s1 : [0..DEG_N1];
7      // Kickstart
8      [trigger] s1 = 0 -> (s1'=1);
9      // Degradation
10     [  ] 0 < s1 & s1 < DEG_N1-1 ->
11                 (DEG_N1-1.0)/DEG_MTTF1 : (s1'=s1+1);
12     [f1]          s1 = DEG_N1-1 ->
13                 (DEG_N1-1.0)/DEG_MTTF1 : (s1'=s1+1);
14     // Maintenance: clean
15     [cln] 1 = s1                  -> true;
16     [cln] 1 < s1 & s1 < DEG_N1    -> (s1'=s1-1);
17     [cln] s1 = DEG_N1             -> true;
18     // Maintenance: repair
19     [rep] s1 < DEG_N1             -> true;
20     [rep] s1 = DEG_N1             -> (s1'=2);
21     // Maintenance: replace
22     [rplc] true                   -> (s1'=1);
23 endmodule
```

Code 1. PRISM CTMC snippet for EBE 1

In Code 1 we show the CTMC model for EBE 1 described in the syntax of PRISM. The local states of the module are encoded by the valuations of variable `s1`. The first transition is on line 8: it can *fire* if `s1` has the value 0, in which case it updates that value to 1. The transition rate is omitted and thus implicitly assumed $= 1$. Instead, in lines 10 and 11 the rate appears explicitly: the value is `(DEG_N1-1.0)/DEG_MTTF1`, i.e. $(5-1)/(20*364) \approx 0.000549$. Module synchronisation is done by means of synchronisation labels, and transitions with the same label in all system modules must occur simultaneously. For instance, the transition on line 22 is always enabled for module `EBE1`, but it will only fire in synchrony with all other modules with transitions labelled `rplc`. In particular that transition encodes the completion of a replacement action, which must have been triggered by an overhaul.

Property queries in PRISM follow their own specification language, which for CTMCs corresponds to CSL. Usually, "formulae" are defined that can inspect the internal state of modules like `EBE1`. These formulae can be labelled and used in quantitative queries like those described in Sec. 3.3.1. In

Code 2 we show the query to quantify (un)reliability for the EBE1 module of Code 1, in the time window $[0, T]$.

```
1 formula Failure = (s1=DEG_N1);
2 label "failure" = (Failure);
3 P=? [ F<=T "failure" ]  // un-reliability
```

Code 2. PRISM property queries

4.3. SMC via UPPAAL

In parallel, we use PTA as modelling formalism to carry out SMC analyses. We employ the UPPAAL toolset (Larsen, Pettersson, & Yi, 1997; David et al., 2015) to encode the model and perform the queries. Just like PRISM in the field of Markovian systems, UPPAAL is a prevalent tool in the field of TA and related formalisms like PTA. In particular, it provides statistical model checking functionalities which we use to perform our studies. To model PTA in UPPAAL a GUI is available: an automaton is graphically described as a graph, where locations are the nodes and PTA edges are the edges in the graph. A small example modelling the TLE is depicted in Fig. 7.

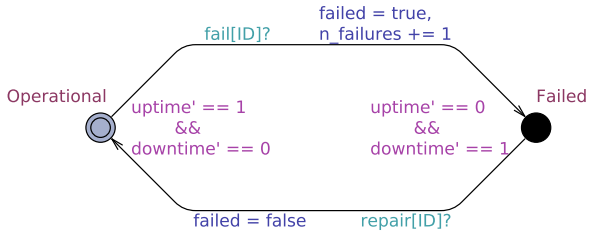

Figure 7. UPPAAL PTA snippet for the TLE

The left location (Operational) models an HVAC operating properly. Time can pass in a location as long as its invariant is satisfied. In Fig. 7 all invariants (uptime' == $\cdots$) are always true[2], which means that time can elapse with no restrictions. When some component in the HVAC fails, a broadcast signal will be sent through the fail[$*$] communication channel. The TLE model in Fig. 7 will then synchronise and take the transition towards the right location (Failed), which models a non-operational system. As the transition is taken two variables are updated, namely the Boolean variable failed is set, and the integral variable n_failures is incremented.

Property queries for SMC analyses need extra information (with respect to PMC) to determine statistical convergence. In Code 3 this is explicit in the availability query for a 15 years time horizon on line 3: the literal 9999 defines the number of simulations to run for building the confidence interval which will be returned as the estimate. Instead, the reliability query on line 2 of Code 3 will converge when the width of the confidence interval is less than certain predefined precision value. This numeric value is an option set on the GUI of the tool, and does not appear explicitly on the query.

[2] Technically speaking, invariants in Fig. 7 are used to control time drift in the clocks, and impose no condition on the passage of time in locations.

```
1 const int YEAR = 364;
2 Pr [ <=15*YEAR      ] (<> TLE.failed)
3 E  [ <=15*YEAR;9999 ] (max: TLE.downtime/(15.0*YEAR))
```

Code 3. UPPAAL property queries

In the queries from Code 3, TLE is the instance of the model depicted in Fig. 7, i.e. the PTA representing the top level event of the FMT. In turn, failed and downtime are internal variables of the model: failed is the Boolean variable set when the HVAC is not operational, and downtime is the real-valued clock whose time drift is modified in the locations of the TLE model from Fig. 7, and which serves to keep track of the system downtime.

5. RESULTS

In Sec. 4 we describe the modelling frameworks used to analyse the HVAC FMT. In this section we present the metrics computed using both (quantative model checking) techniques. More specifically, we show plots describing system reliability, availability, ENF, and differentiated costs, for both maintenance policies at various time horizons. We compare these metrics from different angles, to further highlight the trade-off associated with each methodology.

5.1. Model checking of FMT

We begin by showing that the models developed in both tools, i.e. PRISM and UPPAAL, are actually encoding the same system behaviour. This is relevant because the CTMC and PTA models of the FMT from Fig. 6 are independent. Thus, evidence should be provided showing that the modelling from Secs. 4.2 and 4.3 yields equivalent semantics for the FMT.

Specifically, for some fixed confidence level (e.g. 95%) and time horizon T, a KPI value yielded by PMC on the CTMC coincides with an estimate (confidence interval) yielded by SMC on the PTA, if the interval contains the value. As highlighted in Sec. 3.3.2, the state space representation for analysis via PMC may require a large amount of memory. In particular, modelling the whole FMT using PRISM (called *PMC exact*) requires more than 10^8 states, which is an upper bound for the capabilities of the tool in current computer settings (Kwiatkowska, Norman, & Parker, 2017). A novel state-space reduction technique is proposed in Cauchi et al. (2017), whose essence is to abstract away IEs, replacing them for EBEs with the equivalent MTTF of the IE. The resulting reduced CTMC allows analysing the (approximate) behaviour of the whole FMT using PRISM: we call *PMC reduced* the analyses performed on such models.

To demonstrate that our CTMC and PTA models coincide, we first focus on the subtree from Fig. 6 with IE labelled

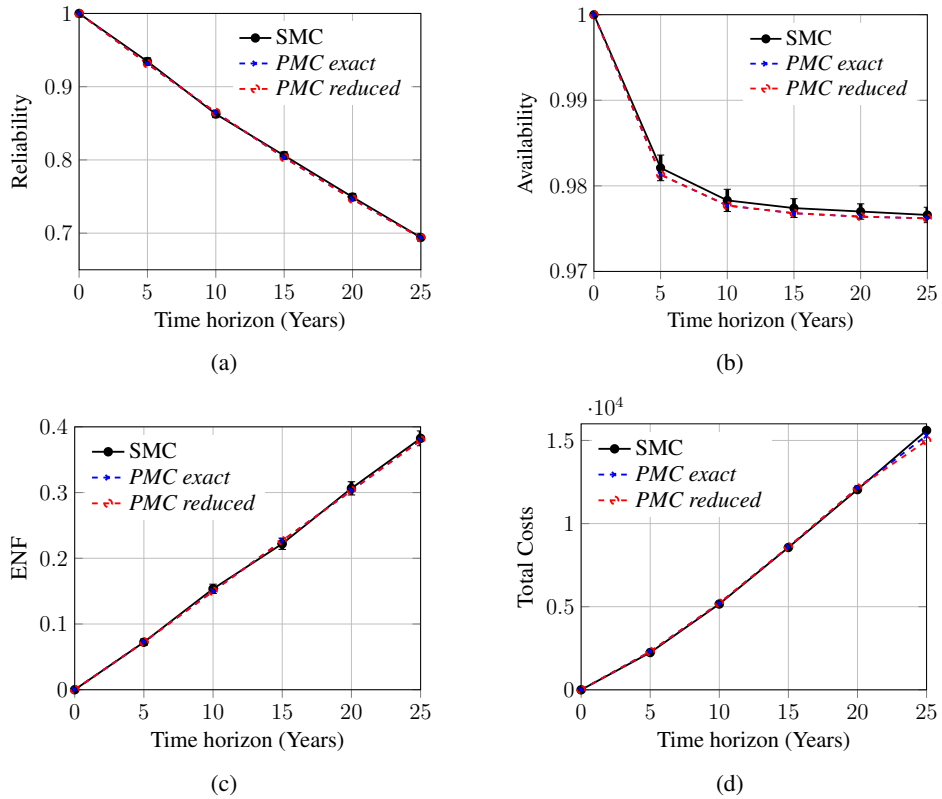

Figure 8. Model checking for FMT: Reduced capacity

"Reduced capacity," composed of EBEs 1, 5, and 6. We experiment with the full maintenance scheme, i.e. inspections every 6 months, repair checks every 2 years, and overhauls every 15 years. For this subtree all three analysis techniques (*PMC exact*, *PMC reduced*, and SMC) can be used to estimate the KPI metrics listed in Sec. 3.2, viz. reliability, availability, ENF, and total costs. In Fig. 8 we show the values computed for $T = 0, 5, 10, \ldots, 25$ years: estimations coincide between SMC and both PMC analyses in all cases.

As expected, a small gap is observed between *PMC exact* and *PMC reduced*, see e.g. Fig. 8b. This is a consequence of the reduction technique, which replaces the OR gate on top of EBEs 5 and 6 for an EBE with equivalent MTTF. Thus instead of using the fastest firing time between two Erlang distributions, i.e. one for EBE 5 and one for EBE 6, in *PMC reduced* we measure the firing time of a single EBE with the same MTTF. This has the advantage of reducing the number of states of the CTMC analysed (everything below the replaced IE is now a single EBE), but creates a minor deviation in the general model behaviour. Such deviation should increase with the number of times this reduction is performed.

These reasonings are corroborated by our studies. In Fig. 9 we show the KPI metrics for a subtree that includes the IEs "Reduced capacity" and "Failure of heating coil," i.e. EBEs 1 and 5–8. Again we experiment with the full maintenance policy on five time horizons. In the figure we observe that SMC and *PMC exact* coincide in all values computed, but the difference with *PMC reduced* is exacerbated, since more IEs have been reduced via the abstraction method.

We highlight that Figs. 9e to 9h display the different contributions to the maintenance costs. The aggregation of these values, plus the operational system costs, composes the "Total Costs" from Fig. 9d. In particular Fig. 9h depicts the money spent in replacements for time horizons $T = 0, 5, 10, \ldots, 25$. Notice that the overhaul period for full maintenance is 15 years, yet the plot shows positive costs incurred in replacements at 5 and 10 years. This is a consequence of approximating discrete time periods via 3-phase Erlang distributions: the *mean time* for an occurrence of the event, viz. an overhaul triggering a replace, *is 15 years*, but the effective time observed in particular simulations variates to some degree[3] around such value, causing the (averaged) behaviour shown in Fig. 9h. Research is currently being done to simulate "truly deterministic" time periods rather than Erlang approximations. When deterministic times are used, an increase of €5000 must be observed for the replace costs exactly at 15 years (and 30 years and so on), and not before.

Finally, we present our studies on the full HVAC model from

[3] A random variable following an $\mathrm{Erlang}(3/15, 3)$ distribution has standard deviation $\sqrt{\frac{3}{(3/15)^2}} = 5\sqrt{3}$.

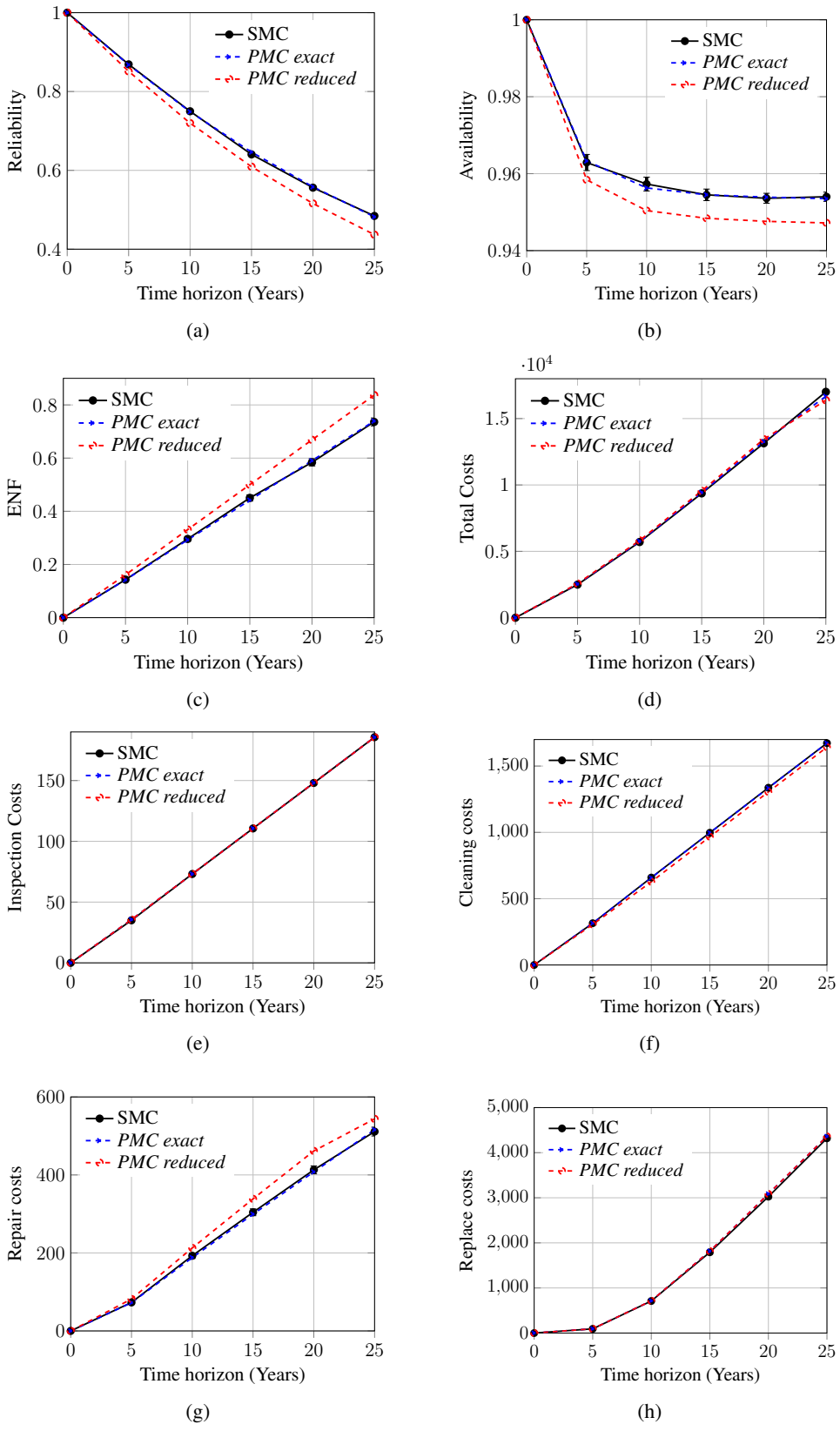

Figure 9. Model checking for FMT: Reduced capacity & Failure of heating coil

Fig. 6. In this case and as previously discussed, *PMC exact* cannot be used due to state space explosion and physical memory constraints. As a result only SMC and *PMC reduced* are compared, measuring the KPIs in seven time horizons for both maintenance policies, viz. full and half maintenance. The outcomes are presented in Fig. 10: the difference between both analysis techniques is analogous to the one observed in Fig. 9, and therefore falls within the acceptable range.

5.2. Comparison of maintenance policies

So far the focus has been on correctness, and in Sec. 5.1 we have shown that both PMC and SMC are capable of providing concrete (and matching) KPI metrics for the FMT from Fig. 6. In this section we use the values presented in Fig. 10 to assess the RAMS properties of the HVAC model. Concretely, we compare and discuss the two maintenance policies studied, i.e. (i) full maintenance, for which inspections, repair checks, and overhauls, take place every half, two, and fifteen years respectively, and (ii) half maintenance, where they take place every one, four, and thirty years respectively.

First and foremost, we notice that system resilience is significantly affected when the service periods are halved. For time horizons $T = 5, 10, 15, \ldots, 35$, reliability is lower for half maintenance than full maintenance by factors of approximately 11%, 24%, 37%, 50%, 60%, 70%, and 82%, see Fig. 10a. In turn, availability is lower for half maintenance by approximately 6%, 9%, 11%, 12%, and 13% for the last three time steps, see Fig. 10b. Lastly, ENF with half maintenance is consistently around twice that of full maintenance for all seven time horizons, see Fig. 10c. Hence, it is abundantly clear that in terms of system resilience, the full maintenance policy is much more effective than half maintenance.

Given those values for the KPIs measuring system resilience, it is very interesting to see in Fig. 10d that the costs incurred by either policy differ by a very small amount, for all time horizons. This is a result of the interplay among: the extra costs incurred in maintenance for the full policy, the higher availability this generates for the system, and the operational costs considered, in which there is a 1-to-4 ratio between system uptime and downtime costs. More in detail, the half maintenance policy accrues to less expenses derived from maintenance actions than the full policy. The aggregated costs incurred by all maintenance actions (see Figs. 10e to 10h) show that the full maintenance policy is more expensive than half maintenance by 57%, 64%, 70%, 70%, 67%, 63%, and 60% for time horizons $T = 5, 10, \ldots, 35$. Nonetheless, using half maintenance results in more system downtime, which accrues €4 per day. This should be contrasted with the €1 per day accrued per day of system uptime.

The exact amount of money spent in maintenance/operational costs is also relevant. With the full policy, maintenance costs grow in a linear fashion from $\approx$ €650 to €11135. These values account for 30% (5 years) to 76% (35 years) of the operational system costs. In contrast, with half maintenance, the increase in maintenance costs is also linear and goes from $\approx$ €350 to €6000, for values which account for 15% (5 years) to 31% (35 years) of the operational system costs. In both cases and on average, the operational costs have a greater impact than maintenance costs in the total expenses.

In summary, the full policy has higher maintenance expenses but achieves higher availability. This means that the operational costs are somewhat lower for full than for half maintenance. The exact difference in system availability between both policies is key, as well as the cost ratio between the operational costs for system downtime and uptime. The general outcome of all these factors, as observed in Fig. 10d, is that the total costs are practically the same with both maintenance policies. Therefore, *the full policy is evidently better than half maintenance*, given the higher system resilience it yields.

As a final remark for this section we mention that, when different maintenance actions are involved, like the clean-repair-replacement of the setting studied, system dynamics can become complex and difficult to predict. This is illustrated by Figs. 10f and 10g, where we observe that the expenses to *clean* components are higher for full maintenance, but those to *repair* components are higher for half maintenance. The reason is that clean are triggered by inspections, which occur more frequently in full maintenance. In comparison, in half maintenance, cleans are triggered less often and components can degrade more, and thus the likelihood of observing a failed EBE is higher. Since repairs only affect EBEs in its `failed` phase, this explains why half maintenance incurs in more costs for repairs than the full policy. This may not seem particularly hard to foretell; nonetheless it is a priori less obvious that, in spite of such more efficient cleans in the full policy, the aggregated maintenance costs incurred are lower for the half maintenance policy in general, mostly due to costs derived from replacements (see Fig. 10h).

5.3. Comparison of computational times

Lastly, we compare the computational performance of SMC vs. *PMC reduced*, for the analyses used to produce the values presented in Fig. 10. SMC experiments have been run in a cluster with AMD Opteron™ 4386 processors, each with access to 64 GB of dedicated DDR3 RAM. PMC experiments have been run in a cluster with Intel® Xeon® E5-2640 v3 processors, each with access to 64 GB of DDR4 RAM. The resulting execution times in seconds of both techniques, for both maintenance policies and all KPIs, are listed in Table 2.

In the case of *PMC reduced*, the times listed include the computation of the MTTF used to perform the IE-to-EBE reduction, plus the subsequent time it took PRISM to compute the corresponding KPI from the resulting reduced model. Since

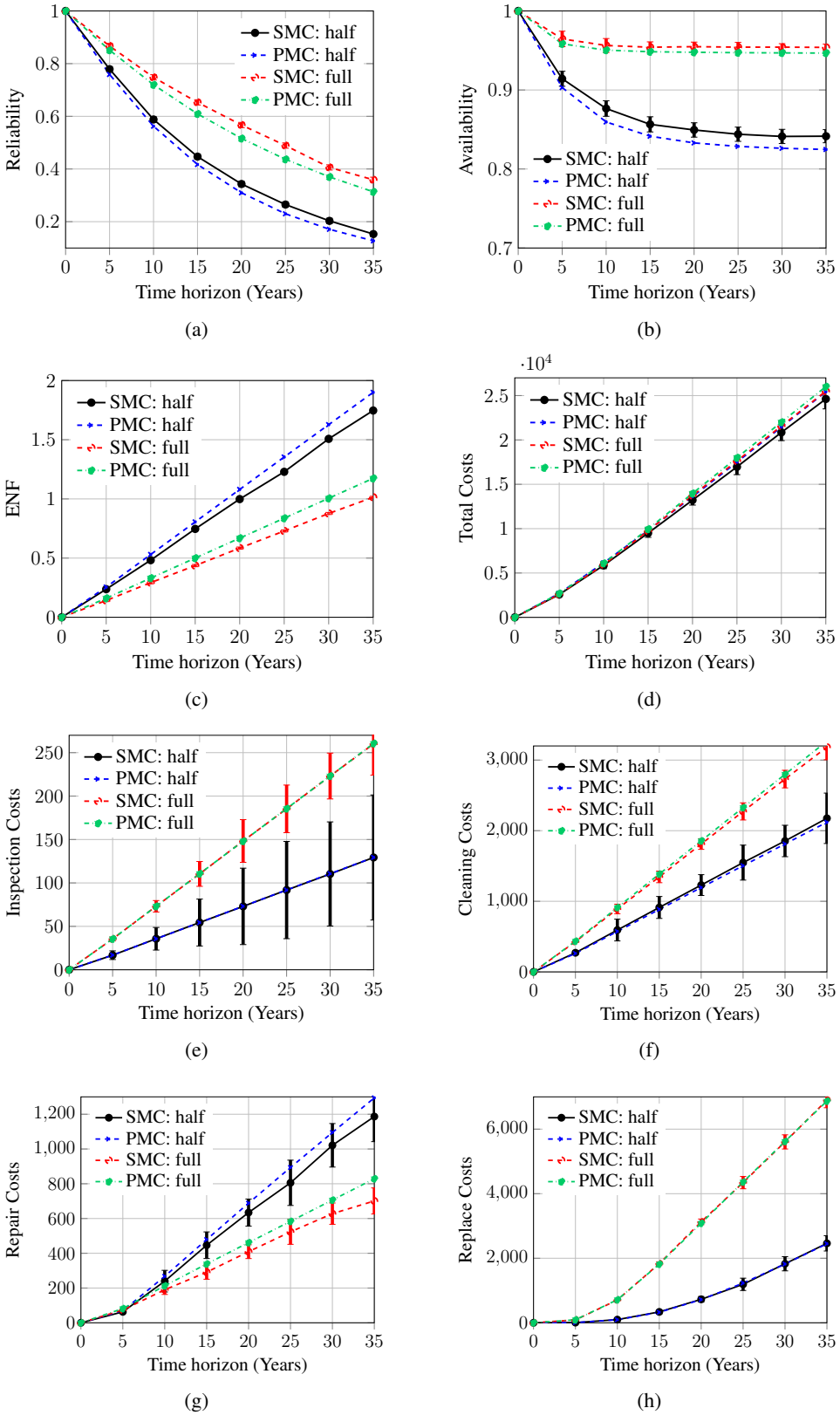

Figure 10. Model checking for FMT: complete HVAC, half & full maintenance

T	Metric	PMC		SMC		Metric	PMC		SMC	
		half	full	half	full		half	full	half	full
5	**Reliability**	0.6	0.8	6.3	6.1	**Availability**	1.0	1.1	7.9	13.1
10		1.6	1.2	14.4	17.0		2.3	2.1	15.5	25.4
15		2.0	1.7	17.4	29.0		3.2	3.5	24.9	35.8
20		2.3	2.9	19.5	38.6		4.4	4.1	31.2	43.2
25		3.0	3.2	17.9	46.2		5.5	3.4	31.6	52.2
30		3.4	3.6	16.1	49.8		7.3	3.2	35.5	57.6
35		4.3	4.4	13.9	52.4		8.2	3.5	38.9	61.9
5	**ENF**	1.1	1.2	7.9	12.9	**Total costs**	5.4	5.2	29.8	47.9
10		2.2	2.3	15.4	24.0		10.5	9.4	56.4	90.0
15		3.2	3.5	21.3	34.2		14.0	13.9	78.5	119.4
20		4.9	4.0	27.7	43.3		17.6	17.1	94.7	145.6
25		5.6	4.4	31.6	50.1		22.1	19.7	105.7	168.3
30		6.2	4.3	36.0	55.9		27.0	22.1	116.4	181.0
35		7.0	4.5	38.2	61.5		31.7	23.6	120.0	187.0

Table 2. Time in seconds to compute KPIs using PMC and SMC, for half and full maintenance policies

the process of plugging in the computed MTTF into the reduced CTMC (from which PRISM computes the metric) is not yet automated, that is omitted from our measurements. We note however that, if the task were fully automated, such time should be negligible with respect to the "true analysis times" required to compute a KPI value.

Table 2 shows that, as expected, PMC converges to a result in significantly less time than SMC. For reliability, availability, and ENF, PMC is usually $\approx 10\times$ faster than SMC; for costs computation the difference is $\approx 6\times$ in favour of PMC. In general, this is a consequence of SMC requiring a sufficiently large random sample (viz. number of simulations) to achieve the desired statistical criteria and build the confidence interval. Here, the underlying trade-off is in runtime vs. computer memory: "standard" PMC, i.e. *PMC exact*, could not be used because its state space (and thus computer memory) requirements go beyond current computer settings. A reduction technique must be used to alleviate this problem, losing in exchange some precision in the metrics. On the other hand, it must be highlighted that the confidence criteria requested for our SMC analyses is standard but not particularly challenging. Requesting 99% confidence level and more precise (narrower) confidence intervals would greatly increase the computation times of SMC.

For both techniques and all metrics, computation times increase linearly with the time horizon analysed. However, PMC is not greatly affected by the maintenance policy studied, because its computational times are mostly a function of the structure of the underlying model. This structure is given by the set of (CTMC) modules and their communication mechanisms, rather than by the particular parameters like e.g. inspection frequency. Such parameters do play a minor role, since they may influence the number of iterations required to determine convergence to a fixed-point in the internal mechanisms of the model checking algorithms. Incidentally, the effect of these parameters in the number of iterations could be counter-intuitive: ENF, availability, and costs computations with PMC are faster for full maintenance than for half maintenance, see Table 2. In any case, PMC is much less affected by the maintenance policy than SMC. Since SMC uses discrete event simulation as its backbone, the more events per time unit that need to be attended, the more computation time it takes to advance one time unit. On average, the full policy generates twice as many events per time unit than half maintenance, and therefore it takes longer to compute the metrics for full policy. The system dynamics are quite complex, see e.g. the final discussion in Sec. 5.1 on the number of cleans vs. the number of repairs for full and half maintenance. Consequently, *quantifying* how longer will SMC analyses take for the full policy is not straightforward. It is nevertheless clear that half maintenance is faster to analyse via SMC than full maintenance, which is corroborated in practically all time measurements presented in Table 2.

6. CONCLUSIONS

In this work we study the RAMS properties of an HVAC unit, applying formal methods to perform fault tree analysis. We model the failure modes of the system in the presence of maintenance actions by means of an FMT, to which we give semantics using the CTMC and PTA formalisms. We then analyse the resulting model(s) by computing KPI metrics via PMC and SMC respectively for each formalism. The goal is twofold: assessing system RAMS properties under two different maintenance policies, namely full and half maintenance; and comparing the PMC and SMC approaches to perform such assessment.

To validate our studies we first confirm the equivalence between the two (concrete models and) techniques in Sec. 5.1, where we highlight the need for a state-space reduction mechanism to enable PMC analyses. Next, we analyse the two maintenance policies in Sec. 5.2, and show that system resilience in terms of reliability, availability, and ENF, is con-

siderably higher under the full maintenance policy than with half maintenance. Furthermore, the total costs incurred are very similar with both policies. We conclude that full maintenance is better than half maintenance, answering our first goal, and highlighting at the same time the benefits of using FMTs during maintenance planning and assessment.

As for the second goal, we report in Sec. 5.3 that PMC is significantly faster to compute all RAMS metrics. This however comes at the expense of applying a state-space reduction technique on the CTMC model, which generates a (predictable) bias in the metrics. Alternatively, SMC can study more general models with less design effort, e.g. using deterministic times should be straightforward. The main drawback of SMC is the need for substantially longer computation times in general, which may also depend on parameters like the frequency of maintenance actions (which have much less impact on the computation times of PMC) or the precision desired for the confidence interval to be generated.

6.1. Future work

Currently in the PMC community work is being done towards extending quantitative model checking to PTA (Kwiatkowska et al., 2018). This is the first natural extension of our work. Analysing PTA via PMC would reduce the design effort required: by using PTA over CTMC we would no longer need to perform the state-space reduction, and thus the accuracy of the computed values should improve, at the same time maintaining the high computational performance of PMC

We are also planning to extend our research in the following directions:

- Automate the state-space reduction algorithm required to perform PMC on large models (see Sec. 5.1);
- Compare and contrast a larger set of maintenance polices for this case study with various time horizons;
- Explore different trade-off schemes between operational and maintenance costs;
- Validate the results computed for our models against KPI metrics obtained from real data.

Note however that, due to the slow degradation rate of HVAC components, obtaining the data required to perform the last task could be particularly involved.

ACKNOWLEDGEMENTS

We would like to thank Arnaud van Harmelen and Enno Ruijters for fruitful discussions that helped in developing the PTA semantics of the HVAC FMT model.

This work is partially supported by the Alan Turing Institute, UK; Malta's ENDEAVOUR Scholarships Scheme; and the NWO SEQUOIA project.

REFERENCES

Alaswad, S., & Xiang, Y. (2017). A review on condition-based maintenance optimization models for stochastically deteriorating system. *Reliability Engineering & System Safety*, *157*, 54–63.

ASHRAE. (1996). HVAC systems and equipment. *American Society of Heating, Refrigerating, and Air Conditioning Engineers, Atlanta, GA*.

Aziz, A., Sanwal, K., Singhal, V., & Brayton, R. (2000). Model-checking continuous-time markov chains. *ACM Transactions on Computational Logic (TOCL)*, *1*(1), 162–170.

Behrmann, G., Larsen, K. G., & Rasmussen, J. I. (2004). Priced timed automata: Algorithms and applications. In *International Symposium on Formal Methods for Components and Objects* (pp. 162–182). Springer.

Bengtsson, J., & Yi, W. (2003). Timed automata: Semantics, algorithms and tools. In *Advanced Course on Petri Nets* (pp. 87–124).

Cauchi, N., Hoque, K. A., Abate, A., & Stoelinga, M. (2017). Efficient probabilistic model checking of smart building maintenance using fault maintenance trees. In *Proceedings of the 4th ACM International Conference on Systems for Energy-Efficient Built Environments* (p. 24).

Clarke, E., Emerson, E., & Sistla, A. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, *8*, 244–263.

Clarke, E., Grumberg, O., & Peled, D. (1999). *Model checking*. MIT press.

David, A., Larsen, K., Legay, A., Mikučionis, M., & Poulsen, D. (2015). Uppaal SMC tutorial. *International Journal on Software Tools for Technology Transfer*, *17*(4), 397–415.

Dong, B., Lam, K., & Neuman, C. (2011). Integrated building control based on occupant behavior pattern detection and local weather forecasting. In *Twelfth International IBPSA Conference. Sydney: IBPSA Australia* (pp. 14–17).

Durkin, T. H. (2006). Boiler system efficiency. *ASHRAE Journal*, *48*(7), 51.

Faisal, I. K., & Mahmoud, M. H. (2003). Risk-based maintenance (RBM): a quantitative approach for maintenance/inspection scheduling and planning. *Journal of Loss Prevention in the Process Industries*, *16*(6), 561–573.

Haesaert, S., Cauchi, N., & Abate, A. (2017). Certified policy synthesis for general markov decision processes: An application in building automation systems. *Performance Evaluation*, *117*, 75–103.

Han, Z., Gao, R. X., & Fan, Z. (2012, May). Occupancy and Indoor Environment Quality Sensing for Smart

Buildings. In *Instrumentation and Measurement Technology Conference (I2MTC), 2012 IEEE International* (pp. 882–887).

Kim, W., & Katipamula, S. (2017). A review of fault detection and diagnostics methods for building systems. *Science and Technology for the Built Environment*, 1–18.

Kwiatkowska, M., Norman, G., & Parker, D. (2007). Stochastic model checking. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems* (pp. 220–270).

Kwiatkowska, M., Norman, G., & Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In *CAV* (Vol. 6806, pp. 585–591). Springer.

Kwiatkowska, M., Norman, G., & Parker, D. (2017). Symbolic verification and strategy synthesis for linearly-priced probabilistic timed automata. In *Models, Algorithms, Logics and Tools: Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday* (Vol. 10460, pp. 289–309). Springer.

Kwiatkowska, M., Norman, G., & Parker, D. (2018). Probabilistic model checking: Advances and applications. In *Formal System Verification* (pp. 73–121). Springer.

Larsen, K. G., Pettersson, P., & Yi, W. (1997). UPPAAL in a nutshell. *International journal on software tools for technology transfer*, *1*(1-2), 134–152.

Legay, A., Delahaye, B., & Bensalem, S. (2010). Statistical model checking: An overview. In *International Conference on Runtime Verification* (pp. 122–135).

Lindelöf, D., Afshari, H., Alisafaee, M., Biswas, J., Caban, M., Mocellin, X., & Viaene, J. (2015). Field tests of an adaptive, model-predictive heating controller for residential buildings. *Energy and Buildings*, *99*, 292–302.

Macek, K., Endel, P., Cauchi, N., & Abate, A. (2017). Long-term predictive maintenance: A study of optimal cleaning of biomass boilers. *Energy and Buildings*, *150*, 111 - 117.

Nicolai, R. P., & Dekker, R. (2008). Optimal maintenance of multi-component systems: a review. In *Complex System Maintenance Handbook* (pp. 263–286). Springer.

Ruijters, E., Guck, D., Drolenga, P., Peters, M., & Stoelinga, M. (2016). Maintenance analysis and optimization via statistical model checking. In *International Conference on Quantitative Evaluation of Systems* (pp. 331–347).

Ruijters, E., Guck, D., Drolenga, P., & Stoelinga, M. (2016). Fault maintenance trees: reliability centered maintenance via statistical model checking. In *Reliability and Maintainability Symposium (RAMS), 2016* (pp. 1–6).

Trojanova, J., Vass, J., Macek, K., Rojíček, J., & Stluka, P. (2009). Fault diagnosis of air handling units. *IFAC Proceedings Volumes*, *42*(8), 366–371.

Vesely, W., Goldberg, F., Roberts, N., & Haasl, D. (1981). *Fault Tree Handbook* (Tech. Rep.). Nuclear Regulatory Commission Washington DC.

Vesely, W., Stamatelatos, M., Dugan, J., Fragola, J., Minarick, J., & Railsback, J. (2002). Fault tree handbook with aerospace applications version 1.1. *NASA Office of Safety and Mission Assurance, NASA HQ*.

Volk, M., Junges, S., & Katoen, J. P. (2018). Fast dynamic fault tree analysis by model checking techniques. *IEEE Transactions on Industrial Informatics*, *14*(1), 370-379.

Volk, R., Stengel, J., & Schultmann, F. (2014). Building information modeling (bim) for existing buildings — literature review and future needs. *Automation in Construction*, *38*, 109 - 127.

Younes, H. L. S., & Simmons, R. G. (2002). Probabilistic verification of discrete event systems using acceptance sampling. In *CAV* (Vol. 2404, pp. 223–235). Springer.

Zhao, H., Qi, Z., Wang, S., Vafai, K., Wang, H., Chen, H., & Tan, S. X. D. (2016). Learning-based occupancy behavior detection for smart buildings. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 954–957).