Revisiting a Pioneering Concurrent Stochastic Problem: The Erlangen Mainframe

Hubert Garavel[1], Holger Hermanns[2], and David Parker[3]

[1] Univ. Grenoble Alpes, Inria, Cnrs, Grenoble Inp, Lig, F-38000 Grenoble, France
[2] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[3] University of Oxford, Oxford, United Kingdom
hubert.garavel@inria.fr, hermanns@cs.uni-saarland.de,
david.parker@cs.ox.ac.uk

Abstract. The present article is an essay in research reproducibility after thirty years. We retrospectively consider a challenging problem proposed in 1994 by Ulrich Herzog and Vassilis Merksiotakis. This problem was about a multiprocessor computer, the Erlangen mainframe, that processes jobs of different priorities and is subject to hardware failures. Using the stochastic process algebra TIPP, a formal model of this mainframe was specified, which makes intensive use of parallel composition, multiway synchronisation between two or more concurrent processes, and compound transitions combining synchronised actions with rates of Continuous-Time Markov Chains. From this formal model, probabilistic results about availability, performability, and proper dimensioning of the mainframe were obtained using the TIPP software tools, which are no longer maintained. We investigate whether the same experiments can be reproduced today using state-of-the-art model checkers such as CADP, PRISM, and Storm.

1 Introduction

The present article was written in honour of Joost-Pieter Katoen and included in a collective *Festschrift* book offered to him on the occasion of his 60th birthday.

The topic of this article has a triple connection with the scientific works of Joost-Pieter Katoen. Firstly, it is about the formal modelling of stochastic systems, to which he has been contributing so actively [4] [20] [8] [9] [21] [29] [46] [32]. Secondly, it uses the Storm model checker, which has been developed by Joost-Pieter Katoen and his collaborators. Thirdly, it builds upon a case study proposed thirty years ago by the IMMD-7 team of Erlangen (Germany) headed by the well-known expert on performance analysis and queueing theory, Ulrich Herzog; at the time, Joost-Pieter Katoen was starting his postdoc in this team, which was striving for formal methods and tools able to overcome the inherent limitations of queueing network performance models, with a particular focus on process algebraic concepts to master complexity — a topic to which the PhD thesis of Joost-Pieter Katoen had substantially contributed [45].

Over the last three decades, the scientific field of formal methods for performance analysis has flourished, leading to a common understanding of the

various ingredients that make up usable formal models of stochastic timed systems. Firstly, one or multiple *state-transition machines* are needed, which may each be extended with state variables and described in various ways, e.g., using process algebras. Secondly, these models often include *rate transitions*, each with some parameter $\lambda \in \mathbb{R}^+$ that corresponds to the elapse of a time period. For Continuous-Time Markov Chains (CTMCs, for short), which we use in this paper, the probabilistic duration of a transition is governed by an exponential distribution with parameter λ. Thirdly, a *parallel composition* operator supports running several of these machines concurrently, possibly forcing two or more machines to synchronise on certain transitions according to *actions* attached to the transitions.

Besides fundamental results, many modelling languages, compilers, and analysis tools have been developed[4] [30] [46]. It is therefore instructive to take a retrospective view and to try applying modern tools to early instances of formal stochastic modelling, in order to observe how science has progressed.

The publications made during the 1990s by the IMMD-7 team in Erlangen contain, in addition to pioneering ideas, many interesting case studies of formal methods for stochastic systems. From this list, we selected the "mainframe" example, which seems to be the oldest example they proposed. This example is described in two workshop papers published in 1994 [37, Sect. 4] and 1995 [35, Sect. 4]. It was formally described in the TIPP (Timed Processes and Performance Evaluation) process algebra and analysed with the TIPPtool software developed at Erlangen [34], which is no longer maintained. At first sight, the mainframe example exhibits suitable qualities: it sounds realistic; it seems detailed enough so that we do not need to invent missing information; its performance analysis generates eight figures that are tempting to reproduce using state-of-the-art tools.

Our challenge is thus formulated as follows: can we formally describe the mainframe example using more recent languages than TIPP, and can we reproduce in 2024 using modern tools the same experiments done thirty years ago with the TIPPtool? To this aim, we considered three well-known software tools:

- CADP[5] [25], which appeared before the TIPPtool, and has been developed since the late 1980s. For many years, CADP has been using LOTOS [43] as its input language (like TIPP, which was also based on LOTOS), but LOTOS has been progressively replaced by a more recent language named LNT. CADP provides most of the TIPPtool functionalities for analysing probabilistic and stochastic systems. CADP received the ETAPS Test-of-Time Tool Award in 2023.
- PRISM[6] [49], which appeared at the same time that the TIPPtool retired, and has been continuously developed since then. It is a widely-used and versatile tool, with support for an extensive range of probabilistic/stochastic

[4] `http://cadp.inria.fr/resources/zoo/`
[5] `https://cadp.inria.fr`
[6] `https://prismmodelchecker.org`

models, temporal logics and analysis techniques. PRISM received the ETAPS Test-of-Time Tool Award in 2024.
– Storm[7] [32], a more recent tool developed by Joost-Pieter Katoen and collaborators. Storm supports a broad range of probabilistic models, input formalisms and techniques, and has established itself as a high-performance and extendible tool for the verification of probabilistic/stochastic systems.

The present article is organised as follows. Section 2 presents the Erlangen mainframe, lists the minimal requirements that a specification language should satisfy to model this system properly, and discusses a few errors and ambiguities found in the original papers [37] [35]. Sections 3 and 4 report on the novel formal models that we developed for the Erlangen mainframe in the PRISM and LNT languages, respectively. Section 5 discusses to what extent we managed to reproduce the numerical experiments and the eight figures given in [37]. Finally, Sect. 6 provides concluding remarks and perspectives for future work.

2 The Erlangen Mainframe Modelled in TIPP

2.1 Description of the Erlangen Mainframe

The case study represents a multiprocessor mainframe that is designed to serve two purposes: (i) it has to maintain an important database and therefore has to process transactions submitted by a number of *users*, and (ii) it is used for program development and has to provide computing capacity to *programmers* for compiling and testing their programs. In addition, two interesting features are present:

– *Failures* may cause system downtimes, by making the mainframe become unavailable until it is repaired.
– Two types of *priorities* are built into the system. Database users need immediate reaction, so they *explicitly* have priority over the jobs issued by programmers. Failures cannot be preempted, which implies that they are neither buffered nor delayed; thus, they *implicitly* have the highest priority and take down the system immediately, until repair.

The description of the mainframe is highly modular and hierarchical (see Fig. 1). On the topmost level, the system is the parallel composition of three parts, the *Loads*, the *Queues*, and the *Processors*:

– *Loads*: There are three different arrival streams that put load on the system, namely the database *users*, the *programmers*, and *failures*. Each of these arrival streams produce events according to a given arrival rate. This rate however is not constant, but is instead modelled to vary according to a so-called Markov Modulated Poisson Process [22]. This means that each arrival stream has multiple phases (as in morning-afternoon-evening-night),

[7] `https://www.stormchecker.org`

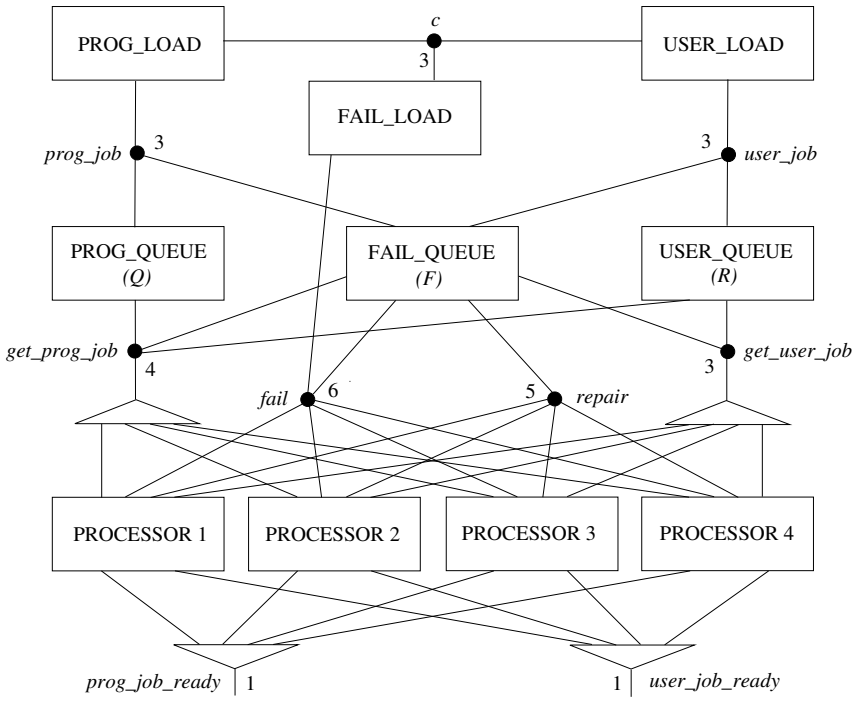

Fig. 1. Architecture of the Erlangen mainframe (black bullets represent n-ary synchronisations and white triangles represent 1-among-4 competitions for synchronisation)

and changing the phase comes with a change in arrival rate. The phase changes are governed by yet another rate, and happen synchronously across the streams; this is achieved by synchronising the three load processes, that otherwise run independently in parallel, on the phase change.

- *Queues*: The mediation between the events arriving from the loads and the processors is handled by three queues. The *user_queue* buffers the jobs generated by database users; the *prog_queue* buffers the jobs generated by programmers; the *fail_queue* reacts to failure events by triggering repairs. The priority mechanisms discussed above are implemented using clever synchronisations, ensuring that programmer jobs are only served if no user jobs are pending. Both types of jobs are, of course, processed only if the mainframe is not in a failure state.
- *Processors*: This part of the mainframe represents a multiprocessor consisting of four identical processors that run in parallel. The processors synchronise altogether on failures and repairs, meaning that failures affect the entire system, halting all processors, until repair.

The various interactions between the components described above (which will be represented by *processes* in our models) are shown in Fig. 1. Transitions performed simultaneously by multiple processes are modelled using *synchronisation*. In Fig. 1, the black bullets denote synchronisations, annotated by the number of processes participating and by an associated *action* (e.g., *prog_job* or *get_prog_job*). Furthermore, white triangles indicate situations where there is competition between several processes (here, always those modelling the four processors) to participate as a process in such a synchronisation.

2.2 Language Requirements for Modelling the Erlangen Mainframe

Because of its intrinsic features, the mainframe problem can only be modelled properly using languages that satisfy the following five requirements:

(a) It should be possible to express a parallel composition in which $n > 1$ client processes execute concurrently and compete together to establish binary synchronisations/communications with a server process. For instance, the four processors of the mainframe compete to communicate with the *Queues* process on actions *get_prog_job* and *get_user_job*.
(b) It should be possible to express multiway synchronisation (also known as n-party rendezvous), i.e., a parallel composition in which $n > 1$ execute concurrently and synchronise altogether on given actions. Multiway synchronisation is a powerful feature for modelling intricate systems [27] and, as such, is heavily used in the Erlangen mainframe, as illustrated by Table 1.

action	synchronisation pattern	rates
c	3-party rendezvous	φ
prog_job	3-party rendezvous	λ_1, λ_2
user_job	3-party rendezvous	μ_1, μ_2
fail	6-party rendezvous	δ_1, δ_2
repair	5-party rendezvous	β
get_prog_job	4-party rendezvous (3 queues, 1 processor)	α
get_user_job	3-party rendezvous (2 queues, 1 processor)	α
prog_job_ready	no rendezvous (interleaving)	ξ
user_job_ready	no rendezvous (interleaving)	ν

Table 1. Synchronised actions in the Erlangen mainframe

(c) It should be possible to express compound transitions of the form (a, λ), where a is an action (possibly synchronised with actions of other concurrent processes) and λ is the rate of an exponentially distributed delay. Indeed, all actions of the Erlangen mainframe are associated with a rate, as also shown by Table 1.

(d) It should be possible to express compound transitions of the form $(a, \mathbb{1})$, where a is an action and $\mathbb{1}$ denotes a neutral element that means "any rate". This neutral element, which exists in TIPP and other stochastic languages [39] [14] [10] as well, is such that the synchronisation of two transitions (a, λ) and $(a, \mathbb{1})$ results in a transition (a, λ). It is used in many places in the mainframe model, where one or multiple partners of synchronisation can be considered *passive*.

(e) It should be possible to mix, in the same sequential process, both types of transitions (a, λ) and $(b, \mathbb{1})$ — contrary to other formalisms, such as I/O automata, where active and passive actions must take place in different processes, as a means to avoid deadlocks. In the mainframe model, such a mix of transitions occurs in the three load processes and the four processors.

The three languages TIPP, PRISM, and LNT used to formally describe the Erlangen mainframe all satisfy these five requirements.

We have not yet discussed the semantics of synchronising two transitions (a, λ) and (a, μ). This has been a matter of debate at the times when stochastic process algebras like TIPP, PEPA [39], EMPA [10], and others [14] were conceived. Among the various approaches proposed, the TIPP semantics [37, rule $\langle \| \rangle$ of Fig. 1] states that the synchronisation of (a, λ) and (a, μ) produces a transition $(a, \lambda\mu)$. This "rate-product" semantics, which was also adopted by PRISM, TwoTowers [7], and GPA [15], generalises the above requirement (d) when $\lambda = \mathbb{1}$ or $\mu = \mathbb{1}$, assuming that $\mathbb{1}$ is understood as the rate 1.0 — yet, this semantics raises the issue of physical units, as rates λ and μ intuitively correspond to frequencies (i.e., the inverse of a duration), but their product does not.[8] Anyway, for the Erlangen mainframe, the rate-product semantics is not mandatory. Indeed, the mainframe model belongs to a particular subclass of TIPP models because it satisfies the following "one-to-many" property: in any n-party synchronisation (with $n > 1$), only one transition has the form (a, λ) while all other transitions have the form $(a, \mathbb{1})$. For instance, in the case of the *get_user_job* (resp. *get_prog_job*) transitions, the rate α is imposed by the *user_queue* (resp. *prog_queue*) processes, while the other processes (loads and processors) offer the neutral rate $\mathbb{1}$ for these transitions.

2.3 Issues in the Original Papers

While studying the mainframe model, we discovered various problems and ambiguities in the original papers. We briefly present these issues and explain how we addressed them:

(1) In the description of the processing unit [37, Sect. 4.2.2], there was a mistake: in the definition of PW_1, the action $(\mathit{user_job_ready}, \xi)$ should be replaced with $(\mathit{prog_job_ready}, \xi)$. This mistake was also present in the second paper [35, Sect. 4.2.2].

[8] There however is a "stoichiometric" interpretation (involving two reactants and a stochastic reaction constant) that can explain the phenomenon [28,14,11].

(2) The processes *ProgLoad*$_i$ and *FailLoad*$_i$ (where $i \in \{1, 2, 3\}$) are not specified in the original papers [37, Sect. 4.1] [35, Sect. 4.1]. As a first approximation, one may assume that they are pairwise similar to the processes *UserLoad*$_i$ (where $i \in \{1, 2, 3\}$), which are fully specified. Obviously, the rate parameters μ_1 and μ_2 of *UserLoad*$_i$ must be replaced by λ_1 and λ_2 for *ProgLoad*$_i$, and by δ_1 and δ_2 for *FailLoad*$_i$. The situation is more involved for the (c, φ) actions present in the *UserLoad*$_i$ processes. If one assumes that *ProgLoad*$_i$ and *FailLoad*$_i$ also propose these actions (c, φ), the "one-to-many" property is violated: the parallel composition of the load processes with 3-party synchronisation on c (which is explicitly mentioned in the original papers) should result, according to the TIPP rate-product semantics, in synchronised actions (c, φ^3). This does not seem realistic, given that $\varphi = 0.00334$: the value $\varphi^3 = 3.726 \times 10^{-8}$ is very small, and would be negligible compared to other rate transitions. By inspecting the source code of the TIPP models for the mainframe, to which we have access, we observed that *ProgLoad*$_i$ and *FailLoad*$_i$ propose actions $(c, \mathbb{1})$ instead of (c, φ), thus ensuring that 3-party synchronisations result in actions (c, φ) instead of (c, φ^3). Such a dissymmetry between the three load processes was not stated in the original papers; in our PRISM and LNT models for the mainframe (see Sect. 3 and 4), we use alternative modelling approaches that give the same results, but preserve the symmetry between the three load processes.

(3) There is a 5-party synchronisation on action *repair* between the failure queue and the four processors. However, the two original papers differ in the way rates are associated to the *repair* action. In the first paper [37, Sect. 4.2.1 and 4.2.2], the failure queue offers the action $(repair, \mathbb{1})$ while each processor proposes a $(repair, \beta/4)$ action; this violates the "one-to-many" property and, according to the semantics of TIPP, results in a synchronised action $(repair, (\beta/4)^4)$. In the second paper [35, Sect. 4.2.1 and 4.2.2], the failure queue offers the action $(repair, \beta)$ while each processor proposes a $(repair, \mathbb{1})$ action, leading to a synchronised action $(repair, \beta)$. We opted for the latter model, which is simpler and, based on the source code of the original TIPP models, seems to have been used for experiments.

(4) In both original papers, the machine is defined as a parallel composition involving four processors P, i.e., $Machine = Queues \,||_B \, (P \,||_C \, P \,||_C \, P \,||_C \, P)$, where B and C are sets of actions to be synchronised. However, in all the source TIPP files, the four processors are replaced by a single (equivalent) sequential process; the rationale behind such a simplification is exposed in [35, Sect. 4.3]: it reduces the size of the model's state space, but at the risk of potentially introducing errors if the modification is done manually. We found such a multiplicity of TIPP models annoying, making it harder to follow each experiment in detail. Instead, we opted for a unique model of the mainframe, with four processors composed in parallel.

```
// phase: 1=low, 2=high, 3=idle

const int l_init; // initial phase (1, 2, or 3) for loads

module ProgLoad   // programmer (low priority) jobs
   pl : [1..3] init l_init;
   [prog_job] pl = 1 -> lambda1 : (pl' = 1); // prog job arrival
   [prog_job] pl = 2 -> lambda2 : (pl' = 2); // prog job arrival
   [c] true -> (pl' = mod (pl, 3) + 1);      // phase change
endmodule

module UserLoad   // user (high priority) jobs
   ul : [1..3] init l_init
   [user_job] ul = 1 -> mu1 : (ul' = 1);     // user job arrival
   [user_job] ul = 2 -> mu2 : (ul' = 2);     // user job arrival
   [c] true -> (ul' = mod (ul, 3) + 1);      // phase change
endmodule

module FailLoad   // failures
   fl : [1..3] init l_init;
   [fail] fl = 1 -> delta1 : (fl' = 1);      // failure occurrence
   [fail] fl = 2 -> delta2 : (fl' = 2);      // failure occurrence
   [c] true -> (fl' = mod (fl, 3) + 1);      // phase change
endmodule

module LoadPhase  // rate for phase change of loads
   [c] true -> phi : true;
endmodule
```

Fig. 2. PRISM model fragment specifying the load modules

3 Modelling the Erlangen Mainframe in PRISM

The PRISM modelling language provides a consistent formalism for specifying the various different types of probabilistic models that are supported by the tool. The language is also accepted by many other probabilistic verification tools, notably Storm, which we also make use of in the present article, and has established itself as a common format for model exchange and benchmarking [50,31].

It is inspired by the Reactive Modules formalism of Alur and Henzinger [1], taking a slightly simplified version of this language and extending it with support for models with probabilistic behaviour. PRISM models comprise the parallel composition of multiple *modules*, which are able to perform (binary or multiway) synchronisation, thus satisfying the requirements outlined in Sect. 2.2.

The state of a module is described by a set of finite-ranging variables and its dynamics by *guarded commands* of the form $[a]\ g \to \lambda_1 : u_1 + \ldots \lambda_n : u_n$, which state that if guard g (a predicate over the global state of the model) is satisfied, then an a-labelled transition can occur in which module variables change according to one of the updates u_i. The values λ_i annotate each update u_i with the rate (for a CTMC, as here) or probability (for models with discrete probabilistic semantics) with which it occurs.

Fig. 2 shows a fragment of the PRISM model, namely, four simple modules that implement the *Loads* processes mentioned in Sect. 2.1. For instance, the module *UserLoad* describes the arrivals process for user jobs (i.e., the $UserLoad_i$ processes described in Sect. 2). Variable *ul* tracks the phase, which increases

periodically via a transition labelled with action c. This is specified by the third command, with the notation *ul'=expr* denoting that the value of *ul* will be updated to *expr* after the transition has occurred. The first two guarded commands, labelled with *user_job*, synchronise with another module representing the *user_queue*. The state of module *UserLoad* does not change when this transition occurs, but its state determines the rate attached to the transition (μ_1 or μ_2, depending on the phase).

In PRISM (as in TIPP), the combined rate of two synchronising transitions is taken to be the product of the individual rates. We often use the strategy, discussed earlier, of making one transition *passive*, with rate 1 (echoeing the neutral element $\mathbb{1}$), with the other transition specifying the rate. For example, in Fig. 2, each c transition in modules *ProgLoad*, *UserLoad*, and *FailLoad* is passive (an omitted rate is assumed to be 1) and a separate module *LoadPhase*, without any local state, provides the rate φ. This approach is adopted in this case to permit multiple modules (the three load-related components) to engage in multiway synchronisation (since their phases change simultaneously) with the rate specified separately, for convenience and clarity.

The full model, including the properties, is 250-line long (150 lines if comments and blank lines are excluded)[9]. It comprises 11 modules, which are composed using the PRISM's default parallel composition operator ||, under which modules synchronise on their common actions and can transition asynchronously on others. In doing so, we deviate slightly from the original TIPP model, which composes the four processors asynchronously, allowing them to have common actions such as *get_user_job*, that synchronise only with other modules and not with each other. This is achievable with the ||| operator within PRISM's "**system**...**endsystem**" construct but, since this is not supported by all tools, we instead use || and include four copies (one for each processor) of actions such as *get_user_job*.

4 Modelling the Erlangen Mainframe in LNT

The PRISM modelling language is not considered as a process algebra, although its parallel composition operators are those of TCSP [12] [41]. As the Erlangen mainframe was originally specified in the process algebra TIPP, it makes sense today to reformulate it using a recent process algebra such as LNT.

LNT [26] [57] [17] is a modern language for describing complex concurrent systems, which derives from the international standards LOTOS [43] and E-LOTOS [44], and therefore combines the best ideas from TCSP and CCS [55] [56]. It also draws inspiration both from functional programming languages and imperative languages, such as CSP [40], Occam [54] [42] [6], and Ada [2].

The following LNT constructs are used to describe the Erlangen mainframe: "**par** $E_1 ... E_m$ **in** $B_1 \,||...||\, B_n$ **end par**" specifies the parallel composition of n

[9] The model has been added to the PRISM benchmark suite [50], under the name "erlangen", and all files needed for the analysis done in this paper are available from `https://www.prismmodelchecker.org/files/erlangen/`.

behaviours $B_1, ..., B_n$, which should all synchronise on m events $E_1, ..., E_m$; "**alt** B_1 []...[] B_n **end alt**" specifies the nondeterministic choice between n behaviours B_1, ..., B_n; "**loop** B **end loop**" and "**loop** L **in** B **end loop**" specify the infinite repetition of behaviour B, the latter being possibly interrupted by a "**break** L" construct; "**if** V **then** B **end if**" and "**case** V **in** ... **end case**" specify conditionals; "**var** $X : T$ **in** B **end var**" declares a variable X of type T that is local to behaviour B; variables can be modified using assignments of the form "$X := V$"; "E" specifies the occurrence of event E (pure synchronisation); "$E(V)$" and "$E(?X)$" specify, respectively, the emission of value V on event E, and the reception of some value in variable X on event E; finally, "**process** $P[E_1 : C_1, ..., E_m : C_m](X_1 : T_1, ..., X_n : T_n)$ **is** B **end process**" declares a process P of body B with m event parameters $E_1, ..., E_m$ having channel types $C_1, ..., C_m$, and n variable parameters $X_1, ..., X_n$ having types $T_1, ..., T_n$.

LNT, like LOTOS, has no built-in notion of probabilities or rates and, thus, cannot express DTMCs or CTMCs directly. It is nevertheless possible to use LOTOS or LNT to specify and analyse Markovian systems [23] [18] [53]. So far, this has been mostly done in the theoretical framework of *Interactive Markov Chains* (IMCs) [33] and *Interactive Probabilistic Chains* (IPCs) [19], in which the transitions labelled with a rate or a probability are clearly separated from ordinary transitions. The Erlangen mainframe is radically different from IMCs and IPCs since all its transitions are compound (see Sect. 2.2).

In our LNT model, we translate each TIPP transition (a, λ) with $\lambda \neq \mathbb{1}$ to an LNT transition "$a(\lambda)$", where a is an LNT event and λ a rate, meaning that λ is a value emitted on a. Intuitively, λ could be simply a real number but, for convenience, we defined instead a RATE type, the values of which are either real numbers or symbolic names (β, δ_2, μ_2) of rate parameters defined in [37].

In our LNT model, we also translate each TIPP transition $(a, \mathbb{1})$ to an LNT transition "a (**any** RATE)", meaning that some rate value is received on a. The synchronisation rules of LNT, which are those of TCSP and LOTOS, ensure that the synchronisation of $a(\lambda)$ and a (**any** *RATE*) gives $a(\lambda)$, which is the same result as in TIPP when (a, λ) is synchronised with $(a, \mathbb{1})$. Notice that we do not translate $(a, \mathbb{1})$ to "$a(1.0)$", which would probably cause a deadlock according to the synchronisation rules of LNT.

The complete LNT model for the Erlangen mainframe is 250-line long (200 lines if comments and blank lines are excluded)[10]. For instance, Fig. 3 shows how the three load processes are described in LNT.

Although the encoding of rates in compound transitions is radically different from the IMC theory, we get here the same benefits as for the IMC approach: to describe and analyse stochastic systems, we can apply a "classical" process algebra (i.e., LNT or LOTOS), keeping its semantics unchanged and reusing its software tools without modification. Such a systematic translation to LNT of the original TIPP model is only correct because two conditions hold:

[10] All files can be found in the demo example "demo_15" of CADP, which is available from `https://cadp.inria.fr/demos.html`

```
process LOADS [C, FAIL, PROG_JOB, USER_JOB: DELAY] (DELTA1, DELTA2,
          LAMBDA1, LAMBDA2, MU1, MU2, PHI: RATE, INIT_PHASE: PHASE) is
   par C in
      LOAD [C, FAIL] (DELTA1, DELTA2, PHI, INIT_PHASE)
   ||
      LOAD [C, PROG_JOB] (LAMBDA1, LAMBDA2, PHI, INIT_PHASE)
   ||
      LOAD [C, USER_JOB] (MU1, MU2, PHI, INIT_PHASE)
   end par
end process

process LOAD [C, JOB: DELAY] (R1, R2, PHI: RATE, in var P: PHASE) is
   loop
      case P in
         1 ->    -- low-load phase
            loop PHASE1 in
               alt
                  JOB (R1)
               []
                  C (PHI);
                  break PHASE1
               end alt
            end loop;
            P := 2
      |  2 ->    -- high-load phase
            loop PHASE2 in
               alt
                  JOB (R2)
               []
                  C (PHI);
                  break PHASE2
               end alt
            end loop;
            P := 3
      |  3 ->    -- idle phase
            C (PHI);
            P := 1
      end case
   end loop
end process
```

Fig. 3. LNT model fragment specifying the load processes

(1) The mainframe model satisfies the "one-to-many" property stated in Sect. 2.2, meaning that each $a(\lambda)$ may only be synchronised with wildcard receptions "a (**any** RATE)". The LNT semantics does not support the product of rates, so that a synchronisation of $a(\lambda)$ and $a(\mu)$ does not give $a(\lambda\mu)$ as in TIPP, but either **stop** if $\lambda \neq \mu$ or $a(\lambda)$ if $\lambda = \mu$. Notice that Fig. 3 exploits the latter property, as the parallel composition of the nearly identical load processes involves a 3-party synchronisation between three transitions (c, φ), which, in LNT but not in TIPP, has the same effect as synchronising one transition (c, φ) with two transitions $(c, \mathbb{1})$.

(2) The LNT compilers of CADP represent the transitions going out of each state using a multiset, rather than a set: for instance, the LNT behaviour "**alt** $a(\lambda)$ [] $a(\lambda)$ **end alt**" will not generate a single transition "$a(\lambda)$" (i.e., will not factorise identical transitions as permitted by the operational semantics of most process algebras), but a choice between two "$a(\lambda)$" transitions

— which are later merged into a single transition "$a(2\lambda)$" when applying strong or branching stochastic bisimulation.

To reproduce the numerical results of the original papers (see Sect. 5 below), one needs to express properties about certain state variables. In this respect, LNT follows the principles of process algebras and labelled transition systems: information is attached to transitions, not to states, so that the contents of states cannot be observed. The usual solution is therefore to add self-loop transitions (called "probes") to the LNT model, the labels of these transitions exporting information contained in the states they are attached to.

In the mainframe model, probes need to be introduced only in the three queue processes. In the *fail_queue*, we insert a *z_avail* probe that is attached to each global state of the CTMC in which the local state of the *fail_queue* is F_0 ("working") rather than F_1 ("failed"). In the *prog_queue*, we insert a *z_prog_queue*(n) probe that is attached to each global state of the CTMC in which the *prog_queue* contains n jobs. In the *user_queue*, we insert a similar *z_user_queue*(n) probe.

5 Numerical Solutions

5.1 Objectives and Methodology

The original paper [37, Sect. 4.3] provides eight figures, numbered from 3 to 10; we surround these numbers by square boxes to distinguish them from figure numbers of the present article. Two figures (Fig. 7 and 8) can also be found in [35]. Figures 3, 4, 5, and 6 are about steady-state probabilities (i.e., in the long term, after an equilibrium has been reached). Figures 7, 8, 9, and 10 are about transient probabilities (i.e., at specific time instants).

Our challenge, as we defined it, was to reproduce these eight figures by applying the CADP, PRISM, and Storm tools to our models of the mainframe.

Obtaining the same values as in the original paper was not an easy task, as several difficulties arose. As could be seen from the source TIPP files, the authors did many experiments, with different TIPP models (featuring different processor models), different queue sizes, and different rate parameters.

Such a variability has undesirable consequences: these experiments are difficult to follow because of their multiple experimental settings, and they are difficult to reproduce, as information is incomplete in places, with useful details and parameter values missing from the original paper.

In particular, numerical results do not always match across figures because of untold changes in models or parameters. For instance, Fig. 9 of [37], which assumes a couple of fixed values for the parameters β and δ_2, displays a steady-state availability result $A(\infty) = 0.981$, this value being independent from the phase in which the load processes are started. But in Fig. 5 of [37], which assumes that the load processes are started in phase 1 and explores various values of β and δ_2, none of the steady-state availability results displayed for $A(\infty)$ matches the value 0.981 of Fig. 9. Looking into the TIPP files, we presume

that Fig. 9 was obtained by using a simpler processor model and different queue sizes (4, 4) than Fig. 5.

In a first attempt, we fought to reproduce exactly the same values as in the original paper by redoing the same experiments with various models, various queue sizes, and various sets of rate parameters. Using PRISM, we managed to reproduce the original results for Fig. 5–8 with good precision (from 3 to 6 decimal places).

We then considered that an exact reproduction of the original experiments was perhaps not the most suitable goal. Rather than mere imitation, we felt it would be better to recreate a simplified experimental setting that would be easier to understand, so that the mainframe example would get, hopefully, more chances to be studied by others and reused for various purposes, e.g., as a benchmark for performance analysis tools or as a lab exercise in university classrooms.

We therefore adopt the following simplifying assumptions, which will be detailed and justified below:

- We use a unique model of the mainframe, which is parameterized by the maximal sizes of the *prog_queue* and *user_queue*, and by the initial phase (1, 2, or 3) in which the three load processes are stated.
- Concerning the maximal queue sizes, the original paper [37, Sect. 3.4] observed that reducing them from (40, 10) to (10, 4) does not sacrifice model accuracy very much. We further reduce these sizes to (4, 4) by default, since larger sizes do not change the shape of figures (as justified below in Sect. 5.6), but increase the number of states of generated CTMCs and the execution time for computing steady-state and transient probabilities.
- Concerning the initial phase, we start all load processes in phase 1 (low load) by default. We verified that, despite the sophisticated behaviour of the three load processes (with three phases and different rates λ_1, λ_2, μ_1, μ_2, δ_1, δ_2, and φ), the long-run probability of being in each phase is identical (i.e., 1/3) for each of the three phases. This suggests that, when the three load processes are connected to the three queues, their sophisticated behaviour becomes more regular, just as a torrent loses its capricious flow when it pours into a reservoir lake with a barrage.
- Concerning the rate parameters, most of them have constant values, meaning that the impact of their modification is not studied in the experiments for producing Fig. 3–10. We assign the following rate parameters the same default values as in the original paper [37]:

$$\alpha = 48 \quad \beta = 0.01 \quad \delta_1 = 0.00035 \quad \delta_2 = 0.0007 \quad \lambda_1 = 0.01667$$
$$\lambda_2 = 0.16 \quad \mu_1 = 0.033 \quad \mu_2 = 2 \quad \nu = 12 \quad \varphi = 0.00334 \quad \xi = 0.3$$

  All these values are expressed in the same unit: min^{-1}. Only three of these parameters (β, δ_2, and μ_2) vary in the experiments.

Under these assumptions, and a few others to be stated hereafter, we obtain numerical results that are close to those of the original papers and, noticeably, preserve the shape of the curves of Fig. 3–10. We now detail how to reproduce each of these eight figures in turn.

5.2 Reproduction of Figures 3 and 4 (Steady-State Analysis)

These two figures, which appear only in the first original paper [37], answer a dimensioning question: they explore how the size of queues impacts the probability that the mainframe system is blocked, waiting for new requests to be processed. Fig. 3 concerns the *prog_queue* (denoted Q in the original papers) for low-priority jobs. Fig. 4 concerns the *user_queue* (denoted R in the original papers) for high-priority jobs. Both figures vary two parameters: the queue size and the arrival rate μ_2 of user jobs. Fig. 4 displays the figures generated using CADP — those generated using PRISM being similar.

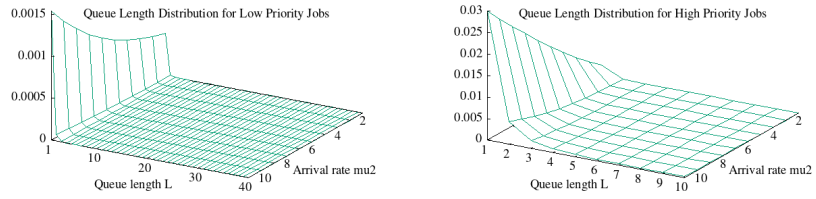

Fig. 4. Figures 3 and 4 generated using CADP

These two figures have been the most difficult ones to reproduce, because of ambiguities and lack of information. We briefly mention the issues we faced and the solutions we adopted:

(a) The z-axis of Fig. 3 and 4 are labelled P(Q = l | load = "high") and P(R = l | load = "low"), respectively. Here, "l" does not denote the digit one, but the queue length indicated on the x-axis of both figures. To avoid any confusion, we hereafter denote queue length with a capital "L".

(b) In the original paper, the x-axes of Fig. 3 and 4 start at $L = 0$ and the shape of the curves gives the impression that there is a value on the z-axis when the queue length is zero — this value being the probability 0.002 for Fig. 3 and 0.051 for Fig. 4. This is not the case actually: the probabilities when $L = 0$ are very high (close to one) and would make the plots unreadable if displayed. To avoid such a misleading impression, the x-axes of our Fig. 3 and 4 explicitly start at $L = 1$ rather than $L = 0$.

(c) In Fig. 3, the x-axis displays the queue length L ranging from 1 to 40. This allows two possible interpretations: either (i) the mainframe system has a *prog_queue* Q of size 40, and the z-axis of Fig. 3 represents the distribution over L (and μ_2), i.e., the probability that the *prog_queue* has L elements, for all possible values of L but zero; or (ii) each value of L represents a different

instance of the mainframe system in which the *prog_queue* has size L, and the z-axis represents the probability that the queue is full, i.e., contains L elements. Both interpretations are also plausible for Fig. 4, in which the size L of the *user_queue* R ranges from 1 to 10. We opted for the first interpretation, which seems compatible with the legends of Fig. 3 and 4, namely "Queue Length Distribution for Low Priority Jobs" and "Queue Length Distribution for High Priority Jobs". We implemented both interpretations in LNT and compared the results obtained using CADP: the probabilities are actually different, but the differences are so small that the shapes of Fig. 3 and 4 do not change.

(d) The z-axis labels of Fig. 3 and 4, namely, P(Q = L | load = "high") and P(R = L | load = "low") suggest that they express conditional probabilities, although the notion of conditional probability is not mentioned anywhere in the original papers. We first tried to interpret z-axis values this way by computing them according to the definition of conditional probabilities, i.e.:

$$\mathrm{P}(\mathrm{Q} = \mathrm{L} \mid \text{load} = \text{“high”}) = \frac{\mathrm{P}(\mathrm{Q} = \mathrm{L} \cap \text{load} = \text{“high”})}{\mathrm{P}(\text{load} = \text{“high”})}$$

and:

$$\mathrm{P}(\mathrm{R} = \mathrm{L} \mid \text{load} = \text{“low”}) = \frac{\mathrm{P}(\mathrm{R} = \mathrm{L} \cap \text{load} = \text{“low”})}{\mathrm{P}(\text{load} = \text{“low”})}$$

As mentioned already, we observed, using PRISM and CADP, that the steady-state probabilities P(load = "high") and P(load = "low") are both equal to 1/3. To compute the probabilities of both intersections, additional probes were introduced in the LNT specification to indicate when the phase of the load processes is high or low; these probes were then synchronised with the probes expressing the number of items in each queue, and their steady-state throughput was computed using CADP. Doing so, we obtained curves that were similar to those of the original paper, but with a visible difference in the z-axis values when $L = 1$. We thus decided to forget about conditional probabilities and to simply compute P(Q = L) and P(R = L) for the z axes; despite this simplification, we still obtain plausible curves.

Finally, we produced Fig. 3 and 4 (see Fig. 4) that closely resemble those of the original paper. Yet, the maximal values on the z-axis (when $L = 1$ and $\mu_2 = 10$) are not the same: in [37], these probabilities are approximately 0.002 for Fig. 3 and 0.051 for Fig. 4; on our figures, they are 0.0015 for Fig. 3 and 0.03 for Fig. 4. This probably arises from differences in mainframe models and rate parameters used for the various experiments.

5.3 Reproduction of Figures 5 and 6 (Steady-State Analysis)

These two figures describe the impact of failures and repairs on the processing of high-priority jobs. To do so, these figures vary the failure arrival rate δ_2 and the repair rate β and display, for each pair (δ_2, β), an availability value,

denoted $A(\infty)$, for Fig. 5 and a throughput value, denoted $M(\infty)$, for Fig. 6. Fig. 5 displays the figures generated using CADP, which are identical to those generated using PRISM, as well as those given in the original paper [37].

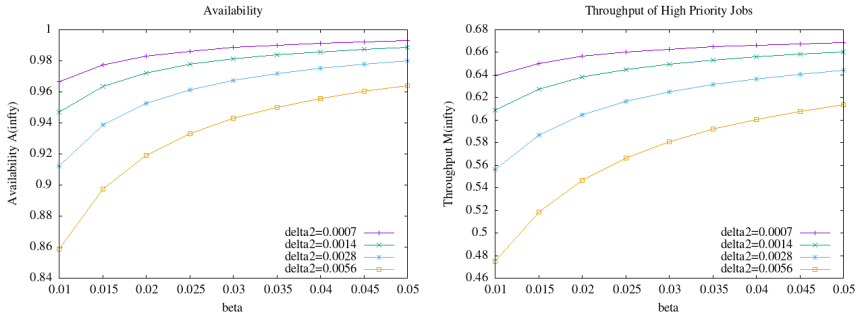

Fig. 5. Figures 5 and 6 generated using CADP

In Fig. 5, $A(\infty)$ is the steady-state limit of the *point availability* $A(t)$, which is defined [37, rule (2) of Sect. 3.2] as the probability that the mainframe is operational at time t; the intended meaning of "operational" here is that the failure queue process, which has two states F_0 and F_1, is in state F_0 where no failure is taking place. A different definition of "operational" (e.g., all processors are serving jobs) might produce different numerical results.

In Fig. 6, $M(\infty)$ is the steady-state limit of the *point throughput* $M(t)$ of the action *get_user_job* that occurs every time a high-priority job is submitted to the processors. $M(t)$ can be defined, according to [37, rules (9) and (10) of Sect. 3.2], as $M(t) = \sum_{i \in 1,\ldots,n} r(s_i) P(X(t) = s_i)$, where s_i iterates over all reachable states of the CTMC, and where $r(s_i) = \sum_j \rho_j$ for all (get_user_job, ρ_j) transitions going out of state s_i.

5.4 Reproduction of Figures 7 and 8 (Transient Analysis)

These two figures bear similarity with Fig. 5 and 6 but display transient values for, respectively, point availability $A(t)$ and point throughput $M(t)$ instead of displaying their steady-state limits $A(\infty)$ and $M(\infty)$. Also, Fig. 7 and 8 assume a constant value (namely, 0.0007) for the parameter δ_2 and, rather than varying δ_2, consider 15 different time instants.

Fig. 6 shows the figures generated using CADP and PRISM. These figures are pairwise identical, and also identical to those given in the original papers [37] and [35].

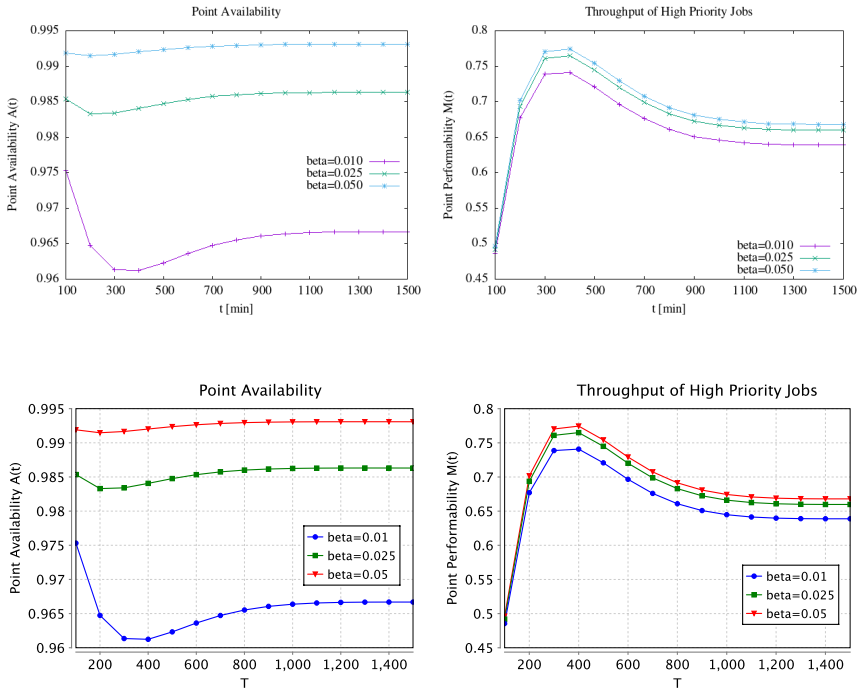

Fig. 6. Figures 7 and 8 generated using CADP (top) and PRISM (bottom)

5.5 Reproduction of Figures 9 and 10 (Transient Analysis)

These two figures are similar to Fig. 7 and 8, with the difference that they give the parameter β a constant value (namely, 0.01) and vary the *phase*, i.e., the initial state ("low", "load", or "idle") of the three load processes — these processes being all started in the same phase and modifying their phases simultaneously by means of a 3-party synchronisation on action c.

Fig. 7 shows the Fig. 9 and 10 generated using CADP. We felt that the top label of Fig. 9 in [37] ("Point Availability During High Load Phase") was misleading, as it suggests the use of conditional probabilities that are never evoked elsewhere in the original papers; we therefore changed this label to "Point Availability", while a longer, completely accurate label would be "Point Availability upon Initialisation in High Load".

The curves of Fig. 9 obtained using CADP and PRISM clearly have the same shape as those of the original paper. However, the values are slightly different. For instance, the steady-state limit $A(\infty)$ is 0.967 for CADP and PRISM, whereas it was 0.981 in [37]. We already discussed this issue in Sect 5.1 and believe that our Fig. 9 is coherent, as its value 0.967 matches the steady-state value of our Fig. 6 for $\beta = 0.01$ and $\delta_2 = 0.0007$ (see Fig. 5), which was not the case in the

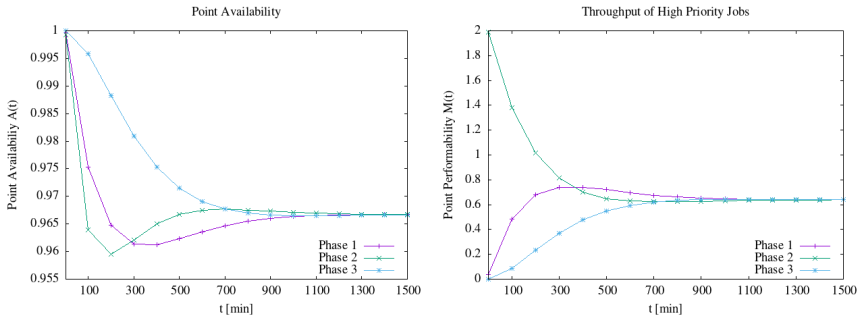

Fig. 7. Figures 9 and 10 generated using CADP

original paper. Interestingly, the value of β has a key influence on the shape of Fig. 9 (see Fig. 8 for a comparative view). Finally, our Fig. 10 is identical, in shape and values, to the one of [37].

5.6 Impact of Queue Sizes

The original paper [37] claims that reducing queue sizes from (40, 10) to (10, 4) has little impact on numerical results; we went further in this direction by reducing queue sizes to (4, 4). In this section, we present additional experiments that support these claims.

Table 2 summarises the experiments done with CADP for various sizes of the *prog_queue* and *user_queue*. For each experiment, the 2nd column of the table gives the number of states of the corresponding CTMC, after minimisation modulo stochastic strong bisimulation. The 3rd, 4th, and 5th columns quantify the loss of precision in the set of throughput values generated for Fig. 5–6, Fig. 7–8, and Fig. 9–10, respectively; the loss of precision for line (m, n) and column C is defined as the largest value, for $i \in \{1, ..., n\}$, of $2\,|x_i - y_i|/(x_i + y_i)$, where $\{x_1, ..., x_n\}$ is the set of throughput values generated for queue sizes $(40, 10)$ and the figures of column C, and $\{y_1, ..., y_n\}$ the set of throughput values generated for queue sizes (m, n) and the figures of column C.
From this table, we draw three conclusions:

- The numbers of CTMC states for queue sizes (40, 10) and (10, 4) are exactly those mentioned in [37, Sect. 4.3]. The fact that we obtain the same numbers is a clear indication that our LNT and PRISM models are compatible with those of the original paper.
- The losses of precision for queue sizes (10, 4) and (4, 4) are equal. Actually, all the throughput values for (10, 4) and (4, 4) are pairwise identical, except in two cases where a difference occurs at the 6th decimal position (0.63996 vs 0.639961). This retrospectively justifies our decision of reducing queue sizes to (4, 4).

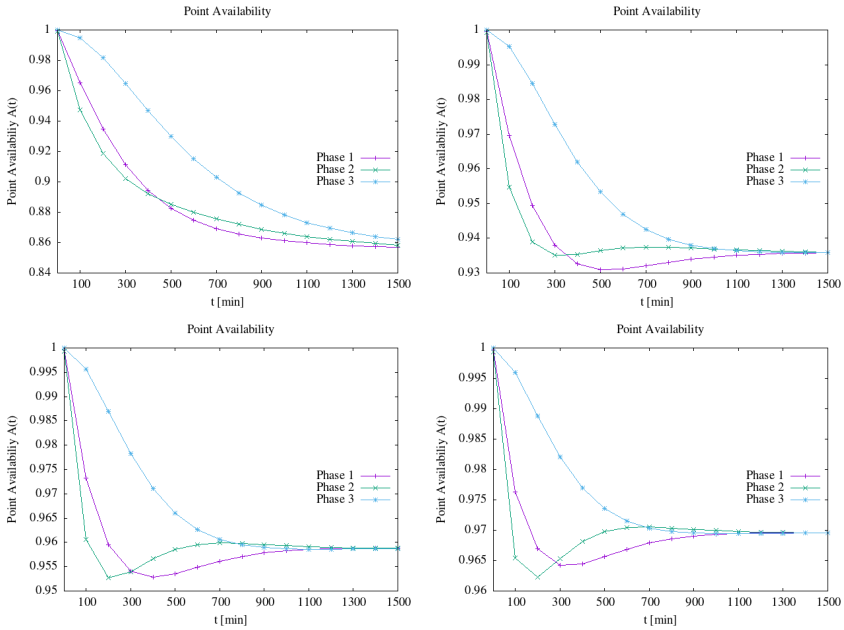

Fig. 8. Variants of Fig. 9 for $\beta = 0.002$, 0.005, 0.008, and 0.011

– Globally, the sizes of queues have negligible impact (≈1.5%) on throughput values. This appears as a specific consequence of the chosen rate parameters and particular experiments for producing Fig. 5–10.

5.7 Using PRISM and Storm

We analysed the mainframe model using both the PRISM and Storm toolsets. There are differences in the two tools' overall functionality and focus (e.g., support for Markov automata only in Storm, and for stochastic games only in PRISM/PRISM-games), but there is a large common core of probabilistic model checking functionality, including what is needed for this exercise. The inputs to the two tools, in terms of model and property specifications, are identical for our purposes, since Storm accepts the PRISM modelling language (for which we activate the "PRISM-compatibility mode" of Storm) and the tools largely agree on the syntax for temporal logic queries.

Temporal logic. For property specifications, we use the temporal logic CSL (continuous stochastic logic) [3,4], which is a branching-time temporal logic for characterising transient and steady-state aspects of CTMCs. In the years since the mainframe example was first developed, CSL has established itself as widely used means to formally specify performance and reliability criteria of computer

queue size	CTMC states	Fig. 5–6	Fig. 7–8	Fig. 9–10
(40, 10)	21,648	—	—	—
(30, 8)	13,392	0.0031%	0.0032%	0.0032%
(20, 5)	6048	0.0186%	0.0187%	0.0196%
(10, 4)	2640	0.0312%	0.0314%	0.0326%
(4, 4)	1200	0.0312%	0.0314%	0.0326%
(3, 3)	768	0.0527%	0.0530%	0.0551%
(2, 2)	432	0.1121%	0.1126%	0.1167%
(1, 1)	192	1.4955%	1.4988%	1.5262%

Table 2. Impact of queue sizes

and communication systems, bridging the fields of performance analysis and formal verification [5].

For the mainframe example, we use the following formulae:

(i) $\mathbf{S}_{=?}[\, ujq{=}l \,]$: long-run probability of queue length being l (Fig. 3 and 4)
(ii) $\mathbf{S}_{=?}[\, fq{=}0 \,]$: long-run availability (Fig. 5)
(iii) $\mathbf{R}^{thru_hi}_{=?}[\, \mathbf{S} \,]$: long-run throughput of high-priority jobs (Fig. 6)
(iv) $\mathbf{P}_{=?}[\, \mathbf{F}^{=T} fq{=}0 \,]$: point availability (Fig. 7 and 9)
(v) $\mathbf{R}^{thru_hi}_{=?}[\, \mathbf{I}^{=T} \,]$: point throughput of high-priority jobs at time T (Fig. 8)

The formula $\mathbf{S}_{=?}[\, \varphi \,]$ asks for the long-run (steady-state) probability that predicate φ is true. Here, we write φ in terms of the variables that make up the PRISM model: *ujq* is the current size of the *user_queue* and *fq* equals 1 if the mainframe has failed, 0 if not. Formula $\mathbf{P}_{=?}[\, \mathbf{F}^{=T} \varphi \,]$ is the transient equivalent, asking for the probability of φ being true at time instant T.

Formulae $\mathbf{R}^{r}_{=?}[\, \mathbf{S} \,]$ and $\mathbf{R}^{r}_{=?}[\, \mathbf{I}^{=T} \,]$ ask for the expected value of a reward r in the long-run and at time instant T, respectively. Here, *thru_hi* is a reward structure that, in any state, equals the total rate of outgoing (*user_job_ready*) transitions that correspond to a user job being processed. We note that the current exercise illustrates some of the benefits of temporal logic, conveying easily and precisely the properties being used, in a form directly accessible to multiple tools.

Solution methods and performance. CSL model checking [4] of the formulae above essentially reduces to either *steady-state* or *transient* solution of the CTMC. Steady-state analysis is done in [37] using iterative numerical techniques, in particular the Gauss-Seidel method. PRISM takes exactly the same approach. Storm's default technique here is also iterative: the generalised minimal residual method (GMRES) method, with ILU preconditioning.

Transient analysis in [37] is performed using the refined randomisation technique [52], which is very similar to the method known as *uniformisation*, proposed for CSL model checking in [4] and used by most probabilistic model checkers. In this sense, in contrast to the situation for other classes of probabilistic

models, the default techniques used to solve core CTMC queries by modern probabilistic model checkers have not changed significantly.

Performance, however, has of course improved. While [37] reports numerical solution of the CTMC for queue sizes $(40, 10)$ taking about 165 seconds, PRISM and Storm build and solve the model in 2-3 seconds. Whilst the numerical solution part, which uses similar techniques to [37], is largely benefitting from hardware advances, other aspects of the process have changed. Techniques for model exploration and construction, for example, have improved. For the largest mainframe instance we considered here ($\sim$1.6 million states), both PRISM and Storm build the model in just a few seconds, the former with symbolic (binary decision diagram based) methods, the latter using explicit-state techniques.

As has been investigated at length [16], comparing the runtime of probabilistic verification tools needs to be done with considerable care when they use different solution methods offering different guarantees on the precision of their results. In this particular case, the methods and their configuration are very similar (for example, both tools by default terminate iterative numerical methods when they have converged to within a maximum relative difference of 10^{-6}). Whilst [16] showed Storm to generally perform better than other comparable tools across a broad range of benchmarks and model types, on these CTMCs, PRISM, and Storm exhibit very similar runtimes, e.g., both taking $\sim$34 seconds to solve the (steady-state) property (ii) above on a CTMC with 922,746 states.

Queue size	CTMC states		Transient (T=1000) time (s)			Steady-state time (s)		
	Full	Reduced	No bisim.	Bisim.	Change	No bisim	Bisim	Change
40,10	110,946	21,648	93.3	**19.0**	-80%	2.2	**1.1**	-50%
80,20	418,446	81,648	331.9	**78.7**	-76%	13.2	**11.0**	-17%
120,30	922,746	180,048	1,328.3	**492.5**	-63%	**33.8**	37.7	12%
160,40	1,623,846	316,848	1,949.1	**505.0**	-74%	**71.8**	125.2	74%

Table 3. Computation times (Storm) for transient and steady-state properties with and without bisimulation minimisation.

Bisimulation. Since strong bisimulation (lumping) for CTMCs [13,38,48] is known to preserve satisfaction of the logic CSL, a key result shown in [4], we also investigate applying bisimulation minimisation before model solution. Once such a minimisation has been applied (to the PRISM model described in Sect. 3), both PRISM and Storm produce models of exactly the same size as generated by CADP and as in [37]. Lumping is mentioned in [37] but does not seem to be applied. In fact, investigation shows that the reduction in size produced by minimising the PRISM model of Sect. 3 is solely due to redundancy in the representation of the state of the processors, and manually exploiting this at the modelling language level (by combining the processors into a single abstract process, already discussed at the end of Sect. 2.3) yields precisely the same

reduced models. This suggest that applying bisimulation to the models of [37] may have yielded no further reductions anyway.

It is known, from practical investigations in [47] that minimisation can, but does not always, result in an overall reduction in computation time, depending on whether the benefits of solving a smaller, minimised CTMC outweigh the cost of performing minimisation. We measured times for representative CSL queries – (iv) and (ii) above – on CTMCs of various sizes, created by varying queue sizes. Although Section 5.6 showed that this has limited effect on the numerical results obtained, this provides a convenient way to examine the impact of the state space size on performance. To see this impact more clearly, we use larger models than in the earlier sections. Table 3 shows models sizes and times to run Storm (which has a larger range of minimisation options built in; here we use the default settings) with and without bisimulation. The faster of the two times are indicated in bold face.

We see that all models are reduced by the same amount: a factor of 5.25. Regarding computation time, we observe that minimisation yields significant gains (a factor 4 speed-up) for the transient property. On the other hand, for the steady-state property, the gain decreases with model size and using minimisation eventually becomes slower. This seems to be caused by an increase in the number of iterations required for numerical solution of the minimised model to converge, despite its smaller size (both in terms of the number of states and the average number of transitions for each state).

5.8 Using CADP

Unlike PRISM and Storm, which are integrated tools, CADP is rather a tool set, i.e., a collection of about fifty different tools that can be combined to achieve very diverse tasks. Only a small fraction of these tools is dedicated to the analysis of probabilistic and stochastic systems. CADP does not support a stochastic temporal logic such as CSL, but provides various tools that compute strong and branching stochastic bisimulations, steady-state probabilities, and transient probabilities [36] [18].

The CADP tools can be invoked directly from the command line or, in the case of involved scenarios, from verification scripts written in SVL language[11] [24] [51]. SVL eases the combination of the various CADP tools and plays a crucial role for the reproducibility of experiments. It provides high-level CADP-specific language constructs, which can be freely combined with POSIX shell commands. The latter, which are preceded by the escape symbol '%', provide for variable assignments and substitutions, "**if-then-else**" conditionals, "**for**" loops, parameterized procedures, etc.

The generation of Fig. 3–10 for the Erlangen mainframe is fully described by a 350-line SVL script (200 lines if not counting comments and blank lines). This script can either generate all figures, or only a specified subset of them (namely, 3, 4, 5–6, 7–8, or 9–10). It contains three procedures:

[11] `https://cadp.inria.fr/man/svl-lang.html`

– Procedure *INITIALIZE* sets rate parameters, queue sizes, and initial phase to their default values.
– Procedure *GENERATE* takes the LNT description of the mainframe, replaces most rate parameters by their default values but keeps a few parameters in symbolic form (i.e., β, δ_2, and/or μ_2) that have to iterate over multiple values, replaces the queue sizes and initial phase by specified values, and generates a finite, explicit-state CTMC stored in a file (encoded in the BCG format of CADP). This CTMC contains compound action-rate labels on its transitions — some of these rates being still symbolic. Then, in this CTMC, every self-loop transition corresponding to a probe is either deleted if that probe is not useful for the particular figure to be generated, or is given the rate 1.0 otherwise (so that the throughput of the probe transition is equal to the probability of being in the state the probe is attached to).
– Procedure *INSTANTIATE* takes a CTMC with partially symbolic rates, replaces these rates by specified values, and minimises the CTMC modulo stochastic strong bisimulation using the BCG_MIN[12] tool of CADP. The resulting minimised CTMC is aperiodic, irreducible, has no deadlock state and no τ-transitions.

In the SVL script, the various code fragments for generating each figure are similar. First, they invoke the *INITIALIZE* procedure and, possibly, the *GENERATE* procedure. Then, they perform one loop or two nested loops to vary one or two rate parameters (β, δ_2, or μ_2) or the initial phase. The body of the innermost loop successively invokes the *GENERATE* procedure (if it has been called already before the loop), the *INSTANTIATE* procedure, and either the BCG_STEADY[13] tool or the BCG_TRANSIENT[14] tool of CADP for computing steady-state or transient probabilities.

Each invocation of the two latter tools appends to a text file a new line containing the transition throughputs (i.e., sum of transition rates multiplied by incoming state probabilities) of selected actions of the CTMC. For instance, the point availability values $A(t)$ and $A(\infty)$ defined in Sect. 5.2 are computed, using BCG_STEADY and BCG_TRANSIENT, as the throughput of self-loop transitions corresponding to the *z_avail* probe. Similarly, the point throughput values $M(t)$ and $M(\infty)$ of Sect. 5.2 are computed as the throughput of *get_user_job* transitions. Lastly, eight small Gnuplot scripts convert these throughput files to eight figures in PNG format.

The complete execution of the SVL script takes about 10 minutes on a standard laptop (Intel x64 processor, 16 GB RAM) running Linux.

Finally, we checked, using the BCG_CMP[15] tool of CADP, that, for the eight different queue sizes of Table 2, the CTMCs generated by PRISM are pairwise equivalent, modulo stochastic strong bisimulation, to the CTMCs (without probe transitions) generated by CADP.

[12] `https://cadp.inria.fr/man/bcg_min.html`
[13] `https://cadp.inria.fr/man/bcg_steady.html`
[14] `https://cadp.inria.fr/man/bcg_transient.html`
[15] `https://cadp.inria.fr/man/bcg_cmp.html`

6 Conclusion

The present work is an essay in research reproducibility. We revisited the Erlangen mainframe case study, a challenging example proposed thirty years ago [37] [35]. We improved this case study in two ways:

- *Corrections*: The original papers contained a few mistakes, which we found and fixed. Also, the explanations given in these papers were incomplete or ambiguous in places, but our collaborative thinking managed to recover the missing parts and provide the most plausible interpretations. When appropriate, we dug into the source TIPP files to get confirmation of our decisions.
- *Simplifications*: Some experiments for producing the eight figures displayed in [37] were unnecessarily complex. We addressed this issue by proposing a few simplifications that, while producing nearly identical results as in the original papers, are easier to understand and execute faster.

We developed two novel formal models for the Erlangen mainframe, one in the automata-based language PRISM, and another one in the process-algebra-based language LNT. The tools PRISM, Storm, and CADP performed all numerical experiments and successfully reproduced the same figures as in the original papers. This suggests that these tools could be used in combination to analyse other systems, especially critical ones, as the fact that different tools developed independently give identical results on the same model is a convincing argument for safety/security certification agencies.

As regards future developments, the present work could be reused and extended in several directions:

- The Erlangen mainframe is a suitable basis for lab exercises in university courses. Our paper gives hints on how using PRISM, Storm, and CADP on this case study in order to model the system, express its properties, and obtain performance numbers. Many more experiments could be proposed by varying other parameters than β, δ_2, and μ_2.
- The Erlangen mainframe could be specified using other languages and analysed using other tools, provided that the language requirements of Sect. 2.2 are satisfied. This example could also be used as a scalable model for software competitions since, by varying the size of queues, one easily obtains CTMCs of increasing complexity.
- Our modelling of the Erlangen mainframe relies on compound transitions that combine actions and rates. It would be interesting to investigate whether this problem can also be expressed in alternative formalisms without compound transitions, such as Interactive Markov Chains [33], in which transitions carry either an action or a rate.

Acknowledgements

We are grateful to the anonymous reviewers for their constructive remarks and to Nazareno Garagiola (Saarland University) for early experiments with Storm on the mainframe specifications.

References

1. Alur, R., Henzinger, T.A.: Reactive Modules. Formal Methods in System Design **15**(1), 7–48 (1999). https://doi.org/10.1023/A:1008739929481
2. ANSI: Ada Programming Language. Military Standard ANSI-MIL-STD-1815A, American National Standards Institute, New Year, USA (Jan 1983)
3. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.K.: Verifying Continuous Time Markov Chains. In: Alur, R., Henzinger, T.A. (eds.) Proceedings of the 8th International Conference on Computer Aided Verification (CAV'96), New Brunswick, NJ, USA. Lecture Notes in Computer Science, vol. 1102, pp. 269–276. Springer (Jul 1996). https://doi.org/10.1007/3-540-61474-5_75
4. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.: Model-Checking Algorithms for Continuous-Time Markov Chains. IEEE Transactions on Software Engineering **29**(6), 524–541 (2003). https://doi.org/10.1109/TSE.2003.1205180
5. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.: Performance Evaluation and Model Checking Join Forces. Communications of the ACM **53**(9), 76–85 (2010). https://doi.org/10.1145/1810891.1810912
6. Barrett, G.: OCCAM 3 Reference Manual (Mar 1992), iNMOS Limited, Draft
7. Bernardo, M., Cleaveland, R., Sims, S., Stewart, W.: TwoTowers: A Tool Integrating Functional and Performance Analysis of Concurrent Systems. In: Budkowski, S., Cavalli, A.R., Najm, E. (eds.) Proceedings of the IFIP TC6/WG6.1 Joint 11th International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols and 18th International Workshop on Protocol Specification, Testing and Verification (FORTE/PSTV'98), Paris, France. IFIP Conference Proceedings, vol. 135, pp. 457–467. Kluwer (Nov 1998)
8. Bohnenkamp, H., Pedro R. d'Argenio, Hermanns, H., Katoen, J.P.: MoDeST: A Compositional Modeling Formalism for Hard and Softly Timed Systems. IEEE Transactions on Software Engineering **32**(10), 812–830 (2006). https://doi.org/10.1109/TSE.2006.104
9. Bozzano, M., Cimatti, A., Katoen, J., Nguyen, V.Y., Noll, T., Roveri, M.: Safety, Dependability and Performance Analysis of Extended AADL Models. The Computer Journal **54**(5), 754–775 (2011). https://doi.org/10.1093/COMJNL/BXQ024
10. Bravetti, M., Bernardo, M.: Compositional Asymmetric Cooperations for Process Algebras with Probabilities, Priorities, and Time. In: Corradini, F., Inverardi, P. (eds.) Proceedings of the International Workshop on Models for Time-Critical Systems (MTCS'00), State College, PA, USA. Electronic Notes in Theoretical Computer Science, vol. 39, pp. 197–230. Elsevier (Aug 2000). https://doi.org/10.1016/S1571-0661(05)80749-2
11. Brinksma, E., Hermanns, H.: Process Algebra and Markov Chains. In: Brinksma, E., Hermanns, H., Katoen, J. (eds.) Revised Lectures on Formal Methods and Performance Analysis, First EEF/Euro Summer School on Trends in Computer Science, Berg en Dal, The Netherlands. Lecture Notes in Computer Science, vol. 2090, pp. 183–231. Springer (Jul 2000). https://doi.org/10.1007/3-540-44667-2_5
12. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A Theory of Communicating Sequential Processes. J. ACM **31**(3), 560–599 (Jul 1984). https://doi.org/10.1145/828.833
13. Buchholz, P.: Exact and Ordinary Lumpability in Finite Markov Chains. Journal of Applied Probability **31**(1), 59–75 (Mar 1994). https://doi.org/10.2307/3215235
14. Buchholz, P.: Markovian Process Algebra: Composition and Equivalence. In: Herzog, U., Rettelbach, M. (eds.) Proceedings of the 2nd Workshop on Process Al-

gebras and Performance Modelling (PAPM'94), Regensberg/Erlangen, Germany. pp. 11–30 (Jul 1994)
15. Buchholz, P., Kemper, P.: Quantifying the Dynamic Behavior of Process Algebras. In: de Alfaro, L., Gilmore, S. (eds.) Proceedings of the Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM-PROBMIV'01), Aachen, Germany. Lecture Notes in Computer Science, vol. 2165, pp. 184–199. Springer (Sep 2001). https://doi.org/10.1007/3-540-44804-7_12
16. Budde, C.E., Hartmanns, A., Klauck, M., Křetínský, J., Parker, D., Quatmann, T., Turrini, A., Zhang, Z.: On Correctness, Precision, and Performance in Quantitative Verification: QComp 2020 Competition Report. In: Proceedings of the 9th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'20), Rhodes, Greece. Lecture Notes in Computer Science, vol. 12479, pp. 216–241. Springer (Oct 2020). https://doi.org/10.1007/978-3-030-83723-5_15
17. Champelovier, D., Clerc, X., Garavel, H., Guerte, Y., McKinty, C., Powazny, V., Lang, F., Serwe, W., Smeding, G.: Reference Manual of the LNT to LOTOS Translator (Version 7.3) (May 2024), https://cadp.inria.fr/publications/Champelovier-Clerc-Garavel-et-al-10.html, INRIA, Grenoble, France
18. Coste, N., Garavel, H., Hermanns, H., Lang, F., Mateescu, R., Serwe, W.: Ten Years of Performance Evaluation for Concurrent Systems Using CADP. In: Margaria, T., Steffen, B. (eds.) Proceedings of the 4th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation ISoLA 2010 (Amirandes, Heraclion, Crete), Part II. Lecture Notes in Computer Science, vol. 6416, pp. 128–142. Springer (Oct 2010). https://doi.org/10.1007/978-3-642-16561-0_18
19. Coste, N., Hermanns, H., Lantreibecq, E., Serwe, W.: Towards Performance Prediction of Compositional Models in Industrial GALS Designs. In: Bouajjani, A., Maler, O. (eds.) Proceedings of the 21th International Conference on Computer Aided Verification (CAV'09), Grenoble, France. Lecture Notes in Computer Science, vol. 5643, pp. 204–218. Springer (Jul 2009). https://doi.org/10.1007/978-3-642-02658-4_18
20. D'Argenio, P.R., Katoen, J.: A Theory of Stochastic Systems, Part I: Stochastic Automata. Information and Computation **203**(1), 1–38 (2005). https://doi.org/10.1016/J.IC.2005.07.001
21. Esteve, M., Katoen, J., Nguyen, V.Y., Postma, B., Yushtein, Y.: Formal Correctness, Safety, Dependability, and Performance Analysis of a Satellite. In: Glinz, M., Murphy, G.C., Pezzè, M. (eds.) Proceedings of the 34th International Conference on Software Engineering, (ICSE'12), Zurich, Switzerland. pp. 1022–1031. IEEE Computer Society (Jun 2012). https://doi.org/10.1109/ICSE.2012.6227118
22. Fischer, W., Meier-Hellstern, K.S.: The Markov-Modulated Poisson Process (MMPP) Cookbook. Performance Evaluation **18**(2), 149–171 (Sep 1993). https://doi.org/10.1016/0166-5316(93)90035-S
23. Garavel, H., Hermanns, H.: On Combining Functional Verification and Performance Evaluation using CADP. In: Eriksson, L.H., Lindsay, P.A. (eds.) Proceedings of the 11th International Symposium of Formal Methods Europe (FME'02), Copenhagen, Denmark. Lecture Notes in Computer Science, vol. 2391, pp. 410–429. Springer (Jul 2002). https://doi.org/10.1007/3-540-45614-7_23, full version available as INRIA Research Report 4492
24. Garavel, H., Lang, F.: SVL: a Scripting Language for Compositional Verification. In: Kim, M., Chin, B., Kang, S., Lee, D. (eds.) Proceedings of the 21st IFIP WG 6.1

International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'01), Cheju Island, Korea. pp. 377–392. Kluwer Academic Publishers (Aug 2001). https://doi.org/10.1007/0-306-47003-9_24, full version available as INRIA Research Report RR-4223

25. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. Springer International Journal on Software Tools for Technology Transfer (STTT) **15**(2), 89–107 (Apr 2013). https://doi.org/10.1007/s10009-012-0244-z
26. Garavel, H., Lang, F., Serwe, W.: From LOTOS to LNT. In: Katoen, J.P., Langerak, R., Rensink, A. (eds.) ModelEd, TestEd, TrustEd – Essays Dedicated to Ed Brinksma on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 10500, pp. 3–26. Springer (Oct 2017). https://doi.org/10.1007/978-3-319-68270-9_1
27. Garavel, H., Serwe, W.: The Unheralded Value of the Multiway Rendezvous: Illustration with the Production Cell Benchmark. In: Hermanns, H., Höfner, P. (eds.) Proceedings of the 2nd Workshop on Models for Formal Analysis of Real Systems (MARS'17), Uppsala, Sweden. Electronic Proceedings in Theoretical Computer Science, vol. 244, pp. 230–270 (Apr 2017). https://doi.org/10.4204/EPTCS.244.10
28. Gillespie, D.T.: Exact Stochastic Simulation of Coupled Chemical Reactions. The Journal of Physical Chemistry **81**(25), 2340–2361 (Dec 1977). https://doi.org/10.1021/j100540a008
29. Hahn, E., Hartmanns, A., Hermanns, H., Katoen, J.P.: A Compositional Modelling and Analysis Framework for Stochastic Hybrid Systems. Formal Methods in System Design **43**(2), 191–232 (2013). https://doi.org/10.1007/S10703-012-0167-Z
30. Hartmanns, A., Hermanns, H.: In the Quantitative Automata Zoo. Science of Computer Programming **112**, 3–23 (2015). https://doi.org/10.1016/j.scico.2015.08.009
31. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The Quantitative Verification Benchmark Set. In: Vojnar, T., Zhang, L. (eds.) Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'19), Prague, Czech Republic. Lecture Notes in Computer Science, vol. 11427, pp. 344–350. Springer (Apr 2019). https://doi.org/10.1007/978-3-030-17462-0_20
32. Hensel, C., Junges, S., Katoen, J., Quatmann, T., Volk, M.: The Probabilistic Model Checker Storm. International Journal on Software Tools for Technology Transfer **24**(4), 589–610 (2022). https://doi.org/10.1007/S10009-021-00633-Z
33. Hermanns, H.: Interactive Markov Chains: The Quest for Quantified Quality, Lecture Notes in Computer Science, vol. 2428. Springer (2002). https://doi.org/10.1007/3-540-45804-2
34. Hermanns, H., Herzog, U., Klehmet, U., Mertsiotakis, V., Siegle, M.: Compositional performance modelling with the TIPPtool. Performance Evaluation **39**(1-4), 5–35 (Feb 2000). https://doi.org/10.1016/S0166-5316(99)00056-5
35. Hermanns, H., Herzog, U., Merksiotakis, V.: Stochastic Process Algebras as a Tool for Performance and Dependability Modelling. In: Iyer, R.K. (ed.) Proceedings of the International Computer Performance and Dependability Symposium (IPDS'95), Erlangen, Germany. pp. 102–111. IEEE (Apr 1995). https://doi.org/10.1109/IPDS.1995.395813

36. Hermanns, H., Joubert, C.: A Set of Performance and Dependability Analysis Components for CADP. In: Garavel, H., Hatcliff, J. (eds.) Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03), Warsaw, Poland. Lecture Notes in Computer Science, vol. 2619, pp. 425–430. Springer (Apr 2003). https://doi.org/10.1007/3-540-36577-X_30
37. Herzog, U., Merksiotakis, V.: Stochastic Process Algebras Applied to Failure Modelling. In: Herzog, U., Rettelbach, M. (eds.) Proceedings of the 2nd Workshop on Process Algebras and Performance Modelling (PAPM'94), Regensberg/Erlangen, Germany. pp. 107–126 (Jul 1994), https://www.researchgate.net/publication/2731331
38. Hillston, J.: A Compositional Approach to Performance Modelling. Ph.D. thesis, University of Edinburgh, Scotland, United Kingdom (Dec 1994), https://hdl.handle.net/1842/15027
39. Hillston, J.: The Nature of Synchronisation. In: Herzog, U., Rettelbach, M. (eds.) Proceedings of the 2nd Workshop on Process Algebras and Performance Modelling (PAPM'94), Regensberg/Erlangen, Germany. pp. 143–160 (Jul 1994), https://www.researchgate.net/publication/2311019
40. Hoare, C.A.R.: Communicating Sequential Processes. Commun. ACM **21**(8), 666–677 (Aug 1978). https://doi.org/10.1145/359576.359585
41. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs, NJ (1985)
42. INMOS Limited: OCCAM 2 Reference Manual. International Series in Computer Science, Prentice-Hall (1988)
43. ISO/IEC: LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, International Organization for Standardization – Information Processing Systems – Open Systems Interconnection, Geneva (Sep 1989)
44. ISO/IEC: Enhancements to LOTOS (E-LOTOS). International Standard 15437:2001, International Organization for Standardization – Information Technology, Geneva (Sep 2001)
45. Katoen, J.P.: Quantitative and Qualitative Extensions of Event Structures. Ph.D. thesis, University of Twente, The Netherlands (Apr 1996). https://doi.org/10.3990/1.9789036507998
46. Katoen, J.: The Probabilistic Model Checking Landscape. In: Grohe, M., Koskinen, E., Shankar, N. (eds.) Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, (LICS'16), New York, NY, USA. pp. 31–45. ACM (Jul 2016). https://doi.org/10.1145/2933575.2934574
47. Katoen, J., Kemna, T., Zapreev, I.S., Jansen, D.N.: Bisimulation Minimisation Mostly Speeds Up Probabilistic Model Checking. In: Grumberg, O., Huth, M. (eds.) Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07), Braga, Portugal. Lecture Notes in Computer Science, vol. 4424, pp. 87–101. Springer (Mar–Apr 2007). https://doi.org/10.1007/978-3-540-71209-1_9
48. Kemeny, J.G., Snell, J.L., Knapp, A.W.: Denumerable Markov Chains, Graduate Texts in Mathematics, vol. 40. Springer-Verlag, 2nd edn. (1976)
49. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of Probabilistic Real-Time Systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11), Snowbird, UT, USA. Lecture Notes in Computer Science, vol. 6806, pp. 585–591. Springer (Jul 2011). https://doi.org/10.1007/978-3-642-22110-1_47

50. Kwiatkowska, M.Z., Norman, G., Parker, D.: The PRISM Benchmark Suite. In: Proceedings of the 9th International Conference on Quantitative Evaluation of Systems (QEST'12), London, UK. pp. 203–204. IEEE Computer Society (Sep 2012). https://doi.org/10.1109/QEST.2012.14, https://prismmodelchecker.org/benchmarks
51. Lang, F.: Compositional Verification using SVL Scripts. In: Katoen, J.P., Stevens, P. (eds.) Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02), Grenoble, France. Lecture Notes in Computer Science, vol. 2280, pp. 465–469. Springer (Apr 2002). https://doi.org/10.1007/3-540-46002-0_33
52. Lindemann, C.: Employing the Randomization Technique for Solving Stochastic Petri Net Models. In: Lehmann, A., Lehmann, F. (eds.) Proceedings of the 6th GI/ITG Conference on Modelling, Measurement and Evaluation of Computing Systems (MMB'91), Neubiberg, Germany. Informatik-Fachberichte, vol. 286, pp. 306–319. Springer (Sep 1991). https://doi.org/10.1007/978-3-642-76934-4_21
53. Mateescu, R., Serwe, W.: Model Checking and Performance Evaluation with CADP Illustrated on Shared-Memory Mutual Exclusion Protocols. Science of Computer Programming **78**(7), 843–861 (Jul 2013). https://doi.org/10.1016/j.scico.2012.01.003
54. May, D.: OCCAM. SIGPLAN Notices **18**(4), 69–79 (1983). https://doi.org/10.1145/948176.948183
55. Milner, R.: A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol. 92. Springer (1980). https://doi.org/10.1007/3-540-10235-3
56. Milner, R.: Communication and Concurrency. Prentice-Hall (1989)
57. Sighireanu, M., Catry, A., Champelovier, D., Garavel, H., Lang, F., Schaeffer, G., Serwe, W., Stoecker, J.: LOTOS NT User's Manual (Version 3.14) (Jun 2024), INRIA/CONVECS, Grenoble, France, https://vasy.inria.fr/ftp/traian/manual.pdf, 88 pages