Nondeterminism in Interactive Markov Chains, with Application to the Erlangen Mainframe

Hubert Garavel[1] and Holger Hermanns[2]

[1] Univ. Grenoble Alpes, Inria, Cnrs, Grenoble Inp, Lig, F-38000 Grenoble, France
[2] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
hubert.garavel@inria.fr, hermanns@cs.uni-saarland.de

Abstract. To formally describe and numerically analyse the performance of systems consisting of concurrent stochastic processes, there exist two orthogonal approaches: "compound" models, each transition of which carries both an event (as in process algebra) and a rate (as in exponential distributions), and Interactive Markov Chains, each transition of which carries either an event or a rate. We investigate the subtle differences between both approaches, with a particular focus on the presence of nondeterminism in Interactive Markov Chains, and the ways nondeterminism can be either handled (resulting in Markov automata) or eliminated (resulting in Continuous-Time Markov Chains). We apply these ideas to the Erlangen mainframe, a challenging problem that exhibits the limitations of classical queueing theory by featuring multiple processors, job queues of different priorities, parallel composition with multiway synchronisation between two or more concurrent processes, and aperiodic failure/repair events.

1 Introduction

The present article was written in honour of Christel Baier and included in a collective *Festschrift* book offered to her on the occasion of her 60th birthday.

The work presented here builds upon the solid foundations established by Christel Baier for modelling and analysing complex systems that combine functional behaviour and quantitative aspects. To model such systems, a difficult challenge lies in the proper integration of process calculi and (discrete-time or continuous-time) Markov chains. Mainstream process calculi are inherently nondeterministic (e.g., due to their choice operators and to the interleaving semantics of their parallel composition operators), whereas the concept of nondeterminism is absent in Markov chains. Thus, it is not straightforward to define conservative extensions of a process calculi with Markov chains. A beautiful, though not sufficiently well known approach can be found in Christel Baier's habilitation thesis [1, Chap. 4], which provides a blueprint, by moving to Markov decision processes, for incorporating discrete time in process calculi.

The work presented here faces another challenge relating to the efficient verification of large quantitative models. Christel Baier also pioneered the development of verification algorithms for Markovian models. On the discrete-time

side, she contributed substantially to model-checking algorithms for Markov decision processes and discrete-time Markov chains [1, Chap. 9–10]. On the continuous-time side, which is at the core of our work, Christel Baier (together with Holger Hermanns, Joost-Pieter Katoen and, later, Boudewijn Haverkort) designed major model-checking algorithms for continuous-time Markov chains and continuous-time Markov decision processes. Many of her papers are acknowledged as reference publications, e.g., [4], which received a CONCUR Test-of-Time Award (2022), [2], which received the IFIP Jean-Claude Laprie Award in Dependable Computing (2022), or [3], which is a most-cited TCS paper.

These results have been transferred to other continuous-time models, such as interactive Markov chains and Markov automata, and implemented in modern verification and performance-evaluation tools. The work presented here substantially reuses these results, while exploring less known facets of the modelling and analysis of stochastic systems. Specifically, the present article addresses three related issues:

1. Two types of state-transition models have been proposed for the formal description of continuous-time stochastic systems, in which delays are governed by exponential distributions of known rate parameters. In the first type, each transition consists of an event and a rate glued together, while in the second type, each transition carries either an event or a rate. Both models are theoretically different but have been proposed to describe the same classes of systems. This raises compatibility issues (to which extent are both models interchangeable, in the sense that any of them could be used for a given system?), as well as choice issues (which model should be used preferably?).
2. The next issues concern the possibility of automated conversions between both types of models. Specifically, we define a translation function that converts models of the first type into models of the second type. We study how faithful this translation is with respect to crucial properties such as deadlocks, determinism, and the preservation of steady-state and transient probabilities.
3. Finally, we study the issues related to the presence of nondeterminism in the models of the second type produced by our translation function. We analyse the reasons why such nondeterminism appears and discuss how to cope with it or to eliminate it using successive transformations of the model.

We investigate these issues on a concrete case study, the "Erlangen mainframe", for which models of the first type already exist, and for which we develop a new model of the second type. This example is concise but involved enough to raise all major issues that may arise with continuous-time stochastic systems.

The present article is organised as follows. Section 2 introduces the models of the first type ("compound models") and the models of the second type ("interactive Markov chains"); it defines a translation function between both types of models and compares them with respect to essential properties; it also addresses related questions about state spaces, such as the proper access to information contained in states and transitions. Section 3 introduces the Erlangen mainframe, its already existing formal models, and the quantitative properties to be

measured on this system. Section 4 presents a new model, based on interactive Markov chains, for the Erlangen mainframe and shows how this model must be extended to express the quantitative properties of interest. Section 5 reports about the (nondeterministic) state spaces generated from this formal model and the various correctness checks performed during and after generation. Section 6 presents the results of the quantitative analyses of these nondeterministic state spaces using two state-of-the-art tools for Markov automata. Section 7 suggests an alternative approach in which nondeterminism is gradually removed from the model by using two types of transformations that preserve either the atomicity or the chronology of sequences of event/rate transitions. Finally, Section 8 gives concluding remarks and suggests directions for future work. The set of model files and scripts mentioned in the present article is publicly available on Zenodo[3].

2 Events-and-Rates versus Event-or-Rates

2.1 Events-and-Rates: Compound Models

A first approach to formally describe the functional behaviour and quantitative performance of concurrent systems with stochastic time is to adopt *compound models*. These are state-transition models, the transitions of which are pairs (a, λ), where a is an *event*, possibly synchronised with other events of processes executing concurrently, and where λ is a *rate* that defines a delay exponentially distributed with parameter λ (mean: $1/\lambda$). Compound models are also called *integrated models* elsewhere, e.g., in [5].

The theory of compound models has been explored throughout various process calculi, among which MTIPP [25] [39], PEPA [42], and EMPA [6]. The essential difference lies in the semantics chosen for the synchronisation of two compound transitions (a, λ) and (a, μ). MTIPP defines the result of this synchronisation as a transition with the product of rates $(a, \lambda\mu)$. PEPA computes the maximum of mean delays while incorporating the individual synchronisation capacities of processes. Finally, EMPA forbids this type of synchronisation and requires one process to impose the rate, e.g., (a, λ), while the other process(es) must accept any rate, written $(a, \mathbb{1})$, where $\mathbb{1}$ denotes a neutral element semantically different from the constant 1.0.

These ideas have been implemented in many dedicated software tools (compilers, model checkers, etc.), a few of which are actively maintained. It was also shown recently [20, Sect. 4] that compound models obeying the EMPA restriction on synchronisation (which we call the "*one-to-many*" property) can easily be described in LNT [13] and, by extrapolation, in any value-passing concurrent language that supports CSP-like rendezvous synchronisation, even without specific provisions for modelling stochastic aspects, provided that the compiler for this language preserves multiple identical transitions (i.e., transitions are stored in a multiset, not a set).

[3] https://doi.org/10.5281/zenodo.15243150

The compound models given below as examples satisfy the one-to-many property. They are written in a small process calculus, using only a few operators having their usual CSP/LOTOS/LNT semantics (see [18] for details): "(a, λ)" denotes an *active* compound transition that imposes rate λ; "$(a, \mathbb{1})$" denotes a *passive* compound transition that accepts any rate; "$P^\bullet$" denotes the infinite repetition of process P; "$P\,;Q$" denotes the sequential composition of process P followed by process Q; "$P \,\Box\, Q$" denotes the nondeterministic choice between processes P and Q; "$P \parallel Q$" denotes the parallel composition without synchronisation (i.e., full interleaving) of processes P and Q; "$P \parallel_E Q$" the parallel composition of processes P and Q with synchronisation on the set E of events. Operator ";" has higher syntactic precedence than all other binary operators.

Table 1 summarizes the semantics of this small process calculus. In this table, "$\checkmark$" denotes a special event indicating the termination; "ϵ" denotes the process that terminates immediately; C denotes a compound event that is either (a, λ) or $(a, \mathbb{1})$; the (overloaded) predicate $C \in E$ is true iff C is (a, λ) or $(a, \mathbb{1})$ and $a \in E$; L denotes an event that is either (a, λ), or $(a, \mathbb{1})$, or $\checkmark$; the predicate $sync(L, L_1, L_2, E)$ is true iff $L = L_1 = L_2 = \checkmark$, or $L = L_1 = L_2 = (a, \mathbb{1})$ and $a \in E$, or $L = L_1 = (a, \lambda)$ and $L_2 = (a, \mathbb{1})$ and $a \in E$, or $L = L_2 = (a, \lambda)$ and $L_1 = (a, \mathbb{1})$ and $a \in E$ — thus, any violation of the one-to-many property results in a local deadlock, as it is impossible to synchronise (a, λ) with (a, μ). The operator $P \parallel Q$ is defined as a shorthand for $P \parallel_\varnothing Q$.

$$\frac{}{\epsilon \xrightarrow{\checkmark}} \qquad \frac{}{C \xrightarrow{C} \epsilon} \qquad \frac{P \xrightarrow{C} P'}{P^\bullet \xrightarrow{C} P'\,;P^\bullet} \qquad \frac{P \xrightarrow{C} P'}{P\,;Q \xrightarrow{C} P'\,;Q} \qquad \frac{P \xrightarrow{\checkmark} \quad Q \xrightarrow{L} Q'}{P\,;Q \xrightarrow{L} Q'}$$

$$\frac{P \xrightarrow{L} P'}{P \,\Box\, Q \xrightarrow{L} P'} \qquad \frac{Q \xrightarrow{L} Q'}{P \,\Box\, Q \xrightarrow{L} Q'} \qquad \frac{P \xrightarrow{C} P' \quad C \notin E}{P \parallel_E Q \xrightarrow{C} P' \parallel_E Q} \qquad \frac{Q \xrightarrow{C} Q' \quad C \notin E}{P \parallel_E Q \xrightarrow{C} P \parallel_E Q'}$$

$$\frac{P \xrightarrow{L_1} P' \quad Q \xrightarrow{L_2} Q' \quad sync(L, L_1, L2, E)}{P \parallel_E Q \xrightarrow{L} P' \parallel_E Q'}$$

Table 1. Structural Operational Semantics rules for the small "compound" calculus

2.2 Events-or-Rates: Interactive Markov Chains

Compound models are not entirely satisfactory for, at least, two reasons:

- Compound transitions (a, λ) can be introduced in a process calculus, but they tend to modify the intuitive meaning of the choice and parallel composition operators, relative to the original process calculus.
- When modelling "real" systems, compound transitions force the modeller to attach a rate to every event. In practice, the concrete values of rates are not known for all events in a system, and rates are often useful for only a few,

well-chosen events. One can always replace an unknown or irrelevant rate by the neutral rate $\mathbb{1}$, but this approach is not fully convincing.

For these and other reasons, Holger Hermanns and Joost-Pieter Katoen proposed *Interactive Markov Chains* (IMCs, for short), an alternative model in which events a and rates λ are dissociated transitions [32] [38]. Among other qualities, such dissociated transitions (which are called *orthogonal* in [5]) allow the introduction of rates only where actually needed and deliver conservative extensions of the process calculi they are incorporated to.

IMCs have been equipped with model-checking facilities by adapting and extending analysis tools already available for untimed concurrent systems [15], and also by developing dedicated algorithms for timed reachability [43] [51] [53].

IMCs (expressed in LOTOS, Statemate or, more recently, LNT) have been used in many case studies, including an access control mechanism of ATM networks [32, Chap. 4.6], a plain old telephony system [37] [32, Chap. 6.2], a reliability model for the Hubble space telescope[4] [31], the arbitration mechanism of the SCSI-2 bus protocol[5] [19], a turntable system for drilling products[6] [47], a signalling system for high-speed trains [7], a distributed cache-coherence protocol [14], a multiprocessor data-flow architecture for multimedia streams [16], a comprehensive collection of mutual exclusion protocols for shared-memory systems[7] [49], etc.

From a methodological point of view, the typical use of IMCs for modelling and analysing a system can be decomposed into successive steps:

1. The system under study is formally modelled in a language that supports the concepts of IMCs. There is no language dedicated to IMCs, but they can easily be described in value-passing process calculi, such as LOTOS or LNT. In the formal model, rates are described in the same way as events, but given symbolic names (usually, λ, μ, etc.). A key principle of IMCs is that synchronising rates is forbidden, even if they are equal.
2. From the formal model, a Labelled Transition System (LTS, for short) is generated. Its transitions are labelled either with rates, or with visible events, or with τ, which is the CCS notation for hidden events [50]. Duplicated rate transitions having the same source state, the same target state, and the same label should be preserved (that is, rate transitions are stored in a multiset).
3. On the rate transitions of the LTS, all symbolic rates are replaced by their concrete values (e.g., all rates π are replaced by 3.14).
4. On the event transitions of the LTS, all visible events are renamed to τ, which makes them both invisible and urgent.
5. The LTS is modified using *maximal progress* to enforce the urgency of τ-transitions: any rate transition going out of some state s is removed if there exists a τ-transition also going out of state s.

[4] `https://cadp.inria.fr/ftp/demos/demo_30`
[5] `https://cadp.inria.fr/ftp/demos/demo_31`
[6] `https://cadp.inria.fr/ftp/demos/demo_39`
[7] `https://cadp.inria.fr/ftp/demos/demo_10`

6. The LTS is minimised according to an equivalence relation compatible with the IMC semantics. Throughout the present article, we only consider *stochastic branching bisimulation*, which combines branching bisimulation [24] and lumpability [44]. This minimisation preserves self-loops labelled with τ when they are attached to states having no other outgoing transition (contrary to branching bisimulation). We also call IMC the state-transition model obtained after minimisation.
7. If the IMC contains only rate transitions, it is said to be *deterministic* and falls in the category of Continuous-Time Markov Chains (CTMCs, for short).
8. If the IMC contains at least one τ-transition, it is said to be *nondeterministic* and falls in the category of Markov Automata (MAs, for short).

This methodology is not inflexible. Certain steps could be permuted (e.g., 3 and 4) or merged with other steps (e.g., 5 and 6, or even 2 and 5). More complex scenarios are also possible, especially with *compositional verification* of IMCs [37] [19], which we do not address in the present article.

Like compound models, our IMC examples given below are written in a small LNT-like process calculus having the following operators: "λ" (or any other Greek letter but τ) denotes a rate transition; "a" denotes a visible event; "τ" denotes the hidden event; "$P^{\bullet}$" denotes infinite repetition; "$P\,;Q$" denotes sequential composition; "$P \,\square\, Q$" denotes nondeterministic choice; "$P \parallel Q$" denotes parallel composition without synchronisation; "$P \parallel_E Q$" denotes parallel composition with synchronisation. Again, operator ";" has higher syntactic precedence than all other binary operators.

Table 2 summarizes the semantics of this small process calculus. In this table, "$\checkmark$" denotes a special event indicating the termination; "ϵ" denotes the process that terminates immediately; C denotes either a visible event a, or the hidden event τ, or a rate transition λ; L denotes an event that is either C or $\checkmark$; in any parallel operator $\parallel_E$, the synchronisation set E should contain only visible events; yet, our semantics also covers the extended case where E contains rates, which is normally not accepted in the usual IMC theory. Again, the operator $P \parallel Q$ is defined as a shorthand for $P \parallel_{\varnothing} Q$.

$$\frac{}{\epsilon \xrightarrow{\checkmark}} \qquad \frac{}{C \xrightarrow{C} \epsilon} \qquad \frac{P \xrightarrow{C} P'}{P^{\bullet} \xrightarrow{C} P'\,;P^{\bullet}} \qquad \frac{P \xrightarrow{C} P'}{P\,;Q \xrightarrow{C} P'\,;Q} \qquad \frac{P \xrightarrow{\checkmark} \quad Q \xrightarrow{L} Q'}{P\,;Q \xrightarrow{L} Q'}$$

$$\frac{P \xrightarrow{L} P'}{P \,\square\, Q \xrightarrow{L} P'} \qquad \frac{Q \xrightarrow{L} Q'}{P \,\square\, Q \xrightarrow{L} Q'} \qquad \frac{P \xrightarrow{C} P' \quad C \notin E}{P \parallel_E Q \xrightarrow{C} P' \parallel_E Q} \qquad \frac{Q \xrightarrow{C} Q' \quad C \notin E}{P \parallel_E Q \xrightarrow{C} P \parallel_E Q'}$$

$$\frac{P \xrightarrow{L} P' \quad Q \xrightarrow{L} Q' \quad L \in E \cup \{\checkmark\}}{P \parallel_E Q \xrightarrow{L} P' \parallel_E Q'}$$

Table 2. Structural Operational Semantics rules for the small IMC calculus

2.3 Standard Translation from Compound Models to IMCs

Any compound model S that satisfies the one-to-many property can be translated to an IMC denoted $[\![S]\!]$ by applying the rewrite rules given in Tab. 3. Rule (1) is the most significant one: it expresses that any compound transition (a, λ) translates to a sequence of two transitions, in which event a occurs after a random delay of rate λ. Rule (2) expresses that any compound transition $(a, \mathbb{1})$ translates to a single transition a. The remaining rules (3)–(7) state that the translation function $[\![\cdot]\!]$ is a morphism with respect to the repetition, sequence, choice, and parallel operators.

$$
\begin{array}{rl}
[\![(a,\lambda)]\!] = \lambda;a & (1)\\
[\![(a,\mathbb{1})]\!] = a & (2)\\
[\![P^\bullet]\!] = [\![P]\!]^\bullet & (3)\\
[\![P\,;Q]\!] = [\![P]\!]\,;[\![Q]\!] & (4)\\
[\![P \,\Box\, Q]\!] = [\![P]\!] \,\Box\, [\![Q]\!] & (5)\\
[\![P \parallel Q]\!] = [\![P]\!] \parallel [\![Q]\!] & (6)\\
[\![P \parallel_E Q]\!] = [\![P]\!] \parallel_E [\![Q]\!] & (7)
\end{array}
$$

Table 3. Rules defining the standard translation from compound models to IMCs

Although Rule (2) looks surprising at first, it is justified by the fact that the fundamental equality $(a, \lambda) \parallel_{\{a\}} (a, \mathbb{1}) = (a, \lambda)$ is preserved by $[\![\cdot]\!]$, as both terms of the equation translate to $(\lambda; a)$.

This translation function is quite similar to the encoding from MTIPP to IML [32] given by Marco Bernardo et al. in [5, Def. 5.1], with two differences concerning rules (1) and (2): (i) MTIPP uses action prefix, whereas our calculus models use sequential composition; (ii) MTIPP defines the synchronisation of rates as a product of rates, so that $\mathbb{1}$ corresponds to the numeric rate 1.0 in MTIPP; thus, the encoding of [5] does not have rule (2) and uses instead rule (1) with $\lambda = 1.0$.

Despite these differences, we proudly call our function $[\![.]\!]$ the *standard translation* from compound models to IMCs — especially to distinguish it clearly from the other translations and translators presented in Sect. 6.2 below.

Alternative translations could be considered instead. On the one hand, we are not aware of any approach that would translate (a, λ) to a sequence $(a; \lambda)$ — conventionally, λ represents a preliminary delay, which should be spent before event a, rather than after it. On the other hand, one could imagine an approach that would translate (a, λ) to a sequence $(a'; \lambda; a'')$ and $(a, \mathbb{1})$ to a sequence $(a'; a'')$, where the events a' and a'' respectively correspond to the start and the end of the original event a. We have not explored this approach and only rely, throughout the present article, on the standard translation.

2.4 Compound Models vs IMCs

Much could be written about the comparison of compound models and IMCs, although some questions are still open, especially in the presence of synchronisation and nondeterminism. We focus here on three key points:

1. IMCs are theoretically more expressive than compound models because they can express both probabilistic choice and nondeterministic choice, whereas compound models can only express probabilistic choice.
2. There exist results establishing that strong Markovian bisimilarity is preserved for processes that do not contain synchronisation [5]. Precisely, if two MTIPP compound models are bisimilar, their translations to IML by the aforementioned encoding of [5, Def. 5.1] are bisimilar. Conversely, if two IML models are bisimilar, their translations to MTIPP using the reverse encoding of [5, Def. 6.1] are also bisimilar. These results are valuable but cannot be reused directly in the present article, since rule (2) of our standard translation differs from the encoding of [5, Def. 5.1], and since our case study intensively uses synchronisations, as shown below in Sect. 3.2.
3. A crucial issue is the preservation of steady-state and transient probabilities between compound models and IMCs. When strong Markovian bisimilarity is preserved, e.g., under the assumptions of [5], both steady-state and transient probability distributions over states are preserved. However, the situation is not always so propitious.
   If the IMC contains nondeterminism, then one gets [min, max] intervals instead of single probabilities. This is a major difference, which raises a new question: are the single probabilities computed for a compound model S within the bounds of the [min, max] intervals computed for $[\![S]\!]$?
   Without nondeterminism, but in the presence of synchronisations, transient probabilities are not preserved, as shown by the following example.

   Example 1. Consider the compound system $S_1 := (a, \lambda) \parallel_{\{a\}} (b, \mu); (a, \mathbb{1})$. Because of the synchronisation on event a, the left-hand side operand cannot spend time (i.e., λ) before the right-hand side has fully executed (b, μ). The corresponding CTMC is, thus, the sequence "$\mu; \lambda$". The standard translation of this system is $[\![S_1]\!] = \lambda; a \parallel_{\{a\}} \mu; b; a$. In this IMC, the rates λ and μ start elapsing from the initial state and the corresponding CTMC is the diamond "$\lambda \parallel \mu$". Hence, $[\![\cdot]\!]$ does not preserve transient probabilities. ■

   The next example (suggested by Marco Bernardo) shows that steady-state probabilities are also not preserved in the presence of synchronisations.

   Example 2. Consider the system $S_2 := (a, \lambda)^\bullet \parallel_{\{a\}} \big((b, \mu); (a, \mathbb{1})\big)^\bullet$, which is the compound system S_1 of Example 1 modified by adding repetitions to ensure that both concurrent processes never terminate, so as to obtain an ergodic CTMC. The corresponding CTMC has two states and can be expressed as $(\mu; \lambda)^\bullet$. The steady-state probability of being in the initial state of this CTMC is $\lambda/(\lambda+\mu)$. The standard translation of this system is $[\![S_2]\!] =$

$(\lambda; a)^\bullet \parallel_{\{a\}} (\mu; b; a)^\bullet$. The corresponding CTMC has three states and can be expressed as $(\lambda; \mu \,\Box\, \mu; \lambda)^\bullet$. The steady-state probability of being in the initial state of this CTMC is $\lambda\mu/(\lambda^2 + \mu^2 + \lambda\mu)$. Thus, $[\![\cdot]\!]$ does not preserve steady-state probabilities. ■

In the next sections, we extend to other kinds of properties this comparison between compound models and their IMCs obtained by standard translation.

2.5 Alternation in IMCs

In compound models, events and rates alternate on all execution paths: every event is preceded by a rate, and vice versa (except in the initial state). In IMCs, this alternation property does not hold in general: rates are only introduced where they are actually needed, so that several events not separated by rates may occur in sequence. It is possible, however, to consider only *strictly alternating* IMCs, whose semantics was shown to coincide with that of Continuous-Time Markov Decision Processes (CTMDPs) [43]. One may wonder whether the IMCs generated from compound models are strictly alternating or not. The answer is negative in the general case, and difficult to predict at intermediate steps:

- The standard translation clearly breaks alternation by removing all $\mathbb{1}$ rates, as each compound transition $(a, \mathbb{1})$ translates to a.
- Parallel composition may restore alternation, as in $(a; b \parallel_{\{a\}} \lambda; a)$, or break it, as in $(\lambda; a \parallel \mu; b)$, where interleaving enables λ and μ to occur in sequence.
- The successive application of event hiding and maximal progress may also restore alternation by removing sequences of rates, as in $(\lambda; a \parallel \mu; b)$, where maximal progress cuts a λ-transition that follows a μ-transition and a μ-transition that follows a λ-transition.
- Finally, stochastic branching bisimulation breaks alternation by removing τ-transitions. If all of them are removed, the minimised IMC is a CTMC with only rate transitions, making the notion of alternation meaningless. If some τ-transitions remain, the minimised IMC is a MA, a model that does not impose any constraint concerning alternation.

2.6 Deadlocks in IMCs

Another point to be mentioned is that the standard translation does not preserve the absence of deadlocks, and may indeed produce IMCs containing "artefact" deadlocks that did not exist in the original compound models.

Example 3. Consider a system $S := P \parallel_{\{a,b\}} Q$ consisting of two concurrent processes P and Q that synchronise on events a and b. Assume that both P and Q are infinitely looping processes that offer a choice between a $\mathbb{1}$ rate and a non-$\mathbb{1}$ rate: $P := ((a, \lambda) \,\Box\, (b, \mathbb{1}))^\bullet$ and $Q := ((a, \mathbb{1}) \,\Box\, (b, \mu))^\bullet$. This compound model has no deadlock. Its standard translation to IMCs is $[\![S]\!] = [\![P]\!] \parallel_{\{a,b\}} [\![Q]\!]$, where $[\![P]\!] = (\lambda; a \,\Box\, b)^\bullet$ and $[\![Q]\!] = (a \,\Box\, \mu; b)^\bullet$. The LTS of $[\![S]\!]$ contains a deadlock,

which is reached when P does a λ-transition and Q does a μ-transition, in any order. This deadlock is not removed later by applying maximal progress and stochastic branching bisimulation. ■

Such deadlocks can be explained as follows: the standard translation converts any compound transition (a, λ) into a sequence "$\lambda; a$" of two transitions. Following the IMC methodology, the λ-transition should not be synchronised with anything else. It can thus evolve freely (exactly like τ-transitions) and may cause deadlocks if it appears in a context of nondeterministic choice.

It is worth noticing that maximal progress may not introduce deadlocks, as it only cuts rate transitions that are in choice with τ-transitions, leaving such τ-transitions unchanged. Stochastic branching bisimulation does not introduce deadlocks either since, unlike "standard" branching bisimulation, it does not remove τ-loops.

It has been suggested by Pedro d'Argenio that deadlocks could be avoided if all concurrent processes in the compound model would be either *active* (i.e., do not use the rate $\mathbb{1}$) or *passive* (i.e., only use the rate $\mathbb{1}$). The following example illustrates this idea.

Example 4. Consider a system $S' := P' \parallel_{\{a,b\}} Q'$ where $P' := ((a, \lambda) \,\Box\, (b, \mu))^\bullet$ and $Q' := ((a, \mathbb{1}) \,\Box\, (b, \mathbb{1}))^\bullet$. This compound model is bisimilar to system S of Example 3. Its standard translation to IMCs is $[\![S']\!] = [\![P']\!] \parallel_{\{a,b\}} [\![Q']\!]$, where $[\![P']\!] = (\lambda; a \,\Box\, \mu; b)^\bullet$ and $[\![Q]\!] = (a \,\Box\, b)^\bullet$. The LTS of $[\![S']\!]$ contains no deadlock; this property is preserved by maximal progress and stochastic branching bisimulation. ■

Two remarks can be drawn from the comparison of these examples. First, bisimilarity is not preserved by the standard translation, as the two compound systems S and S' are bisimilar, while their translations $[\![S]\!]$ and $[\![S']\!]$ are not (the LTS and IMC of $[\![S]\!]$ both have deadlocks, and the IMC contains τ-transitions, which is not the case for $[\![S']\!]$).

Second, these examples suggest a strategy to remove deadlocks by carefully permuting, in the compound model before translation, (a, λ) and $(a, \mathbb{1})$ transitions across concurrent processes, still preserving bisimilarity at the compound-model level. However, such transformations might not be always possible, e.g., when several active processes compete to synchronise with, and impose their rates to, a passive process.

2.7 Nondeterminism in IMCs

As mentioned before, IMCs can genuinely express nondeterminism, if it is an explicit intention of the specifier. But nondeterminism may also arise unexpectedly when translating a compound model to an IMC.

Example 5. Consider the system $S_1 := \big((a, \lambda)^\bullet \parallel (a, \mu)^\bullet\big) \parallel_{\{a\}} \big((b, \varphi); (a, \mathbb{1})\big)^\bullet$. Its standard translation is $[\![S_1]\!] = \big((\lambda; a)^\bullet \parallel (\mu; a)^\bullet\big) \parallel_{\{a\}} (\varphi; b; a)^\bullet$. In this

example, repetitions are intended to avoid deadlocks. The IMC of $[\![S_1]\!]$ is nondeterministic in the case both delays specified by λ and μ expire before the delay specified by φ. Indeed, from the initial state of the IMC, there are two sequences "$(\lambda \parallel \mu); \varphi; (\tau \,\Box\, \tau)$" leading to a nondeterministic state.

Incidentally, this scenario is referred to as the "lunch with Christel" problem. Christel invites two friends, with the intention of going for lunch with the one who arrives first. Her plan is safe, because they both need different exponentially distributed delays (λ and μ) to join her, so that the probability that they arrive at the same moment is null. But Christel needs a cigarette and goes to the smoking area. Smoking also takes an exponentially distributed delay (φ). If it is too long (especially if φ is much smaller than λ and μ), the two friends may be already there when she returns from the smoking area, and then, she faces a choice problem.

There is a similar situation in queueing theory. When a user has the choice between two waiting queues, the strategy "join the shortest queue" avoids nondeterminism in many cases. But if both queues are equally filled, the choice is necessarily nondeterministic. ■

One might argue that Example 5 is peculiar and that its nondeterminism primarily comes from the interleaving between two occurrences of the same event a on the left-hand side of the $\|_{\{a\}}$ operator, meaning that nondeterminism was already present and intended in the compound model. The next example refutes this objection.

Example 6. Consider $S_2 := \big((a,\lambda)^\bullet \parallel (b,\mu)^\bullet\big) \parallel_{\{a,b\}} \big((c,\varphi); ((a,\mathbb{1}) \,\Box\, (b,\mathbb{1}))\big)^\bullet$. Its standard translation is $[\![S_2]\!] = \big((\lambda;a)^\bullet \parallel (\mu;b)^\bullet\big) \parallel_{\{a,b\}} \big(\varphi;c;(a \,\Box\, b)\big)^\bullet$. The IMC of $[\![S_2]\!]$ is nondeterministic since, from its initial state, there are two sequences "$(\lambda \parallel \mu); \varphi; (\tau \,\Box\, \tau)$" leading to nondeterminism. ■

In Examples 5 and 6, and in the more complex situations presented in Sect. 7.5, 7.7, and 7.8 below, the occurrences of nondeterminism arising in IMCs generated from compound models seem to follow a common scenario. Two processes P and Q execute in parallel, without being synchronised together. Maximal progress ensures the natural alternation of their rates and events. It also preserves some form of "atomicity", in the sense that a rate λ originating from a given compound transition (a,λ) is immediately followed by its corresponding event a. But processes P and Q are synchronised on certain events a and b, respectively, with a third process R. At some point, R blocks both events a and b, but cannot block their corresponding rate transitions λ and μ — since rate transitions are never synchronised in classical IMC theory. This breaks the alternation of rates and events, and also breaks atomicity, as occurrences of λ and μ are no longer followed by occurrences of a and b, respectively. The system continues to evolve and, eventually, R unlocks both a and b, which suddenly become enabled at the same moment, hence causing nondeterminism.

Examples 5 and 6 are simple, but we tried hard to make them even simpler. Doing so, we experimented with dozens of variations and observed that

nondeterminism in IMCs is "fragile", like unstable physical particles or chemical compounds, and tends to disappear in most cases, unless certain precise conditions are met. As a result, it does not occur so often statistically, which may explain why former case studies based on IMCs did not face this issue. Yet, nondeterminism remains a possibility that must be considered.

2.8 Building and Debugging State Spaces

A practical difference between compound models and IMCs lies in their respective state spaces. Because the standard translation replaces each compound transition (a, λ) by a sequence of two transitions "$\lambda; a$", the LTSs generated from IMCs are likely to be much larger (possibly by several orders of magnitude, as shown in Sect. 5.3 below) than those produced from compound models.

Such an increased risk of state explosion has two implications. First, IMCs cannot be used in the absence of robust tools capable of handling large state spaces. Second, to avoid generating huge LTSs that will be considerably reduced later by stochastic branching bisimulation, one should use *compositional verification* techniques [23], which have been advocated quite early for IMCs [34] [37] [19].

Models of stochastic systems, like any concurrent program, may be wrong, e.g., contain unexpected behaviour, unreachable code, deadlocks, etc. Thus, it is often necessary to debug them so as to understand and correct the mistakes. In the case of compound models, one can apply the standard debugging techniques to the LTSs generated from compound models, the transitions of which are labelled with pairs (a, λ). In the case of IMCs, debugging is more difficult, as one must examine models at two different levels:

- The IMCs themselves contain little information, as all the visible transitions have been hidden and, later, sequences of τ-transitions have been compacted by stochastic branching bisimulation. Most transitions are labelled with numeric rates, and of some these rates may correspond to different symbolic rates, or be the sum of several other rates as a consequence of lumpability. There may also be τ-transitions, if the model contains nondeterminism, and self-loop transitions, as explained in Sect. 2.10 below. From such a low-level information, it is often tedious to trace down the correspondence with the high-level formal model from which the IMC was produced.
- The LTSs take place half-way between the high-level formal models and the low-level IMCs. LTSs are easier to understand, since they contain all event transitions, with their visible names, and all rate transitions, with their symbolic rates. However, LTSs may contain large parts of unreachable code eliminated later by maximal progress.

Therefore, debugging of large IMCs is not feasible using visual inspection only. Dedicated tools that extract and combine information from both LTSs and IMCs must be developed for this task — an example is given in Sect. 7.4 below.

2.9 Rate Tagging

IMCs create additional difficulties for expressing and computing throughputs of rate transitions, state probabilities, and throughputs of events. Let us start by the former point.

The throughput of a rate transition λ is the sum, for all states s having an outgoing transition λ, of the probability, multiplied by λ, of being in state s.

Normally, λ is a symbolic rate that corresponds to a situation of interest in the system under study. For instance, in the Hubble space telescope case study [31], the throughput of rate μ expresses the probability of entering the so-called "sleep mode".

Two difficulties may occur: (i) there exist several symbolic rates having the same concrete value as λ, making it difficult to distinguish between λ and these other rates in the IMC; (ii) lumpability may remove some rate transitions λ and create new rate transitions where λ is summed with other rates.

These issues can be avoided as follows: when substituting λ with its concrete value, e.g., 100, one may attach a *tag* to the concrete value to remember that it comes from the replacement of λ. For instance, instead of replacing λ with 100, one may replace it with "a; `rate` 100" [36]. Notice that "a; `rate` λ" neither denotes a compound transition (a, λ) nor a sequence of two transition "$a; \lambda$"; it denotes a rate transition, in which tag a is merely a decoration.

Then, maximal progress and stochastic branching minimisation are applied. Notice that the presence of tags (i.e., when two rate transitions have different tags, or when one has a tag and the other not) prevents states to be merged by bisimulation and rate transitions to be merged by lumpability.

This *rate tagging* approach has been intensively used in prior applications of IMCs: mutual exclusion[8], Hubble[9], SCSI-2[10], drilling unit[11], etc. It has two limitations: (i) it only applies to rate transitions, but not to event transitions, the throughput of which may also be of interest; (ii) it is not supported by all tools and, thus, not "portable" across different tools.

2.10 State Probes

Another difficulty lies in the expression of state probabilities in IMCs. More generally, this difficulty is faced with all event-based formalisms, e.g., process calculi, where information is not attached to states, but to transitions. In such formalisms, state variables are not visible and, thus, their values cannot be used to characterise sets of states, the probabilities of which is of interest for computing steady-state or transient quantitative properties.

The solution to this problem is well-known: it consists in exporting state-based information (e.g., the value of state variables) through additional transitions (called *state probes*) introduced in the model. Such extra-transitions usually

[8] `https://cadp.inria.fr/ftp/demos/demo_10/demo.svl` (search for "rate␣")
[9] `https://cadp.inria.fr/ftp/demos/demo_30/demo.svl` (search for "rate␣")
[10] `https://cadp.inria.fr/ftp/demos/demo_31/demo.svl` (search for "rate␣")
[11] `https://cadp.inria.fr/ftp/demos/demo_39/demo.svl` (search for "rate␣")

have the form of *self-loops* attached to certain states, the information relevant to these states being given by the transition labels. The concept of probes is not specific to IMCs: probes are also used in compound models, as well as in non-quantitative models of concurrent systems.

Traditionally, in the latter case (e.g., models written in untimed process calculi), any probe should respect three "neutrality" conditions: (i) it is a visible transition, i.e., not renamed to τ; (ii) it is not synchronised with any other transition; and (iii) it only displays information present in the state it is attached to, but does not modify this state, e.g., by assigning state variables.

Hence, the introduction of probes in a model does not increase the number of states, but only the number of transitions. Moreover, probes are preserved when minimising the model for strong or branching bisimulation.

In the case of IMCs, however, the situation is less simple. One may wonder about the nature of probes: are they event transitions or rate transitions?

- If probes are event transitions (and assuming they remain visible, rather than being hidden), they are preserved by maximal progress (since only rate transitions are cut by τ-transitions), but they may prevent stochastic branching minimisation from merging certain states, potentially increasing the numbers of states and τ-transitions in the resulting IMCs. A frequent situation is the case of a transition $s_1 \xrightarrow{\tau} s_2$, where state s_1 has only a self-loop probe and where state s_2 has only outgoing rate transitions. If probes are events, minimisation will neither merge states s_1 and s_2, nor remove the τ-transition, thus leaving nondeterminism in the IMC.
- If probes are rate transitions (i.e., if a probe with label z is transformed into "z;`rate` λ" by rate tagging), they may be cut by maximal progress if they are in choice with a τ-transition. In the previous example $s_1 \xrightarrow{\tau} s_2$, the probe attached to s_1 is cut by the τ-transition, allowing minimisation to merge s_1 and s_2 and to remove the τ-transition. This is not systematic: because of concurrency and synchronisations, many probes are not deleted by maximal progress after visible transitions have been hidden. Yet, cutting probes may be annoying, as they convey relevant information about states (e.g., being in a critical section, etc.). However, state s_1 here is the source state of a τ-transition, which is considered to be immediate; thus, the probability of s_1 should be zero, so that this state plays no role in calculations of state probabilities and event throughputs.

In the remainder of the present article, we consider probes as rate transitions: each probe z is changed to "z;`rate` λ" before applying maximal progress and branching stochastic minimisation. All the experiments reported below are based on this assumption.

Notice that adding such probes does not alter the state probabilities, because self-loops, which correspond to diagonal elements, are discarded when building the generator matrix of a CTMC. We give to λ the concrete value 1.0, so that the sum of state probabilities for all the states s having the same self-loop z can be computed as the throughput of "z;`rate` 1.0".

The appropriate placement of state probes in IMCs obtained by standard translation of compound models is discussed in Sect. 4.4 below.

2.11 Event Probes

One sometimes needs to compute the throughput of an event transition a, rather than the throughput of a rate transition as in Sect. 2.9. This is difficult in IMCs, since all visible transitions are hidden and later compacted by minimisation, making it impossible to recognise transitions labelled with a. To address this issue:

- A first option would be not to hide the event a if one needs to compute its throughput, but this would prevent minimisation to be done completely. Consequently, we discard this approach.
- A second option would be to identify, if it exists, some rate λ closely related to the event a of interest, so that the throughput of a can be expressed in terms of the throughput of λ. This is not always easy, nor even possible but, if the IMC has been generated by standard translation from a compound model, one may consider choosing the rate λ associated to a in the compound transitions (a, λ).
- A third option would be to modify the LTS and the IMC by adding self-loop probes z (hereafter called *event probes*) to well-chosen states, so that the throughput of event a can be expressed in terms of the throughput of z.

A combination of the second and third options is presented in Sect. 4.5 below, so as to compute, in an IMC generated by standard translation, the throughput of a transition (a, λ) present in the original compound model.

3 The Erlangen Mainframe

3.1 Origin and History of the Problem

The mainframe example was designed in the mid-1990s by the IMMD-7 research team at the University of Erlangen. The IMMD-7 group leader, Ulrich Herzog, had identified three major issues in the design of modern distributed systems: (i) the manageability problems due to increasing complexity; (ii) the fundamental uncertainty in timing and state caused by randomly occurring transmission errors, leading to varying traffic intensities and communication delays; and (iii) the issues of partitioning and allocating tasks of proper granularity on computational resources for optimal performance and dependability. He was convinced that neither classical queueing theory, nor the informal and ad-hoc extensions to queueing networks proposed at that time, were sufficient to handle complex systems and address the resulting challenges.

The publications made by IMMD-7 in the 1990s, e.g. [40] [25] [35], provide pioneering ideas about formal methods and tools, incremental and compositional design, validation, and maintenance, with a particular focus on process-algebraic

extensions for discrete- and continuous-time — a topic to which Christel Baier also greatly contributed during her PhD thesis. These ideas were illustrated with many interesting case studies, among which the Erlangen mainframe [41, Sect. 4] [34, Sect. 4] seems to be the oldest example proposed by IMMD-7.

3.2 System Description

This case study represents a multiprocessor computer 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, failures and priorities, are present. *Failures* may cause system downtimes, by making the mainframe become unavailable until it is repaired. *Priorities* of two types are built into the system. On the one hand, database users need immediate reaction, so they *explicitly* have priority over the jobs issued by programmers. On the other hand, 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 Erlangen mainframe is modular and hierarchical. It consists in ten processes that execute concurrently and synchronise together using n-party ("multiway") rendezvous, i.e., simultaneous events involving n processes, where n can be as large as 6. The various interactions between these processes are shown in Fig. 1.

The Erlangen mainframe is a compound model, as each event is associated with a rate. Table 4 lists the nine possible events and their corresponding rates.

event	synchronisation pattern	rates
c	3-party rendezvous	$\varphi = 0.00334$
prog_job	3-party rendezvous	$\lambda_1 = 0.01667$ or $\lambda_2 = 0.16$
user_job	3-party rendezvous	$\mu_1 = 0.033$ or $\mu_2 = 2$
fail	6-party rendezvous	$\delta_1 = 0.00035$ or $\delta_2 = 0.0007$
repair	5-party rendezvous	$\beta = 0.01$
get_prog_job	4-party rendezvous (3 queues, 1 processor)	$\alpha = 48$
get_user_job	3-party rendezvous (2 queues, 1 processor)	$\alpha = 48$
prog_job_ready	no rendezvous (interleaving)	$\xi = 0.3$
user_job_ready	no rendezvous (interleaving)	$\nu = 12$

Table 4. Events, synchronisations, and rates in the Erlangen mainframe

On the topmost level, the Erlangen mainframe can be seen as the parallel composition of three principal modules, namely, the *loads*, the *queues*, and the *processors*:

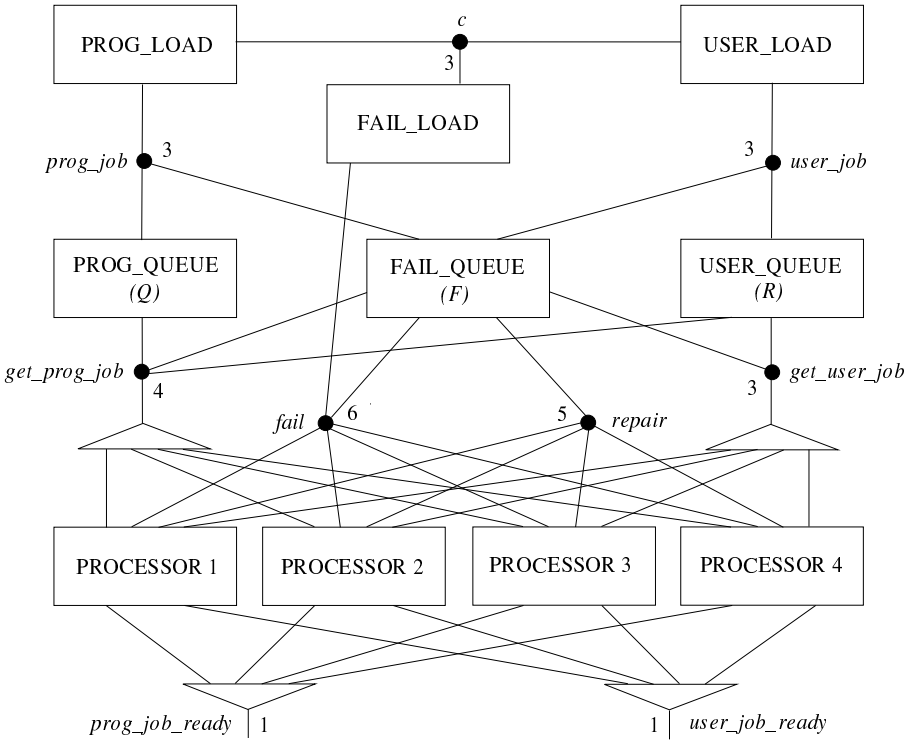

Fig. 1. Architecture of the Erlangen mainframe (black bullets represent n-ary synchronisations and white triangles represent 1-among-4 competitions for synchronisation)

- *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 produces 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 (MMPP) [17]. Each arrival stream has three phases, which come with different arrival rates for the streams: extended working hours (low load: λ_1, μ_1, and δ_1), normal working hours (high load: λ_2, μ_2, and δ_2), and night hours (no activity). The phase changes are governed by yet another rate φ and happen simultaneously across the streams: this is achieved by synchronising the three load processes, which otherwise run independently in parallel, on a common event c that indicates a phase change.
- *Queues*: The mediation between the events arriving from the loads and the processors is handled by three queues. The *prog_queue* buffers the jobs generated by programmers; the *user_queue* buffers the jobs generated by database users; 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*: The mainframe possesses four identical processors running in parallel. Each processor receives from the queues, at rate α, programmers' jobs and users' jobs (via the *get_prog_job* and *get_user_job* events) and delivers these jobs (via the *prog_job_ready* and *user_job_ready* events) at rates ξ and ν. The processors synchronise altogether on failures and repairs, meaning that failures affect the entire system, halting all processors, until repair. When a failure occurs, the jobs being executed by the processors are silently discarded and not delivered, but the contents of queues are preserved.

The combination of all these features (multiple queues, multiple processors, priorities, failures/repairs, three-phase Markov-Modulated Poisson Processes, and multiway synchronisations) makes the Erlangen mainframe significantly more involved and challenging than the standard models studied by queueing theory.

3.3 Quantitative Properties

The original paper [41, Sect. 4.3] provides eight figures corresponding to various analysis scenarios. These figures are numbered from 3 to 10 — we surround their numbers by square boxes to distinguish them from the figure numbers of the present article. These eight figures have been carefully designed to exercise diverse kinds of quantitative properties, as summarised in Tab. 5:

- The first line gives the type of property: "S" for steady-state probabilities (i.e., in the long term, after an equilibrium has been reached) or "T" for "transient" probabilities (i.e., at specific time instants).
- The second line gives the kind of property: "s" for state probabilities (i.e., the probability to be in a certain set of states characterised by a predicate over variables of the mainframe) or "t" for event throughputs.
- The third line gives the parameters that may vary in each figure, e.g., the concrete values of certain rates (β, δ_2, and μ_2), the current size of the *prog_queue* or *user_queue*, the current time instant for transient properties, or the initial phase of the arrival streams.
- The fourth line gives the maximal sizes of the *prog_queue* and the *user_queue*. These values are those used in [20].
- The fifth line gives the number of different state spaces that need to be generated or explored for each figure. These state spaces differ by the values of the corresponding parameters. We call *instances* these variants of the same model, following the terminology of the Model Checking Contest [45]. Certain pairs of figures (namely, Fig. 5–6, Fig. 7–8, and Fig. 9–10) are constructed using the same set of instances; this is why their columns in Tab. 5 are (partially) merged.

– The sixth line gives the number of properties to be evaluated on each instance. The number of plots to be drawn for this figure is the product of the number of instances by the number of properties. The indication "(+1)" for Fig. 3 and 4 means that, although one does not need to compute the probabilities that the queues are empty (these probabilities would be much too large compared to the other values in these figures), one needs to compute the extra properties mentioned in Sect. 6.3.

figure	3	4	5	6	7	8	9	10
type	S	S	S	S	T	T	T	T
kind	s	s	s	t	s	t	s	t
parameters	prog queue, μ_2	user queue, μ_2	β, δ_2		β, time		phase, time	
queue sizes	(40, 4)	(10, 10)	(4, 4)		(4, 4)		(4, 4)	
nb of instances	10	10	36		3		3	
nb of properties per instance	40(+1)	10(+1)	1	1	15	15	16	16

Table 5. Overview of the quantitative properties

3.4 Compound Models of the Erlangen Mainframe

The first formal model of the mainframe [41, Sect. 4] appeared in 1994. It was specified using the TIPP (Timed Processes and Performance Evaluation) process algebra and analysed with the TIPPtool software developed at Erlangen [33], which is no longer maintained. A few variants (also expressed in TIPP) of this model have been explored, e.g. in [34, Sect. 4], where the four processors are replaced by one sequential process.

Thirty years later, two novel models of the Erlangen mainframe have been designed [20]. The first model can be processed by either the PRISM[12] [46] or Storm[13] [30] model checkers; it is written in the PRISM language and its files can be found in the PRISM library of models[14]. The second model was developed using the CADP[15] [22] verification toolbox; it is written in the LNT language and its files are available from the list of CADP demo examples[16].

This work brought a few corrections and simplifications to the original TIPP model of the Erlangen mainframe, e.g., by pointing out that the size of queues can be greatly reduced without noticeably affecting steady-state and transient probabilities. It was shown in [20] that the PRISM and LNT models can closely

[12] `https://prismmodelchecker.org`
[13] `https://www.stormchecker.org`
[14] `https://www.prismmodelchecker.org/files/erlangen`
[15] `https://cadp.inria.fr`
[16] `https://cadp.inria.fr/demos.html` (see "demo_15")

reproduce the eight Figures 3 to 10 originally produced using the TIPP model. It was also shown that the various CTMCs generated by PRISM and CADP are pairwise equivalent modulo strong stochastic bisimulation. For these reasons, we adopt the PRISM and LNT models of [20] as reference specifications to build upon in the present work.

3.5 IMC Models of the Erlangen Mainframe

The three aforementioned specifications of the Erlangen mainframe (in TIPP, PRISM, and LNT languages) all are compound models. So far, there exists no IMC model, and it is tempting to know whether the Erlangen mainframe could be described using IMCs.

To progress this question, the mainframe problem was given in 2022, as a homework exercise, to seven master students of Univ. Grenoble Alpes. Starting from the TIPP specification given in [41], their mission was to develop an LNT specification and reproduce the eight figures 3 to 10. The students were offered the possibility to use either compound models or IMCs; all of them chose IMCs — likely because prior lectures had put emphasis on teaching IMC concepts.

Although we were expecting that modelling the Erlangen mainframe using IMCs would be straightforward, it turned out that this was not the case. Most students managed to produce, using the standard translation (see Sect. 2.3), an LNT specification and generate the corresponding IMC using the CADP tools, but they did not succeed in reproducing the eight figures. Indeed, the generated IMCs were subsequently rejected by the CADP tools BCG_STEADY[17] and BCG_TRANSIENT[18] [36] because these IMCs contained (i) unreachable states, (ii) deadlocks, and/or (iii) τ-transitions (i.e., nondeterminism).

Issue (i) has been understood and solved: the presence of unreachable states in the generated IMCs was due to the BCG_MIN[19] tool, which implements branching stochastic minimisation. Indeed, BCG_MIN may cut certain transitions if their rate is close to zero or if they are of low priority according to maximal progress. This might create unreachable states, even if, in the model given as input to BCG_MIN, all states are reachable from the initial state. Although this feature was agreed upon by experts and properly documented in the manual page of BCG_MIN, it remained potentially confusing for novice users. To address this issue, Frédéric Lang extended, in September 2023, the probabilistic and stochastic bisimulation algorithms of BCG_MIN to ensure that the output model is truly minimal, i.e., contains only reachable states.

Concerning issue (ii), it appeared that most deadlocks arose from incorrect parallel compositions, which is no surprise given the complex synchronisation patterns (see Fig. 1 and Tab. 4) between the ten processes of the Erlangen mainframe.

[17] https://cadp.inria.fr/man/bcg_steady.html
[18] https://cadp.inria.fr/man/bcg_transient.html
[19] https://cadp.inria.fr/man/bcg_min.html

Concerning issue (iii), a preliminary analysis concluded that the causes of nondeterminism were unclear and that the Erlangen mainframe was substantially more involved than all prior case studies done using IMCs, where nondeterminism would usually vanish after application of maximal progress and stochastic branching bisimulation. This raised rekindled interest in the Erlangen mainframe and prompted a deeper analysis of this problem.

4 Formal Modelling of the Erlangen Mainframe

4.1 Choice of the Modelling Language

The compound models developed for the Erlangen mainframe are either based on process calculi (TIPP and LNT languages) or on automata (PRISM language). For modelling the mainframe using IMCs, we chose LNT, for two reasons: (i) since their inception, IMCs have been closely associated to process calculi, successively TIPP, LOTOS, and LNT; (ii) IMCs in general, and IMC models of the mainframe in particular, may contain nondeterminism and, thus, result in Markov automata, which are not supported by the PRISM tool.
To develop an IMC model in LNT of the mainframe, we started from the compound model in LNT [20] and applied the standard translation rules (see Sect. 2.3). This transformation was straightforward. The main difference between the input and output LNT models is that, in the IMC model, rates are expressed as events whereas, in the compound model, they are expressed as data variables. The transformation thus converted, in all LNT processes, most value parameters into event parameters. Yet, a few value parameters remain in certain processes, e.g., to represent queue sizes or initial phases of Markov-Modulated Poisson Processes.

In the remainder of this section, we detail specific modelling aspects that are not covered by the standard translation.

4.2 Modelling φ-rates

In the Erlangen mainframe, each change of phase (between extended working hours, normal working hours, and night) for the three arrival streams is modelled using a three-party synchronisation on an event c with the rate $\varphi = 0.00334$. This situation can be expressed in various ways. In the PRISM compound model (see [20, Fig. 2]), the three queue processes all propose $(c, \mathbb{1})$ and synchronise together with a fourth additional process that proposes (c, φ) and, thus, imposes rate φ. In the LNT compound model (see [20, Fig. 3]), there is no such additional process: each queue process proposes (c, φ) and the three-way synchronisation of these transitions results in a (c, φ) transition, according to the parallel composition rules of LNT — unlike the parallel composition rules of TIPP, which would give (c, φ^3) in such case (see [20, Sect. 2.3(2)] for a discussion).

For the IMC version of the mainframe, both solutions could have been reused. Yet, we did not retain the PRISM solution, because we preferred not to introduce

an extra process, and we did not retain the LNT solution, because it would have required to synchronise the three queue processes on both c and φ, although synchronisation of rates is discouraged in IMCs. Instead, we opted for a third solution, in which the *fail_queue* process proposes $[\![(c, \varphi)]\!]$, i.e., "$\varphi; c$", while the two other processes *prog_queue* and *user_queue* both propose $[\![(c, \mathbb{1})]\!]$, i.e., "c". These three processes are synchronised on c (but not on φ) and a Boolean parameter *delayed* is introduced to distinguish *fail_queue* from *prog_queue* and *user_queue*.

4.3 Modelling α-rates

In all compound models, two different events have the same rate, namely (get_prog_job, α) and (get_user_job, α), with $\alpha = 48$ (see Tab. 4). Given that these events respectively appear in two concurrent processes *prog_queue* and *user_queue*, this entails, in the LTSs obtained after translation, nondeterminism between two sequences of transitions "$\alpha; get_prog_job$" and "$\alpha; get_user_job$".

Such an "overloading" of rates α is likely to make nondeterminism harder to explain. Therefore, in our LNT model, we distinguish both rates α by giving them different names α_1 (for *get_prog_job*) and α_2 (for *get_user_job*), but keeping the same value $\alpha_1 = \alpha_2 = 48$. However, when searching for the shortest sequences leading to nondeterministic states (see Sect. 7.4 below), we give to α_2 a slightly different value (e.g., 49) to unambiguously back-translate rate sequences to event sequences.

4.4 Introducing State Probes

The compound model in LNT of the Erlangen mainframe contains three state probes, each defined in one of the three queue processes [20, Sect. 4]:

- The probe of the *fail_queue* process attaches a self-loop labelled "*z_avail*" to each global state in which the mainframe is available (i.e., is not in failed state). This state probe is needed for Fig. 5, 7, and 9 only.
- The probe of the *prog_queue* process attaches a self-loop labelled "*z_prog_queue*(n)" to each global state in which the *prog_queue* contains n programmer jobs ($n \geq 0$). This state probe is needed for Fig. 3 only.
- Similarly, the probe of the *user_queue* process attaches a self-loop labelled "*z_user_queue*(n)" to each global state in which the *user_queue* contains n user jobs ($n \geq 0$). This state probe is needed for Fig. 4 only.

These three probes must also be present in the IMC model of the mainframe. Specifying the *z_avail* probe in the IMC setting is easy. The LNT code for this probe (see Fig. 2, left) is the same as that of the compound model in LNT.

Specifying the two other probes is more involved. We only consider here the *z_prog_queue* probe — the case of *z_user_queue* being fully symmetric. In the compound model, *z_prog_queue* is straightforward, but the IMC model is different, as the standard translation splits the compound transition (α_1, get_prog_job)

in two successive transitions $s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{get_prog_job} s_3$. If the probe is inserted in the IMC model exactly in the same way as in the compound model, a self-loop "*z_prog_queue*(n)" will appear on state s_1 and another self-loop "*z_prog_queue*$(n-1)$" will appear on state s_3. This correctly expresses the fact that event *get_prog_job* removes one element from the queue.

But what about the intermediate state s_2 created by the IMC semantics? Should it have also a self-loop "*z_prog_queue*(n)" or not? There are conflicting arguments on this matter.

- For the self-loop: after the delay of rate α_1, the queue still contains n elements until event *get_prog_job* occurs, which may take time if all the processors are busy; it is thus logical to have a self-loop "*z_prog_queue*(n)" on s_2; moreover, in this particular case, the numeric value of α_1 is so large that little time will be spent in state s_1.
- Against the self-loop: the event *get_prog_job* will be ultimately hidden, i.e., renamed to τ; if the intermediate state s_2 has no self-loop, then stochastic branching bisimulation will remove this τ-transition and merge both states s_2 and s_3 in one state; this would be impossible if s_2 had a self-loop with a different label than that of s_3, and such a residual τ-transition could introduce nondeterminism as an artefact in the model.

To decide this question, we devised a probabilistic criterion that the correct modelling solution should satisfy and we experimented both approaches using Storm. The results (see Sect. 6.3 below) convinced us that a self-loop should be put on state s_2.

The last difficulty was to specify this self-loop properly in LNT, taking into account that this probe should be enclosed between α_1 and *get_prog_job*. One must explicitly specify a finite loop, which can be interrupted at any time to let *get_prog_job* happen (see Fig. 2, right), and not an infinite loop running forever.

4.5 Introducing Event Probes

To reproduce Fig. 6, 8, and 10, one needs to compute the throughput of high-priority jobs (which is displayed on the y-axis of these figures), i.e., the throughput of event *get_user_job*. In the compound model, this event is visible and its throughput is the sum, for all states s having an outgoing transition (get_user_job, α_2), of the probability, multiplied by α_2, of being in state s.

In the IMC model of the mainframe, all events are hidden (and possibly merged by bisimulation minimisation), which prevents one from referring to *get_user_job* directly. We decided not to use rate tagging (see Sect. 2.9) on α_2, for two reasons:

- As pointed out in Sect. 2.9, rate tagging may prevent lumpability from merging rate transitions if they carry different tags, or if some of them have tags and others not. But lumpability certainly plays a role in the Erlangen mainframe, especially nearby the four processors. A posteriori, one can indeed

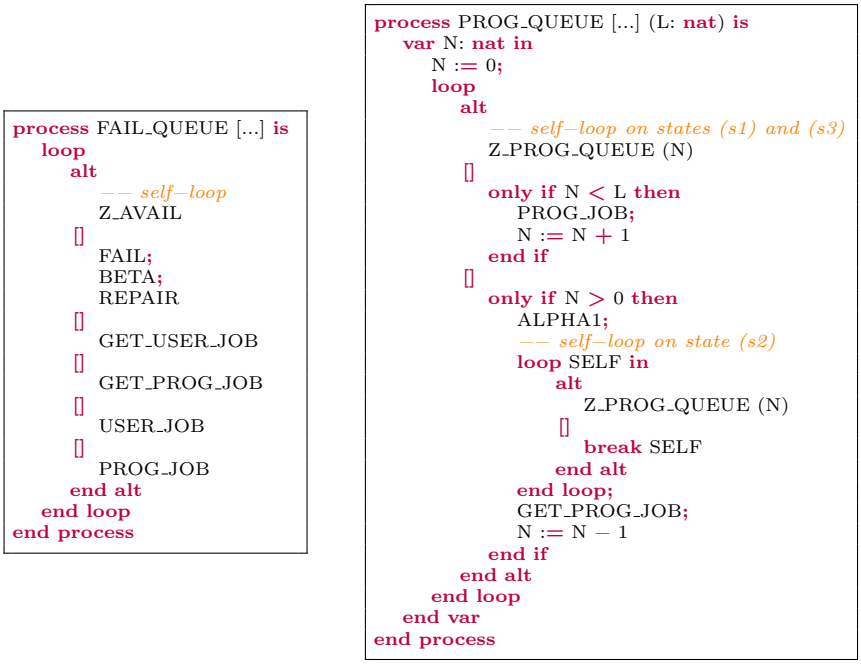

Fig. 2. LNT model fragments illustrating the placement of state probes

observe that the IMCs generated for the mainframe contain rate transitions 0.6, 0.9, 1.2, 24, 36, and 48 that presumably correspond to the lumped rates 2ξ, 3ξ, 4ξ, 2ν, 3ν, and 4ν (the latter rate might also correspond to α_1 or α_2), where $\xi = 0.3$ and $\nu = 12$ are the rates attached to the output events of the processors. The situation could very well have been similar for the input events *get_user_job* and *get_prog_job* of the processors. Today, we know retrospectively that it is not the case, as the IMCs contain no rates larger than α_1 and α_2 but, in doubt, we adopted a cautious modelling approach that avoids rate tagging.

– Rate tagging is implemented in the various tools of CADP that generate or manipulate CTMCs, but it is not supported by other tools capable of handling MAs. Rather than using this convenient, CADP-specific extension, we opted for event probes, which remain in the standard framework of IMCs (without rate tags) and are more likely to be "portable" across various analysis tools.

We thus decided to use event probes (see Sect. 2.11). The LNT specification of the mainframe was extended with an additional self-loop probe *z_get_user_job* intended to measure the throughput of the hidden event *get_user_job*.

The compound transition $(\alpha_2, \mathit{get_user_job})$ is split, when translated to IMCs, in two successive transitions $s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\mathit{get_user_job}} s_3$. Given that a throughput measures the probability of states before a given transition, the self-loop *z_get_user_job* must appear before event *get_user_job*, which excludes state s_3. But should this self-loop be attached to state s_1 only, to state s_2 only, or to both states s_1 and s_2? This problem is similar to the one of Sect. 4.4, yet different as it deals with throughputs instead of state probabilities.

At first sight, placing the self-loop *z_get_user_job* on state s_2 seems to be the most natural option, given that the throughput of *get_user_job* is defined using the probabilities of the states from which this event is enabled. However, this option should be excluded, since the self-loops attached to s_2 may very well disappear completely from the IMC because of maximal progress.

Example 7. Consider a compound system $S := ((a, \lambda); (b; \mu))^\bullet$ and its standard translation $[\![S]\!] = (\lambda; a; \mu; b)^\bullet$. Any self-loop probe inserted between λ and a, or between μ and b, will be cut by maximal progress after a and b are hidden. ■

However, the self-loops attached to s_2 may also remain in the global state space after minimisation by stochastic branching bisimulation, and the states to which these self-loops are attached may have non-null probabilities. This was confirmed by Storm on the various IMCs generated for Fig. 6.

Contrary to Sect. 4.4, where state probes had to be attached to both s_1 and s_2, so as to total the probabilities of these states, event probes should be attached to s_1 only, and not to both s_1 and s_2, because they express flows, not stocks.

Example 8. Consider a compound system $S := (a, \lambda)^\bullet \parallel_{\{a\}} ((b; \mu); (a, \mathbb{1}))^\bullet$ and its standard translation $[\![S]\!] = (\lambda; a)^\bullet \parallel_{\{a\}} (\mu; b; a)^\bullet$. Assume that the goal is to compute, on the IMC, the throughput of the compound transition (a, λ). To do so, three self-loops are inserted in $[\![S]\!]$: probe z is put before λ, probe z' is put between λ and a, and probe z'' is put between b and a. After minimisation, the resulting IMC has three states, four rate transitions, and four self-loops. Probe z has the intended meaning, as it is attached to all (and only those) states having an outgoing rate transition λ. Probe z' is irrelevant, as it is attached to one state having only one outgoing rate transition μ. Probe z'' is irrelevant as well, but for a different reason: it is attached to one state having an outgoing rate transition λ, but not to the initial state, which also has an outgoing rate transition λ; thus, probe z'' does not capture the entire mass of probability needed to compute the throughput of (a, λ). ■

So, our solution to compute the throughput of event *get_user_job* is to put a self-loop on state s_1 only, and to multiply the obtained probability by $\alpha_2 = 48$ in order to obtain values comparable with the compound model.

5 State-Space Generation

5.1 Models, Instances, and SVL Scripts

We now indicate how the LNT model of the Erlangen mainframe can be translated to IMCs, and how this is implemented concretely. Actually, there are several LNT models to be considered: the model of Sect. 4 will be hereafter referred to as V1 (for version 1), but there will be other models V2, V3, V4, etc. For each model, each of the eight figures Fig. 3 to 10 requires to generate between 3 and 36 instances (see Tab. 5).

All these models and instances create a notable complexity, with more than 1,800 files containing LNT code, temporal-logic properties, execution logs, probability/throughput results, as well as LTSs, MAs, and CTMCs in multiple formats for the various tools. Consequently, experiments cannot be carried out manually (the risk of mistakes would be too high) and must be properly automated.

To this aim, we developed a verification script written in the SVL language[20] [21]. This script (560 lines, 380 lines if not counting comments and blank lines) reuses and extends the SVL script formerly developed for the compound model of the Erlangen mainframe — see [20, Sect. 5.8] for a detailed presentation of this former script that, starting from the LNT compound model, performs all the steps needed to automatically produce the eight picture files for Fig. 3–10.

The SVL script developed for the IMC approach also starts from a given LNT model, generates the corresponding LTS, hides all visible events but probes, renames labels to replace symbolic rates by their concrete values, and invokes BCG_MIN to apply maximal progress and stochastic branching minimisation. It performs the required iterations on the aforementioned variables and rate parameters and seeks to avoid duplicated calculations by building intermediate models that mix symbolic and concrete rates.

After stochastic branching minimisation, the SVL script checks, using BCG_INFO[21], whether the resulting IMC is deterministic, i.e., free from τ-transitions. If so, the IMC is a CTMC: the script invokes BCG_STEADY or BCG_TRANSIENT to compute probability/throughput results and then converts these numbers into PNG pictures by reusing the eight Gnuplot scripts developed for the compound version of the mainframe. If not, the IMC is a MA: this case will be detailed in Sect. 6 below.

5.2 Sanity Checks

To make sure that our various models and instances of the Erlangen mainframe contain no obvious mistakes, we devised another SVL script that performs sanity checks about deadlocks, alternation, and probes.

Although deadlocks may appear when translating a deadlock-free compound model to an IMC (see Sect. 2.6), we found no deadlocks in the LTSs and IMCs

[20] https://cadp.inria.fr/man/svl-lang.html
[21] https://cadp.inria.fr/man/bcg_info.html

generated for the various instances. This is fortunate, given that the Erlangen mainframe does not satisfy the sufficient condition mentioned in Sect. 2.6 for avoiding deadlocks: six out of the ten mainframe processes (namely, *prog_queue*, *user_queue*, and the four processors) are both active and passive, i.e., contain both $\mathbb{1}$ and non-$\mathbb{1}$ rates. This also confirms that the deadlocks reported by some students (see Sect 3.5) were not specific to the Erlangen mainframe itself, but rather caused by synchronisation mistakes.

The IMCs obtained after stochastic branching minimisation are not CTMDPs, because they do not respect the mandatory alternation between events and rates. Indeed, for each of the eight figures, we checked that at least one instance enables, from its initial state, a sequence of two consecutive rate transitions labelled with $\varphi = 0.00334$. More generally, when the mainframe is in its initial state, one can perform an infinite sequence "$\varphi; c; \varphi; c; \varphi; c...$" that corresponds to the elapse of nights and days while the mainframe remains inactive. When the c events are hidden, the sequence of rates φ remain. This confirms the point of Sect. 2.5: alternation is not necessarily preserved when translating compound models to IMCs.

Concerning probes, the SVL script checks, using XTL[22] [48], that each transition labelled with a probe (namely, *z_avail*, *z_get_user_job*, *z_prog_queue*, and *z_user_queue*) is a self-loop, i.e., has identical source and target states. The SVL script also checks, using BCG_CMP[23], that the LTS obtained from a modified LNT model, in which all the probes have been removed, is strongly bisimilar to the LTS obtained from the original LNT model after deleting all probe transitions from the latter LTS.

5.3 Sizes of State Spaces

Table 6 gives a comparative overview of our state-space generation experiments. It is worth noticing that, in each column, all the instances for a given figure have the same number of states and transitions, as they only differ by the values of the varying parameters.

The first four lines give the sizes of the LTSs and CTMCs generated for the compound model of [20]. The LTSs have not been minimised and the CTMCs (which contain no τ-transitions) have been minimised for stochastic strong bisimulation — essentially to make sure that the various CTMCs generated by CADP, PRISM, and Storm are comparable, i.e., have similar sizes.

The last four lines give the sizes of the LTSs and MAs generated for the IMC model of the Erlangen mainframe. The LTSs have not been minimised and the MAs have been minimised for stochastic branching bisimulation after applying maximal progress and hiding all visible transitions.

These numbers confirm the point of Sect. 2.8: state spaces are much larger with the IMC approach than with compound models. Indeed, the numbers of states and transitions are 85 and 120 times larger for the LTSs, and 7.4 and 7 times larger for the CTMCs or MAs.

[22] `https://cadp.inria.fr/man/xtl.html`
[23] `https://cadp.inria.fr/man/bcg_cmp.html`

figures	3	4	5 and 6	7 and 8	9 and 10
nb of states compound/LTS	414,387	238,323	48,195	48,195	48,195
nb of transitions compound/LTS	3,188,415	1,826,679	361,389	361,389	361,389
nb of states compound/CTMC	9,840	5,808	1,200	1,200	1,200
nb of transitions compound/CTMC	55,785	33,069	6,570	6,570	6,570
nb of states IMC/LTS	38,834,935	23,274,055	4,213,159	4,213,159	4,213,159
nb of transitions IMC/LTS	382,382,187	230,112,423	41,065,683	41,065,683	41,065,683
nb of states IMC/MA	74,800	43,708	8,560	8,560	8,560
nb of transitions IMC/MA	387,515	227,243	47,060	47,060	47,060

Table 6. Statistics on state-spaces for the compound and IMC models

5.4 Execution Times

Similarly, the IMC approach takes more time than compound models. For the compound model of [20], generating the eight figures takes 10 min. 35 sec. on a Linux laptop with an Intel core i5 processor and 16 GB RAM. For the IMC approach, generating the eight figures for model V4 (see Sect. 7.8 below) that produces CTMCs and invokes BCG_STEADY and BCG_TRANSIENT (thus, having comparable workload with the compound model) takes nearly five hours on the same laptop.

In practice, execution times could be reduced in three ways: (i) potential optimisations have been identified in some CADP tools, but not implemented yet; (ii) experiments have been done sequentially to give an idea of the total time required, but could easily be run in parallel, given that the calculations needed for the various figures (and even for the multiple instances of a given figure) are largely independent one from the other; (iii) so far, compositional verification [19] [23] has not been used, since execution times, although long, were not prohibitive, but it would surely be effective given the large difference in size between the generated LTSs and the minimised MAs or CTMCs.

6 Nondeterministic Analyses

6.1 Analysis Tools for Markov Automata

All the IMCs generated in Sect. 5 for model V1 of the Erlangen mainframe are not CTMCs, as they contain nondeterministic (i.e., τ-transitions not eliminated by stochastic branching minimisation). Thus, they cannot be analysed

using PRISM or the BCG_STEADY and BCG_TRANSIENT tools of CADP. One must use tools that genuinely support Markov automata. To this aim, we selected Modest[24] version v3.1.273 (Oct. 2024) [29] [10] and Storm version 1.9.0 (Nov. 2024). These tools come with various constraints, so that our IMC-based approach must be adapted to fit the possibilities of each tool:

- The LNT model of the Erlangen mainframe extensively uses both parallel composition operators $\|$ and $\|_E$ to describe 1-among-n synchronisations (see the white triangles in Fig. 1). Currently, the native input languages of Modest and Storm only provide a CSP-like $\|$ operator that synchronises processes on their common events. To address this issue, one option would be to combine this CSP-like $\|$ operator with relabellings, so as to express 1-among-n synchronisations, while duplicating certain events in the IMCs to give them unique names — as can be seen in the PRISM compound model of the mainframe[25], in the three queue processes of which the *get_prog_job* and *get_user_job* events are replicated four times, one per processor. Another option would be to use the JANi interchange format [9], which has a $\|_E$ operator and translates automatically in the input languages of Modest and Storm. Eventually, we found it simpler to let CADP expand parallel composition and provide Modest and Storm with explicit state-transition models, from which concurrency has been eliminated.
- It does not seem that Modest and Storm implement the stochastic branching bisimulation needed for the IMC approach — although Storm has an option to quotient the state space using strong or weak bisimulation. It is thus appropriate to rely on the BCG_MIN tool of CADP and give to Modest and Storm models that have been minimised beforehand.
- The property languages of Modest and Storm do not allow to compute event throughputs, which are needed for Fig. 6, 8, and 10. However, in Sect. 4.5, we have shown how to represent event throughputs in terms of event probes. But the property languages of Modest and Storm support neither state probes nor event probes, as they cannot directly express the presence or absence of transitions entering or leaving a state — except perhaps by extending the model with the introduction of rewards [12]. Consequently, we must find a way to encode the concept of probes, so essential in the IMC approach, in terms of state variables, state labels, and state probabilities, which are the concepts understood by Modest and Storm.

6.2 Translators from IMCs to Modest and Storm

Given the large number of models and instances to analyse, the sole solution was to develop translators for automatically converting the explicit-state, minimised IMCs with probes into the input formats supported by Modest and Storm.

Modest accepts the Modest language [28] and the JANi interchange format, which is also supported by many other tools. Storm accepts Markov automata

[24] https://www.modestchecker.net

[25] https://www.prismmodelchecker.org/files/erlangen

in various formats: DRN, IMCA/TRA, JANi, PRISM-MA, etc. Faced with such a multiplicity, we were unsure about the right format to choose, especially with respect to the proper way of encoding probes. We finally opted for the Modest language, which is the native input language of Modest, and for PRISM-MA, which is a Storm extension for Markov automata of PRISM's input language we were already familiar with. We felt that both languages would be easy to generate and proofread.

We developed two translators, one for Modest, another for PRISM-MA, which follow the same principles:

- Each generated program contains a single process (or "module" in PRISM-MA terminology), without concurrency nor recursion.
- Assuming that the IMC has N states, a bounded-integer variable s in the range $[0, N-1]$ encodes the current state in the generated program.
- There is also one variable per state probe and one variable per event probe. Such additional *probe variables* store, in the generated program, the information conveyed by the presence or absence of self-loops on IMC states.
- Precisely, the state probe *z_avail* (resp. the event probe *z_get_user_job*) is translated to a Boolean probe variable *avail* (resp. *get_user_job*), which is true in any state iff the corresponding self-loop is attached to this state.
- Similarly, the state probe "*z_prog_queue*(n)" (resp. "*z_user_queue*(n)") is translated to a bounded-integer probe variable *prog_queue* (resp. *user_queue*) in the range $[-1, L]$, where L is the maximal size of the *prog_queue* (resp. *user_queue*). This probe variable equals $n \geq 0$ in any state iff a corresponding self-loop holding parameter n is attached to this state, or -1 if the state does no have such a self-loop.
- Each rate transition of the IMC is straightforwardly translated to a rate transition in the generated program.
- Each τ-transition (if any) of the IMC is translated to a nondeterministic transition in the generated program.
- Each probe self-loop of the IMC is translated, by its presence or absence, into assignments to the corresponding probe variable. Such a self-loop does not create any transition in the generated program and its rate is not used during the translation.
- In the generated program, each transition starts with a Boolean guard of the form "$s = n$" that checks if the current state is n. Probe variables are not used in these guards and, thus, do not change the number of states or transitions; they only decorate states with extra information.
- Each transition terminates with assignments that modify the value of the current-state variable s and, possibly, of the probe variables. The translation avoids generating useless assignments of the form "$x := v$" if variable x already has value v in the current state.
- With Storm, our translator generates, after the Markov automaton, a list of definitions for *state labels*, which express predicates on the probe variables and are used in the properties to evaluate.

– With Modest, the properties to evaluate are part of the model — whereas, with Storm, the Markov automaton and its properties are put in two separate files. For this reason, our translator for Modest gets an extra command-line argument, which is the name of a file containing the properties, and inserts these properties in the generated Modest model.

Technically, our translators are written in C (230 lines of code each, 180 lines if not counting comments and blank lines) and rely on the BCG_READ[26] programming interface for reading the input IMC, which is stored in a binary file using the BCG format of the CADP toolbox.

Our translators also perform various checks that confirm or extend those of Sect. 5.2, namely: (i) each transition labelled with a probe name is a self-loop; (ii) no state has exactly one outgoing τ-transition; (iii) each state has either no outgoing rate transition and at least one outgoing τ-transition, or no outgoing τ-transition and at least one outgoing rate transition.

6.3 Assumption Checking for State Probes

As discussed above in Sect. 4.4, it is unclear whether, in process *prog_queue*, a self-loop "*z_prog_queue*(n)" must be attached or not to the intermediate state s_2 located after rate α_1 and before event *get_prog_job*. The same question holds for process *user_queue* by replacing α_1, *get_prog_job*, and *z_prog_queue* with α_2, *get_user_job*, and *z_user_queue*, respectively.

To decide between compelling arguments for and against these self-loops, we formulate the following criterion that a correct modelling of the Erlangen mainframe should satisfy: any reachable global state should have a self-loop "*z_prog_queue*(n)", because the *prog_queue* is active at any time and always contains zero or more elements. Given that this probe is only needed for Fig. 3, which deals with steady-state probabilities, this criterion can be refined and generalised as follows: the long-run probability of being in a global state that does not have a self-loop "*z_prog_queue*(n)" should be zero. Obviously, a similar criterion should hold for "*z_user_queue*(n)" and Fig. 4.

We modified our Storm translator to generate, in the PRISM-MA models, an additional state label *prog_queue_no_loop* that identifies all states in which the *prog_queue* variable equals -1. We added two Storm formulas that compute the minimal and maximal long-run probabilities for this state label. Similar changes were also done for the *user_queue*.

We used Storm to analyse the Markov automata generated for Fig. 3 and Fig. 4 on two variants of the mainframe model, with and without self-loops on the intermediate states s_2. Notice that the ten instances generated for Fig. 3 (resp. Fig. 4) all give identical results for the present matter.

For the variant with self-loops: there are many states without self-loops (5,088 states without *z_prog_queue* probes, out of 74,800 states for Fig. 3, 2,940 states without *z_user_queue* probes, out of 43,708 states for Fig. 4). Our Storm translator indicates that each of these states without self-loops has no outgoing rate

[26] https://cadp.inria.fr/man/bcg_read.html

transition and more than one outgoing τ-transition. Given that τ-transitions are assumed to be immediate, the sojourn time in these states should be null. This is confirmed by Storm, which reports that the minimal and maximal long-run probabilities for these states are (exactly) zero.

For the variant without self-loops: interestingly, the number of τ-transitions in the Markov automata for Fig. 3 and Fig. 4 remains unchanged, compared to their variants with self-loops, thus showing that having self-loops on states s_2 does not necessarily increase nondeterminism. Also, Storm reports much larger numbers of states without self-loops (48,368 states without *z_prog_queue* probes, out of 74,800 states for Fig. 3, 19,520 states without *z_user_queue* probes, out of 43,708 states for Fig. 4). Our Storm translator indicates that each of these states without self-loops has (between 1 and 7) outgoing rate transitions, but no outgoing τ-transition — this may be explained by stochastic branching bisimulation merging the τ-transition arriving to state s_3 with the various rate transitions leaving this state s_3. Consequently, the sojourn time in the states without self-loops cannot be null, since only rate transitions are leaving these states. This is confirmed by Storm, which reports that the long-run probabilities for these states are greater than a minimal value ranging from 0.0002 to 0.001 for the ten instances of Fig. 3, and from 0.0008 to 0.01 for the ten instances of Fig. 4. These values are all but negligible compared to the other probabilities computed by Storm for these figures.

These results thus indicate that self-loops should be present on the intermediate states s_2 of processes *prog_queue* and *user_queue*.

6.4 Results of Steady-State Analyses

We experimented with Modest and Storm in order to obtain the steady-state probabilities needed for Fig. 3–6. We wrote, in the specific format required by each tool, properties that compute these probabilities and, given that all instances of model V1 contain nondeterminism, we asked each tool to compute both the minimal and maximal long-term probabilities, which had the effect of doubling the numbers of properties given on the last line of Tab. 5.

All the properties to evaluate on a given instance are put in the same file, which speeds up the analysis by ensuring that the instance is parsed only once.

For convenience, all our experiments with Modest and Storm were done in a VirtualBox virtual machine running Debian Linux 12 with two processors and 8 MB RAM, hosted on the same laptop as in Sect. 5.4. Therefore, the execution times reported below should not be taken too strictly, as (i) executing the tools natively on a high-end server would certainly give better numbers, and (ii) the tools, especially Storm, have so many options that we could not try them all, so that we are unsure whether we selected the optimal ones for our experiments.

Experiments with Fig. 3 of Model V1. Modest was found too slow, as it took more than 50 hours to build the state space of the first instance (we halted the experiment). Storm took 100 hours to evaluate the 820 formulas needed for

Fig. 3. At first sight, the results obtained seem close to those reported by [20] for the compound models, but various observations can be made: (i) in some cases, the "min" probabilities are slightly larger than the corresponding "max" probabilities; (ii) all probabilities that the queue contains a large number L of elements remain close to 5.10^{-7} (the default precision of Storm being 10^{-6}), whereas the probabilities computed by CADP for the compound model in LNT decrease from 10^{-33} to 10^{-38} regularly; (iii) the most meaningful cases seem to be $L = 1$ and $L = 2$, where probabilities are not too small, but we observe that the probabilities for the compound model are, with $L = 1$, always smaller than the corresponding "min" probabilities and, with $L = 2$, always larger than the corresponding "max" probabilities. We consider that these results are too uncertain for drawing valid conclusions.

Experiments with Fig. 4 of Model V1. Modest was found too slow: 18 hours to build the state space of the first instance, and more than 6 hours to evaluate the first formula (we halted the experiment). Storm took 12 hours to evaluate the 220 formulas needed for Fig. 4 and produced results that, at first sight, look similar to those of compound models. Yet, in the case $L = 1$, the probability computed by Storm is twice as large (0.03 on compound models vs 0.05 when using Storm on MAs). Like for Fig. 3, the probabilities computed by Storm for larger values of L stagnate around 5.10^{-7} and some "min" probabilities are larger than the corresponding "max" probabilities. We tried without success to vary the precision of calculations and the LP solver used by Storm. Hence, we consider that these results are not conclusive.

Experiments with Fig. 5 of Model V1. On the first five instances, Modest reported floating-point errors and displayed probabilities that were all equal to 1 or very close to 1, e.g., 0.99999999... (we halted the experiment). Storm took 2 hours to produce the 36 "min" and 36 "max" probabilities needed for Fig. 5 — together with those of Fig. 6. These results are plausible: (i) the "min" probabilities are always smaller than the corresponding "max" probabilities; (ii) the "min" and "max" probabilities are very close (the most distant pair being 0.818153 vs 0.818168), so that the differences are not likely to be visible on the picture generated by Gnuplot; (iii) interestingly, the probabilities computed for the compound models [20] are not in the ["min", "max"] interval, as one would expect, but are all above "max"; (iv) yet, they are close to "max" except for the lowest values of β, for which discrepancy becomes visible, the most distant pair being 0.818168 vs 0.858825 (absolute difference: 0.0407, relative difference: 4.97%). Figure 3 compares the results obtained on compound models and MAs.

Experiments with Fig. 6 of Model V1. The observations for Fig. 6 are similar to those made for Fig. 5, given that both experiments have been done together. The main changes are: (i) the discrepancy between the "min" and "max" probabilities computed by Storm is larger, the most distant pair being

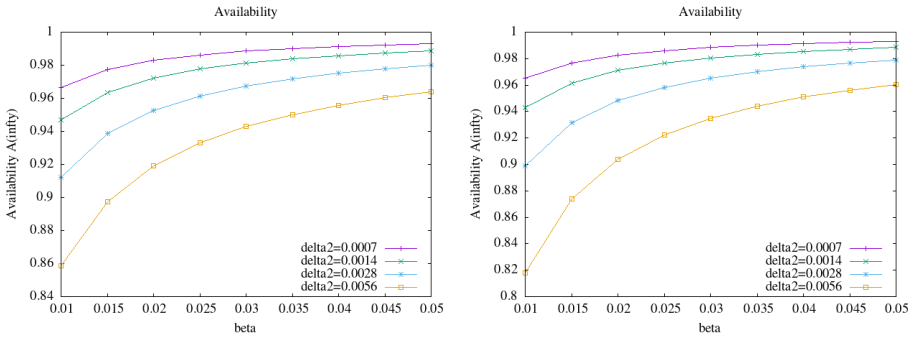

Fig. 3. Figure 5 generated by CADP for the compound model (left) and by Storm for the IMC-MA of model V1 (right)

0.440619 vs 0.441169; (ii) the discrepancy between the "max" probabilities and the probabilities computed for the compound models is also larger, the most distant pair being 0.441169 vs 0.474733 (absolute difference: 0.0336, relative difference: 7.61%). Nevertheless, the shapes of the curves remain comparable, as shown by Fig. 4.

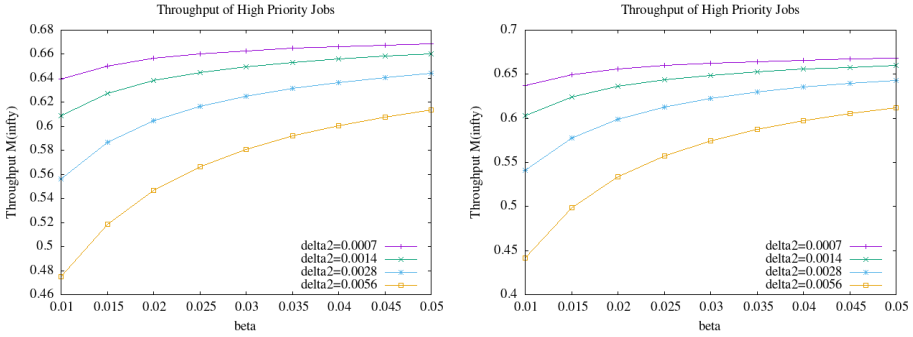

Fig. 4. Figure 6 generated by CADP for the compound model (left) and by Storm for the IMC-MA of model V1 (right)

Thus, IMC modelling and Markov-automata tools can be used to describe a complex system such as the Erlangen mainframe and, when the state space is not too large, produce results close to those obtained with compound models.

6.5 Results of Transient Analyses

We experimented with Modest and Storm in order to obtain the transient probabilities needed for Fig. 7–10. Our experiments were similar to those of Sect. 6.4,

but the properties to evaluate were meant to compute state probabilities or event throughputs at particular instants in time $t = 1$, $t = 100$, $t = 200$, ..., and $t = 1500$.

Modest emitted, for each property π, a warning that π was unsupported and skipped it.

Storm warned that it would use the IMCA method rather the Unif+ method and announced, for the first property ($t = 100$) of Fig. 7, that it would perform 106,819,632,945,608 iterations. For the last property ($t = 1500$), the number of iterations was even 225 times larger. Of course, we halted the execution, which had not converged after 8 hours. We then tried to lower the accuracy of results by setting Storm's time-bounded precision option to 0.1: this decreased to 1,068,196,330 the number of iterations planned for $t = 100$, but this reduction was still not sufficient and we had to halt the execution after 18 hours.

Observing that Modest and Storm fail to compute transient "time-point" properties of the form $t = k$ (where k is a constant) when the Markov automaton gets large, and knowing that they better handle transient "time-bounded" properties of the form $t \leq k$, we tried to compute the former using the latter by expressing $p(t = k)$ either as $p(t \leq k) - p(t < k)$ or as $\big(p(t \leq k+\epsilon) - p(t \leq k-\epsilon)\big)/2\epsilon$, but these attempts did not succeed.

6.6 Possible Enhancements of Analysis Tools

The availability of Modest and Storm to analyse Markov automata is, in itself, a great achievement, considering the rare combination of theoretical breakthroughs and skilful development over many years that was needed to produce these tools. Clearly, our experiments with the Erlangen mainframe are pushing these tools to their current limits, but we are confident that these could be overcome in a near future. To this aim, we can make five suggestions:

1. The main limitation arising from our experiments is the current impossibility of obtaining transient time-point probabilities for Fig. 7–10. On the one hand, Storm's early algorithm [53] for computing these probabilities, which is based on discretisation and inherited from the IMCA tool [27], faces problems scaling to large MAs. On the other hand, the Unif+ algorithm [11] [26], which is implemented in both Modest and Storm, scales much better but only supports time-bounded properties. Extending Unif+ to address time-point properties would be desirable. A discussion in [8] (just before Lemma 2) suggests that this could be done by omitting the initial preprocessing step that turns all target states into absorbing states, prior to the main Unif+ computation.
2. Most of the complexity of the translators from IMCs to Modest and Storm (see Sect. 6.2) lies in the conversions of probes to state variables. Such conversions (and their consecutive introduction of additional probe variables in the models) could be avoided if the property language of Modest and the state labels of Storm and PRISM, which, at present, only refer to state variables, would also refer to transitions. We suggest to extend their syntax with

a predicate *enable*(a) that is true for all states having an outgoing transition labelled with a. This is an old idea [52], which is easy to implement. This way, the states in which the mainframe is available could be directly characterised as *enable*(*"z_avail"*). One could also introduce predicates, e.g., *enable*($z_prog_queue(n)$ *where* $n < 2$), but this is not needed for the Erlangen mainframe and such properties may often be expressed in a different manner, e.g., *enable*(*"z_prog_queue_0"*) $\vee$ *enable*(*"z_prog_queue_1"*).

3. Although the various instances of model V1 produced by our translators have a simple, explicit-state structure and are not so large, Modest and Storm are surprisingly slow in parsing them and building their state spaces. Indeed, the time spent grows more than linearly as the number of states increases. On the three sizes of instances shown in Tab. 6 for Fig. 5, 4, and 3 (namely 8,560, 43,708, and 74,800 states — the corresponding numbers of transitions being in a linear ratio, i.e., 5.2–5.5 times larger), Modest takes more than 15 seconds, 18 hours, and 50 hours, respectively, while Storm takes more than 25 seconds, 10 minutes, and 29 minutes, respectively. A similar observation can be made with PRISM on model V4 (see Sect. 7.8), which, given CTMCs having 9,972, 53,640, and 90,396 states, takes more than 10 seconds, 6 minutes, and 17 minutes, respectively. This suggests that the tools are currently not able to recognise explicit-state models and handle them linearly, and that the user should help them to do so. This could easily be implemented by a command-line option or a pragma in the input file to inform the tool that the model, which should contain no parallel composition, is an explicit-state one, with an indication of the particular variable s chosen to store the current state.
4. We observed that the time taken by Storm on model V1 to compute each of the 800 "min"/"max" probabilities of Fig. 3 is always larger than 300 seconds per property. For the 200 "min"/"max" probabilities of Fig. 4, this time is always larger than 141 seconds. Similarly, PRISM takes on model V4 at least 21 seconds per property of Fig. 3 and at least 14 seconds per property of Fig. 4. This suggests that these tools just evaluate properties as they come, one after the other, without examining the set of properties as a whole to avoid redundant calculations. However, the properties of Fig. 3 (resp. 4) are all alike: each of them computes the steady-state probability for a given subset of states, the collection of these subsets forming a partition of the entire state space. Therefore, computing the stationary solution vector first (the tools have algorithms for this) would allow to answer all the properties at once. Alternatively, the tools could implement a simple form of memoisation by storing in a cache the stationary solution vector computed for the first property and reusing it for the next ones; this way, evaluating the first property would take time, whereas the evaluation of subsequent properties would be instantaneous.
5. Finally, the output format of the tools should be enhanced. At present, this format (visibly inspired by the output of the PRISM tool) consists in human-readable text, in which the essential results (probabilities and throughputs) come together with a lot of auxiliary information about progress of calcula-

tions, models and properties analysed, size of state spaces, timing, etc. To produce the eight figures of the Erlangen mainframe, 778 probabilities and throughputs must be evaluated (see Tab. 5); this number is even multiplied by two when dealing with Markov automata, since one computes both the minimal and maximal probabilities. To produce the eight figures using Gnuplot, each of these numerical results must be dispatched to a precise file, where it must be inserted in a matrix, at a precise line and column. Such a task must be done by the user, since it is really application-specific and beyond the mission of tools for Markov automata. To do so, we developed five scripts (160 lines of Bourne shell) that make intensive use of Awk. This proved to be a time-consuming activity, since it required to finely parse the output files produced by the tools. We noticed that the formats of these files are not convenient for automated exploitation, since the name μ of the model being analysed, the name π of the property being evaluated, and the numerical result p all appear on different lines, intertwined with other information. Gathering all important information on the same line (e.g., "Result of property π on model μ is p") would enable users to exploit the results faster and more easily.

7 Deterministic Analyses

7.1 Removing Nondeterminism

Considering the results of the two analyses that succeeded using tools for Markov automata, the "min" and "max" probabilities are very close, with a largest difference of 1.5×10^{-5} for Fig. 5 and 5.5×10^{-4} for Fig. 6. One may wonder whether nondeterminism is really meaningful here, and worth the difficulties it causes, when its impact on the final results seems negligible.

Taking this line of thought further, would it be possible to slightly modify the IMC model in order to eliminate nondeterminism (and, thus, to obtain a CTMC instead of a MA), still preserving the properties needed to build the eight figures?

From a conceptual point of view, the presence of nondeterminism in a formal model, unless it was purposely intended by the specifier, raises questions. In the IMC models of the Erlangen mainframe, nondeterminism was surely not intended, and its causes were totally unclear when we undertook the present study. In such case, one hesitates between two possible explanations. Is nondeterminism simply an artefact of the standard translation (see Sect. 2.7)? Or is it rather the consequence of specification mistakes that were originally present in the compound model, in some latent form that compound semantics did not reveal, but that IMCs, being a more demanding formalism, bring to light?

From a practical point of view, nondeterminism is currently a nuisance for the analysis, so that replacing MAs with CTMCs would offer many advantages: (i) CTMCs deliver exact probabilities while MAs only produce intervals of "min" and "max" probabilities; (ii) there are many more tools for CTMCs than tools

for MAs; (iii) algorithms for solving CTMCs are older, simpler, and probably faster than those for MAs.

We conjecture that a model having a nondeterministic IMC can be turned into another model whose IMC is a CTMC, by using step-by-step transformations that progressively eliminate all occurrences of nondeterminism and should, hopefully, preserve the numerical solutions (probabilities and throughputs) of the original model.

We additionally require that each transformation from a model M to a model M' ensures that the LTS of M' is (strictly) included in the LTS of M according to the strong bisimulation preorder, which we denote $M' \sqsubseteq M$. This condition means: (i) that M' does only a part of what M can do, with the expectation that some nondeterministic parts of M are absent from M', and (ii) that everything M' can do is already doable by M, i.e., M' does not introduce novel possibilities.

We now present two such transformations, which we then apply to the Erlangen mainframe, starting from model V1 (see Sect. 5.1) to produce successive models V2, V3, and V4, from which nondeterminism is gradually eliminated.

7.2 Removal of Nondeterminism by Enforcing Atomicity

The presence of nondeterminism in many IMC models, including the Erlangen mainframe, arises from broken atomicity (see Sect. 2.7), i.e., the presence of concurrent sequences "$\lambda; a$" and "$\mu; b$", where λ and μ occur first, then a and b get blocked due to impossible synchronisations with a third process, before being unlocked simultaneously by the third process.

One way of avoiding such nondeterminism between a and b is to relax the synchronisation patterns to prevent a (resp. b) from being both blocked by the third party. We will not use this approach, as it violates our requirement of preserving inclusion between LTSs for the strong bisimulation preorder — indeed, the transformed model would have states in which a (resp. b) is enabled, whereas it is not enabled in the corresponding states of the original model.

Notice that it is not mandatory to address both concurrent sequences "$\lambda; a$" and "$\mu; b$": addressing only one of them might be sufficient, in some cases, to remove nondeterminism.

Another way of avoiding nondeterminism is to delay the occurrence of λ (resp. μ) as long as there is a risk that a (resp. b) becomes blocked afterwards. This requires either that (i) the process(es) executing "$\lambda; a$" (resp. "$\mu; b$") must observe the system globally to predict situations in which a (resp. b) is likely to be blocked, or that (ii) an additional *bouncer* process is added to the system, with the capability of blocking λ (resp. μ) in certain situations and unlocking them at other moments.

Solution (ii) automatically guarantees the preorder inclusion $M' \sqsubseteq M$ if M denotes the original model and if the transformed model M' has the form $M' := M \parallel_L B$, where B is a bouncer that does not contain τ-transitions, and where L is the set of all transition labels in B, including (symbolic) rate transitions. Notice that synchronising concurrent processes on rate transitions violates a tenet of IMCs, but this is a meaningful derogation from this principle, especially

when solution (ii) can be expressed equivalently using solution (i), which only requires synchronisations on non-rate transitions — a detailed example is shown in Sect. 7.6 below.

Globally, all these approaches aim at enforcing atomicity, in the sense they try to ensure that either λ and a (resp. μ and b) will occur "together" (i.e., without too long interval in between), or none of them will.

7.3 Removal of Nondeterminism by Enforcing Chronology

In the presence of broken atomicity for two concurrent sequences "$\lambda; a$" and "$\mu; b$", there is an alternative approach to those preserving atomicity. Rather than delaying or blocking λ and/or μ to prevent posterior nondeterminism between a and b, one may let λ and μ happen without restriction, and wait until the point of nondeterminism to resolve it, by preventing the occurrence of either a or b.

Obviously, the choice between a and b cannot be decided using fixed priorities (e.g., $a \gg b$), otherwise it would be unfair. We instead propose the following criterion based on chronology: if λ occurred first, then a will be enabled and b blocked, whereas if μ occurred first, then a will be blocked and b enabled.

This is implemented by adding to the system an additional *resolver* process, which synchronises on the transitions λ, μ, a, and b. This process records the chronology of λ and μ in its internal state and uses this information to decide between a and b.

Example 9. For instance, the model $M := (\lambda; a; z_a^\bullet \parallel \mu; b; z_b^\bullet) \parallel_{\{a,b\}} \nu; (a \,\Box\, b)$, where $z_a^\bullet$ and $z_b^\bullet$ are two self-loop probes, generates an IMC that is nondeterministic due to a choice between a and b. Among various possibilities, an effective resolver for M is $R := \lambda; (a \parallel \mu) \,\Box\, \mu; (b \parallel \lambda)$. Indeed, the parallel composition $M' := (M \parallel_{\{\lambda,\mu,a,b\}} R)$ generates an IMC that has the same number of states as the IMC of M but one transition less (actually, the two τ-transitions are replaced by one rate transition ν). ■

Again, if R is a resolver that contains no τ-transition and if L is the set of all transition labels in R, including rate transitions, then defining $M' := M \parallel_L R$ ensures that $M' \sqsubseteq M$. Again, synchronising M and R on rates goes against an established principle of IMCs, but notice that any meaningful resolver should only observe rate transitions and never block them.

7.4 Tools for Understanding Nondeterminism

Explaining the presence of nondeterminism is the dual problem of explaining the presence of deadlocks caused by synchronisation mistakes. Such deadlocks arise when two events are not present as expected at the same instant. Nondeterminism arises when two events are unexpectedly present at the same instant. Thus, the difficulty is similar in both cases, although nondeterminism is perhaps simpler, as it is easier to study the presence of an event rather than its absence.

Yet, in the case of IMCs, an extra difficulty comes from the fact that all visible transitions have been hidden.

Given the difficulty of analysing IMCs (see Sect. 2.8) and the large sizes of IMC instances (see Tab. 6), visual inspection would not be feasible. So, we reused various software components of CADP to develop a *debugging toolchain* that automatically locates and explains nondeterminism in IMCs. This toolchain works in successive steps described as follows:

1. It first generates the LTS and the IMC corresponding to a given instance of a model (see Sect. 5).
2. It identifies, using an XTL script, all *conflict states* in the IMC, i.e., all states having outgoing τ-transitions.
3. It computes, using the BCG_INFO tool, the shortest paths leading to each of these conflict states in the IMC. Each of these paths is a sequence of transitions labelled with concrete rates.
4. It transforms these paths by replacing all concrete rates with symbolic rates, e.,g., 0.00334 by φ, 0.01667 by λ_1, etc. We call *rate sequences* the resulting sequences of symbolic rates, dropping out all information about states.
5. It detects and removes, using the FDUPES[27] tool, duplicated rate sequences, i.e., identical rate sequences leading to different conflict states.
6. It converts, using the EXHIBITOR[28] tool, each rate sequence into one or many corresponding sequences in the LTS, by retrieving and inserting the visible events that have been hidden and then abstracted away by stochastic branching minimisation. We call *full sequences* the resulting sequences, each of which is such that, by keeping only its symbolic rates, one obtains the rate sequence it was generated from.
7. Finally, it orders full sequences by increasing length, with the expectation that a user wanting to remove nondeterminism will first study the shortest sequences, which are easier to replay and analyse.

7.5 Understanding Nondeterminism in Model V1

We applied our debugging toolchain to model V1, which was obtained by applying the standard translation (see Sect. 2.3) to the compound model of the Erlangen mainframe [20]. For our experiments, we chose one instance among those needed to produce Fig. 5 and 6, because these are the smallest instances of model V1 (the other instances differing mostly because of larger queue sizes) and because Storm successfully handles these instances.

The toolchain reports the existence of 552 conflict states. All the 552 rate sequences leading to these states (there are no duplicate sequences) end with a rate transition β, which suggests that the remaining nondeterminism arises from failures and repairs. The two shortest rate sequences (of length 4) are s_1 := "$\delta_1; \lambda_1; \delta_1; \beta$" and s_2 := "$\delta_1; \mu_1; \delta_1; \beta$". The full sequence corresponding

[27] https://github.com/adrianlopezroche/fdupes
[28] https://cadp.inria.fr/man/exhibitor.html

to s_1 is "$\delta_1; \mathit{fail}; \lambda_1; \delta_1; \beta; \mathit{repair}; (\mathit{fail} \,\square\, \mathit{prog_job})$", of which the full sequence corresponding to s_2 is dual, by replacing λ_1 with μ_1 and *prog_job* with *user_job*.

The scenario of sequence s_1 can be explained as follows: while the mainframe is in its initial state, the *fail_load* process triggers a failure (δ_1 followed by *fail*). The *prog_load* process initiates a new job (λ_1) but the next event *prog_job* is blocked by the *fail_queue* process. The *fail_load* process then initiates another failure (δ_1), but the next event *fail* is also blocked by the *fail_queue* process. Eventually, the mainframe restarts (β followed by *repair*). The *fail_queue* process resumes, which simultaneously unlocks both pending events *prog_job* and *fail*. When all events are hidden, this creates a conflict state with two outgoing τ-transitions.

7.6 Model V2 of the Erlangen Mainframe

The analysis of nondeterminism in model V1 shows that the *fail_load* process may initiate a new failure by spending rate δ_1 (resp. δ_2) while the mainframe is already failed. This questionable scenario cannot happen in the compound model, where δ_1 and δ_2 are, each, glued with *fail* and, thus, cannot occur before the next *repair*. This is clearly a broken atomicity issue, caused by the splitting of compound transitions $(\mathit{fail}, \delta_1)$ and $(\mathit{fail}, \delta_2)$.

To address the issue, we tried various modifications of model V1 in order to enforce atomicity (see Sect. 7.2) or chronology (see Sect. 7.3), before concluding that atomicity would be more effective at this point. We thus present two possible modifications of model V1 that, respectively, illustrate the approaches (i) and (ii) mentioned in Sect. 7.2.

Our first approach consists in modifying the *fail_load* process so that it does not initiate failures while the mainframe is already failed, which means that, in such case, *fail_load* has to wait until the next *repair* before spending δ_1 or δ_2. Consequently, model V1 must be modified to allow *fail_load* to synchronise on event *repair* — this restores some broken symmetry, considering that *fail* was involved in six-party synchronisations while *repair* was involved in only five-party ones (see Fig. 1). Process *fail_load* itself must also be deeply modified (see its LNT code on the left of Fig. 5). This process, which already slightly differed from the two other processes *prog_load* and *user_load* due to the Boolean parameter *delayed* (see Sect. 4.2), becomes even more different with the introduction of *repair* as a new event parameter. The behaviour of *fail_load* is further modified to ensure that, between *fail* and *repair*, events c (which express phase changes) are not blocked. Conversely, the process must also accept a potential occurrence of *repair* between φ and c, not to cause deadlocks. This first approach does not synchronise on rate transitions, thus respecting an IMC principle, but it is involved and error-prone.

Our second approach is much simpler: the *fail_load* process of model V1 is kept unchanged, while the mainframe is composed in parallel with a small bouncer (see its LNT code at the top right of Fig. 5) that prevents rates δ_1 and δ_2 from occurring between *fail* and *repair*. The mainframe and the bouncer synchronise on δ_1, δ_2, *fail*, and *repair*. We checked, using the BCG_CMP tool,

that the LTSs generated by both approaches are strongly bisimilar. Hence, we adopt as model V2 the LNT program of the second approach and will use it as a basis for the next steps.

```
process FAIL_LOAD [...] (in var P: PHASE,
                         in var FAILED: bool) is
   loop
      alt
         if FAILED then
            REPAIR;
            FAILED := false
         else
            case P in
               1 ->
                  -- low-load phase
                  DELTA1;
                  FAIL;
                  FAILED := true
            |  2 ->
                  -- high-load phase
                  DELTA2;
                  FAIL;
                  FAILED := true
            |  3 ->
                  -- idle phase
                  null
            end case
         end if
      []
         PHI;
         if FAILED then
            alt
               null
            []
               REPAIR;
               FAILED := false
            end alt
         end if;
         C;
         case P in
            1 -> P := 2
         |  2 -> P := 3
         |  3 -> P := 1
         end case
      end alt
   end loop
end process
```

```
process BOUNCER [...] is
   loop
      alt
         DELTA1
      []
         DELTA2
      []
         FAIL;
         REPAIR
      end alt
   end loop
end process
```

```
process BOUNCER [...] is
   loop
      alt
         DELTA1
      []
         DELTA2
      []
         MU1
      []
         MU2
      []
         FAIL;
         REPAIR
      end alt
   end loop
end process
```

Fig. 5. LNT model fragments documenting the changes in models V2 and V3

Experiments with model V2 give encouraging results: (i) we checked, using the BISIMULATOR[29] tool, that V2 $\sqsubseteq$ V1; (ii) the number of conflict states drops from 552 in model V1 to 56 in model V2; (iii) the discrepancy between the "min" and "max" probabilities computed by Storm on the MAs of model V2 is at most 4.10^{-9} for Fig. 5 and 4.10^{-8} for Fig. 6; (iv) the discrepancy

[29] https://cadp.inria.fr/man/bisimulator.html

between the probabilities of the compound model and those computed by Storm on model V2 is smaller than with model V1 (maximal relative differences: 0.39% for Fig. 5 and 2.19% for Fig. 6).

7.7 Model V3 of the Erlangen Mainframe

We applied our debugging toolchain to model V2 to better know about the 56 conflict states. All the 56 rate sequences leading to these states (there are no duplicate sequences) end with a rate transition β. The shortest rate sequence (of length 5) is "$\lambda_1; \delta_1; \mu_1; \alpha_1; \beta$" and the corresponding full sequence is "$\lambda_1; \mathit{prog_job}; \delta_1; \mathit{fail}; \mu_1; \alpha_1; \beta; \mathit{repair}; (\mathit{user_job} \,\Box\, \mathit{get_prog_job})$". The scenario is the following: the *prog_load* process sends a job to the *prog_queue* (λ_1 followed by *prog_job*). The *fail_load* process triggers a failure (δ_1 followed by *fail*). While the mainframe is in failed state, the *user_load* initiates a job (μ_1), but the next event *user_job* is blocked by the *fail_queue* process. The *prog_queue* initiates a job transfer to the processors (α_1), but the next event *get_prog_job* is also blocked by the failure. Eventually, the mainframe restarts (β followed by *repair*). The *fail_queue* unlocks the inputs and outputs of the two other queues, simultaneously enabling both pending events *user_job* and *get_prog_job*, hence causing nondeterminism.

The root cause of the problem is that, in the compound model, only (c, φ) transitions may occur between $(\mathit{fail}, \delta_i)$ and (repair, β), i.e., while the mainframe is in failed state. But, in the IMC model, many other transitions may happen between *fail* and *repair*: occurrences of c, β, and φ are normal; occurrences of δ_1 and δ_2 are already prohibited by the bouncer of model V2; occurrences of α_i, λ_i, μ_i, etc. should be examined carefully, as they might break atomicity and introduce nondeterminism.

We first extended the bouncer of model V2 to block, beyond δ_1 and δ_2, all rate transitions α_1, α_2, λ_1, λ_2, μ_1, μ_2, ν, and ξ when the mainframe is in failed state, so as to replicate the behaviour of the compound model in this situation. Using this bouncer, the number of conflict states was reduced from 56 to 28 and Storm produced a Fig. 5 similar to that of model V2. However, Storm also produced a Fig. 6 radically different from the one of model V2 (see Fig. 6 for a comparative view), making it clear that, if this bouncer is effective in further removing nondeterminism, it is overly restrictive concerning synchronisations.

Undertaking a systematic study, we tried to remove one by one the rate transitions α_1, α_2, λ_1, λ_2, μ_1, μ_2, ν, and ξ from the bouncer, while watching the outcome of such removals on the number of conflict states. Changes were only observed with μ_1 and μ_2, the other rates having no impact. When removing either μ_1 or μ_2, the number of conflict states was reduced from 56 to 44. When removing both μ_1 and μ_2, it dropped down to 28. Consequently, we opted for a new model V3 derived from model V2 by adding μ_1 and μ_2 (together with δ_1 and δ_2) to the bouncer (see the LNT code at the bottom right of Fig. 5) and to the list of rates on which the mainframe and the bouncer synchronise.

Our experiments with this model V3 gave results corroborating this decision: (i) we checked, using the BISIMULATOR tool, that V3 $\sqsubseteq$ V2; (ii) the discrep-

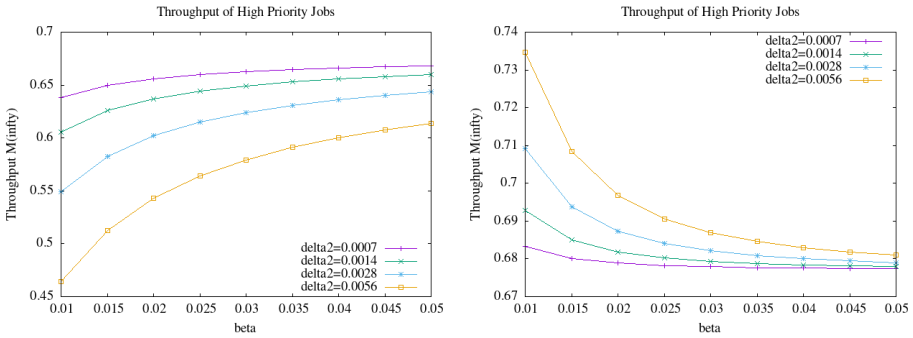

Fig. 6. Figure 6 for model V2 (left) and for model V2 equipped with a restrictive bouncer blocking all rate transitions but β and φ (right)

ancy between the "min" and "max" probabilities computed by Storm on the MAs of model V3 is at most 2.10^{-10} for Fig. 5 and 2.10^{-9} for Fig. 6; (iii) the discrepancy between the probabilities of the compound model and those computed by Storm on model V3 is comparable to that of model V2 (maximal relative differences: 0.37% for Fig. 5 and 2.37% for Fig. 6).

7.8 Model V4 of the Erlangen Mainframe

Again, we applied our debugging toolchain to understand the 28 conflict states in model V3. Again, all the 28 rate sequences leading to these states terminate with a rate transition β. The shortest rate sequence (of length 15) is[30] "$(\lambda_1;\alpha_1)^5;\mu_1;\alpha_2;\mu_1;\delta_1;\beta$", where s^n means n consecutive repetitions of the sub-sequence s. The corresponding full sequence (of length 29) is "$(\lambda_1;\mathit{prog_job};\alpha_1;\mathit{get_prog_job})^4;\lambda_1;\mathit{prog_job};\alpha_1;\mu_1;\mathit{user_job};\alpha_2;\mu_1;\delta_1;\mathit{fail};\beta;\mathit{repair};\mathit{get_user_job};(\mathit{get_prog_job}\,\Box\,\mathit{user_job})$". The scenario can be explained as follows: the *prog_load* process sends four jobs, which are received by the *prog_queue* and delivered to the four processors. The *prog_load* process sends a fifth job to the *prog_queue*, which initiates a transfer to the processors (α_1) that cannot complete since all processors are busy. The *user_load* does the same, sending a job to the *user_queue*, which also initiates a transfer (α_2) that cannot complete. The *user_load* initiates another job (μ_1). The *fail_load* process triggers a failure (δ_1) that does not last long, since the mainframe gets immediately repaired within the expected delay (β). When the mainframe restarts, the four jobs formerly running on the processors are cleared, so that each processor can now accept a new job. The job stored in the *user_queue* is taken first, because of the priority mechanism. This queue becomes empty, so that the lower-priority job stored in the *prog_queue* becomes eligible. At the same time, the *user_queue*

[30] For the clarity of this section, we replace the sequences found by our toolchain by "equivalent" sequences having the same lengths, but simpler to explain.

may receive the pending job sent by the *user_load*. Hence, nondeterminism ensues between the output of the *prog_queue* and the input of the *user_queue*.

We did all kinds of attempts to eliminate such nondeterminism from model V3. For instance, if rate transitions α_2 (or both α_1 and α_2) are removed from model V3, based on the assumption that the corresponding numeric rates are so large that these transitions can be considered as immediate, the number of conflict states drops from 28 to zero (interestingly, if rate transitions α_1 only are removed, the number of conflict states increases from 28 to 37). Yet, we discarded this idea, as the resulting model M does not satisfy $M \sqsubseteq V3$ and the figures 3–10 obtained were much too different from the expected ones.

We tried hard to reuse our atomicity-preserving approach that had been fruitfully applied to reduce nondeterminism in models V1 and V2, but we had no success with model V3. Indeed, the *prog_load* and *user_load* processes are independent, and the *prog_queue* and the *user_queue* processes are only weakly correlated. Due to the complexity of scenarios leading to conflict states (their length ranges between 29 and 39), it would be difficult for the *user_load* or *prog_queue* processes to acquire a global knowledge of the system precise enough to decide whether transferring a job to the *user_queue* or from the *prog_queue* may cause nondeterminism.

In such a case, our chronology-preserving approach seems more appropriate for avoiding, at the very last moment, nondeterminism between far-remote events, a problem that would otherwise be difficult to anticipate and prevent by blocking rate transitions in advance. We designed a new model V4 obtained by equipping model V3 with an additional resolver process. Given that the conflict is between *get_prog_job* and *user_job*, model V3 (i.e., the mainframe and its bouncer) is synchronised with the resolver on these two events and on their associated rates α_1, μ_1, and μ_2. The resolver (see its LNT code in Fig. 7) maintains an internal FIFO queue that stores the arrival order of α_1, on the one hand, and μ_1 or μ_2, on the other hand. Depending of this order, the resolver enables either *get_prog_job* or *user_job*. We observed that, in any reachable state of the LTSs, this FIFO queue contains at most two elements. Notice that this resolver retrospectively gives one more justification to our decision of splitting α into α_1 and α_2 (see Sect. 4.3), as the resolver would not work with nondeterministic choices "α; *get_prog_job* $\Box$ α; *get_user_job*".

Experiments with model V4 gave the following results: (i) we checked, using the BISIMULATOR tool, that V4 $\sqsubseteq$ V3; (ii) the number of conflict states in model V4 is zero, meaning that all instances of this model are CTMCs and can be analysed using tools such as CADP or PRISM; (iii) the discrepancy between the probabilities of the compound model and those computed by BCG_STEADY on model V4 are nearly identical to those observed with Storm on model V3 (maximal relative differences: 0.37% for Fig. 5 and 2.38% for Fig. 6); (iv) we checked that BCG_STEADY and Storm produce similar results for these CTMCs (maximal absolute differences: 3.10^{-5} for Fig. 5 and 3.10^{-6} for Fig. 6); (v) noticing that a CTMC in PRISM-MA format can easily be converted to PRISM format by replacing "`ma`" by "`ctmc`" and all symbols "`<>`" by "`[]`",

```
type FIFO is
    list of char
    with head, tail,
         length, append
end type
```

```
process RESOLVER [...] is
   var F: FIFO in
      F := {};
      loop
         assert length (F) <= 2;
         alt
            ALPHA1;
            F := append ('a', F)
         []
            MU1;
            F := append ('m', F)
         []
            MU2;
            F := append ('m', F)
         []
            GET_PROG_JOB where head (F) = 'a';
            F := tail (F)
         []
            USER_JOB where head (F) = 'm';
            F := tail (F)
         end alt
      end loop
   end var
end process
```

Fig. 7. LNT model fragments documenting the changes in model V4

we checked that BCG_STEADY and PRISM produce similar results (maximal absolute differences: 8.10^{-7} for Fig. 5 and 9.10^{-6} for Fig. 6).

7.9 Assessment of Deterministic Analyses

An overview of the experiments with our four successive models V1, V2, V3, and V4 is given in Tab. 7. As mentioned before, the LTSs of these models are included in each other according to the strong bisimulation preorder, i.e., V4 $\sqsubseteq$ V3 $\sqsubseteq$ V2 $\sqsubseteq$ V1. Notice that $M' \sqsubseteq M$ does not imply that the LTS of M' has fewer states or transitions than the LTS of M (consider, e.g., $M := a^{\bullet}$ and $M' := a; a; a$). From the 4th and 5th columns of Tab. 7, one may conjecture that bouncers decrease the numbers of states and transitions in the generated CTMC, while resolvers increase them. The 6th and 7th column show the discrepancy (maximal absolute difference and maximal relative difference) for Fig. 5 and 6 between the compound model and our models V1–V4. Although there are too few values for drawing firm conclusions, these columns suggest that progressively removing nondeterminism while maintaining LTS inclusion preserves the steady-state probabilities and throughputs of Fig. 5 and 6. This was confirmed in contrast by other experiments that exhibited larger discrepancies when applying transformations that did not maintain LTS inclusion.

Given that the instances of model V4 are CTMCs, one can now use CADP, PRISM, and/or Storm to generate the other figures than Fig. 5–6 and compare them with those produced for the compound model:

model	conflicts	inclusion	states CTMC	transitions CTMC	max.difference for Fig. 5	max.difference for Fig. 6
V1	552	—	8,560	36,427	0.0407 (4.97%)	0.0336 (7.61%)
V2	56	V2 $\sqsubseteq$ V1	7,416	32,803	0.00335 (0.39%)	0.0104 (2.19%)
V3	28	V3 $\sqsubseteq$ V2	7,024	31,195	0.0032 (0.37%)	0.0113 (2.37%)
V4	0	V4 $\sqsubseteq$ V3	9,972	41,619	0.0032 (0.37%)	0.0113 (2.38%)

Table 7. Overview of results obtained with models V1, V2, V3, and V4

- Concerning Fig. 3 and 4, the maximal differences between the compound model and model V4 are small in absolute values (9.10^{-4} for Fig. 3 and 3.10^{-2} for Fig. 4) but large enough in relative values (172% for Fig. 3 and 221% for Fig. 4) to produce visible differences that can be observed in Fig. 8 and 9. The right parts of these figures (those for model V4) have been computed independently using CADP, PRISM, and Storm, which all give nearly identical results (maximal absolute differences between CADP and PRISM: 1.10^{-8} for Fig. 3 and 4.10^{-7} for Fig. 4, and between CADP and Storm: 9.10^{-6} for Fig. 3 and 3.10^{-5} for Fig. 4).
- The four remaining figures obtained from model V4 are very close to those of the compound model (maximal relative differences: 0.014% for Fig. 7, 0.43% for Fig. 8, 0.015% for Fig. 9, and 1.70% for Fig. 10). Again, CADP, PRISM, and Storm give close results on these four figures.

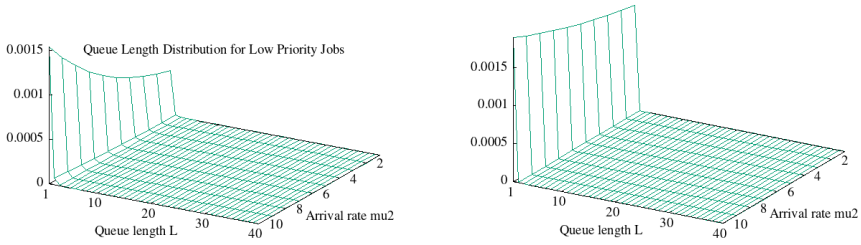

Fig. 8. Figure 3 for the compound model (left) and for model V4 (right)

Thus, our deterministic analysis approach, which is based on the progressive removal of nondeterminism through successive transformations, has two merits:

- The final model V4 is simply the initial model V1 composed in parallel with two small processes, a bouncer and a resolver. No involved modification of model V1 was needed to produce model V4.

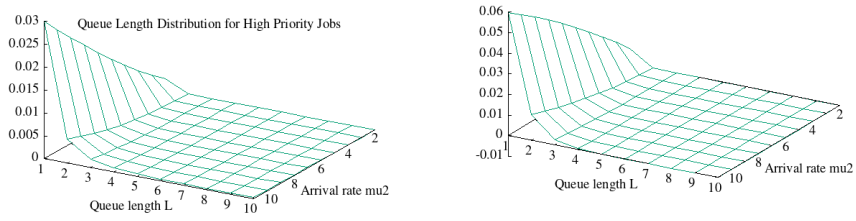

Fig. 9. Figure 4 for the compound model (left) and for model V4 (right)

– Despite the fact that standard translation does not preserve steady-state and transient probabilities in the general case (see Sect. 2.4), it was eventually possible to reproduce quite closely the eight figures of the Erlangen mainframe — except a few probabilities in Fig. 3 and 4 that visibly differ by their relative values.

So, transforming progressively a nondeterministic IMC into a CTMC seems promising. But one may wonder whether this approach is truly a methodology, in the sense that it can be reused and applied by others to solve similar problems. The results obtained should not hide the efforts and difficulties:

– There are many ways to transform a model. Our debugging toolchain may suggest plausible directions, but deciding which changes should be made requires a deep understanding of the system under analysis, as well as intuition, intense brainstorming, and lively discussions.
– This is a trial-and-error approach. Even if a change seems rational and promising, it is difficult to predict whether it will be effective in removing nondeterminism while preserving the expected numerical results.
– Designing bouncers and resolvers is a narrow path. If these processes are too permissive, they will not reduce nondeterminism. If they are too restrictive (e.g., bouncers synchronising certain transitions without necessity, resolvers blocking unforeseen rate transitions, etc.), they may introduce deadlocks or, worse, slow down the system in a way that alters the expected throughputs.
– To find our way in the search space, we used a combination of three guidelines: (i) the number of conflict states, which had to decrease strictly after each transformation; (ii) the inclusion of LTSs, which had to be preserved; and (iii) the probabilities and throughputs computed for Fig. 5 and 6 of the compound model, which we used as a trigger warning to discard those transformations that would diverge too much from these numbers.

8 Conclusion

The present article explored the hidden side of IMCs to get new insights about this formalism. In a nutshell, our contributions fall into three categories:

- We provided a stepwise methodology explaining how IMCs can be applied to an involved example such as the Erlangen mainframe. We set out the concept of "rate tagging", which has been used in most case studies so far, but never documented in theoretical papers. We introduced the concepts of "state probes" and "event probes", which are required to express state probabilities and throughput measures on IMCs. We exposed the inherent difficulties that may arise when using IMCs (proper placement of probes, larger state spaces, unexpected nondeterminism) and indicated solutions to these problems, thus paving the way for a wider dissemination of IMCs.
- We progressed the comparison between compound models and IMCs by assessing, for the first time, both approaches on a common case study through the prism of our "standard translation" function that converts compound models to IMCs. We established that, in the general case, this translation preserves neither the alternation of events and rates, nor the absence of deadlocks, nor the determinism property, nor the steady-state and transient probabilities. We however observed that this same translation, when applied to the Erlangen mainframe, preserved six figures out of eight, two other figures being quite similar except for corner-case values. Our experiments also indicate that, although IMCs have been designed for systems with only a few rates, they can also be used for compound-like models, in which (almost) every event has a rate.
- We studied how to handle nondeterminism, an issue not considered so far in publications related to IMCs. We proposed two different approaches: either (i) accepting nondeterminism as a fact, converting nondeterministic IMCs to Markov automata, and using state-of-the-art tools for Markov automata, or (ii) progressively eliminating nondeterminism by means of IMC transformations that preserve either atomicity ("bouncers") or chronology ("resolvers"), still enforcing model inclusion according to the strong bisimulation preorder.

This work could be pursued in three different directions:

- This study would not have been possible without the availability of advanced software tools, such as CADP, PRISM, Modest, and Storm. Our experiments sometimes pushed these tools to their current limits, which led us to suggest (see Sect. 6.6) five possible enhancements to these tools: two about the expressiveness of quantitative properties, two about the performance (speed) of model checking, and one about the user-friendliness of output formats.
- The ability to specify nondeterministic choices gives IMCs a greater expressiveness than compound models. Yet, in our study, nondeterminism was rather a nuisance than a desirable feature: (i) it popped up where it was not expected, but vanished when we tried to generate it purposely; (ii) it caused

problems to Markov automata tools; (iii) it turned single probabilities into [min, max] intervals, although we observed in our experiments that the difference between "min" and "max" was always negligible, thus questioning the significance of nondeterminism; (iv) the scenarios causing nondeterminism were often difficult to understand. In all the case studies published so far, with the exception of [14,54] and [49], IMCs are deterministic; one would thus welcome new IMC-based case studies, in which nondeterminism would be present and undoubtedly useful.
- Accurately comparing compound models and IMCs is still an open problem. Our results are paradoxical: our translation function does not preserve steady-state and transient probabilities on tiny examples, but widely preserves them on the much more involved Erlangen mainframe. Having two perfectly reasonable modelling formalisms that, when applied to the same system, might give different numeric results is annoying. Just stating that they are incomparable in the general case is not satisfactory, because this fails to explain which formalism should be chosen in practice. More work is needed to understand the situation, namely by extending the theoretical results of [5] to the case of synchronised processes. This may require to modify our translation function to better take synchronisations into account and to perform other case-studies to observe if our results can be reproduced.

Apologies and Acknowledgements

We owe an apology to the master students in Grenoble who tried to model the Erlangen mainframe using IMCs (see Sect. 3.5). A number of them produced valid LNT models, but nondeterminism popped up unexpectedly, making steady-state and transient analyses intractable with CTMC tools. Solving this problem, which did not appear in prior case studies tackled using IMCs, required substantial research and an amount of effort out of proportion to a lab exercise.

We would like to thank Pedro d'Argenio, Marco Bernardo, and Frédéric Lang for sharing their expertise in helpful discussions, as well as Radu Mateescu and the anonymous reviewers for proofreading and commenting the present article. We are also grateful to Arnd Hartmanns (Modest), Dave Parker (PRISM), Tim Quatmann (Storm), and Matthias Volk (Storm) for their most useful answers and advises; we hope, but are not 100% sure, that we used their tools in the most effective way and look forward to get the next versions.

References

1. Baier, C.: On Algorithmic Verification Methods for Probabilistic Systems (1998), Habilitation thesis, Universität Mannheim, Germany
2. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: 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
3. Baier, C., Hermanns, H., Katoen, J.P., Haverkort, B.R.: Efficient Computation of Time-Bounded Reachability Probabilities in Uniform Continuous-Time Markov Decision Processes. Theoretical Computer Science **345**(1), 2–26 (2005)

4. Baier, C., Katoen, J.P., Hermanns, H.: Approximate Symbolic Model Checking of Continuous-Time Markov Chains. In: Baeten, J.C.M., Mauw, S. (eds.) Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99), Eindhoven, The Netherlands. Lecture Notes in Computer Science, vol. 1664, pp. 146–161. Springer (Aug 1999). https://doi.org/10.1007/3-540-48320-9_12
5. Bernardo, M., Corradini, F., Tesei, L.: Timed Process Calculi with Deterministic or Stochastic Delays: Commuting between Durational and Durationless Actions. Theoretical Computer Science **629**, 2–39 (2016). https://doi.org/10.1016/J.TCS.2016.02.022
6. Bernardo, M., Gorrieri, R.: A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. Theoretical Computer Science **202**(1–2), 1–54 (1998). https://doi.org/10.1016/S0304-3975(97)00127-8
7. Böde, E., Herbstritt, M., Hermanns, H., Johr, S., Peikenkamp, T., Pulungan, R., Rakow, J.H., Wimmer, R., Becker, B.: Compositional Dependability Evaluation for STATEMATE. IEEE Transactions on Software Engineering **35**(2), 274–292 (2009). https://doi.org/10.1109/TSE.2008.102
8. Buchholz, P., Hahn, E.M., Hermanns, H., Zhang, L.: Model Checking Algorithms for CTMDPs. 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. 225–242. Springer (Jul 2011). https://doi.org/10.1007/978-3-642-22110-1_19
9. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: Quantitative Model and Tool Interaction. In: Legay, A., Margaria, T. (eds.) Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'17), Part II, Uppsala, Sweden. Lecture Notes in Computer Science, vol. 10206, pp. 151–168 (Apr 2017). https://doi.org/10.1007/978-3-662-54580-5_9
10. Butkova, Y., Hartmanns, A., Hermanns, H.: A Modest Approach to Markov Automata. ACM Transactions on Modeling and Computer Simulation **31**(3), 14:1–14:34 (2021). https://doi.org/10.1145/3449355
11. Butkova, Y., Hatefi, H., Hermanns, H., Krcál, J.: Optimal Continuous Time Markov Decisions. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) Proceedings of the 13th International Symposium on Automated Technology for Verification and Analysis (ATVA'15), Shanghai, China. Lecture Notes in Computer Science, vol. 9364, pp. 166–182. Springer (Oct 2015). https://doi.org/10.1007/978-3-319-24953-7_12
12. Butkova, Y., Wimmer, R., Hermanns, H.: Long-Run Rewards for Markov Automata. In: Legay, A., Margaria, T. (eds.) Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'17), Part II, Uppsala, Sweden. Lecture Notes in Computer Science, vol. 10206, pp. 188–203 (Apr 2017). https://doi.org/10.1007/978-3-662-54580-5_11
13. 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 7.5) (Feb 2025), https://cadp.inria.fr/publications/Champelovier-Clerc-Garavel-et-al-10.html, INRIA, Grenoble, France
14. Chehaibar, G., Zidouni, M., Mateescu, R.: Modeling Multiprocessor Cache Protocol Impact on MPI Performance. In: Proceedings of the IEEE International Workshop on Quantitative Evaluation of Large-Scale Systems and Technologies

(QuEST'09), Bradford, UK. pp. 1073–1078. IEEE Computer Society Press (May 2009)
15. 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)
16. 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)
17. 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
18. Garavel, H.: Revisiting Sequential Composition in Process Calculi. Journal of Logical and Algebraic Methods in Programming **84**(6), 742–762 (Nov 2015). https://doi.org/10.1016/j.jlamp.2015.08.001
19. 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), full version available as INRIA Research Report 4492
20. Garavel, H., Hermanns, H., Parker, D.: Revisiting a Pioneering Concurrent Stochastic Problem: The Erlangen Mainframe. In: Jansen, N., Junges, S., Kaminski, B.L., Matheja, C., Noll, T., Quatmann, T., Stoelinga, M., Volk, M. (eds.) Principles of Verification: Cycling the Probabilistic Landscape (Part II) – Essays Dedicated to Joost-Pieter Katoen on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 15261, pp. 46–74. Springer (Nov 2024). https://doi.org/10.1007/978-3-031-75775-4
21. 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
22. 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
23. Garavel, H., Lang, F., Mounier, L.: Compositional Verification in Action. In: Howar, F., Barnat, J. (eds.) Proceedings of the 23rd International Conference on Formal Methods for Industrial Critical Systems (FMICS'18), Maynooth, Ireland – Essays Dedicated to Susanne Graf at the Occasion of Her 60th Birthday. Lecture Notes in Computer Science, vol. 11119, pp. 189–210. Springer (Sep 2018)
24. van Glabbeek, R.J., Weijland, W.P.: Branching Time and Abstraction in Bisimulation Semantics. Journal of the ACM **43**(3), 555–600 (1996)
25. Götz, N., Herzog, U., Rettelbach, M.: Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis

Using Stochastic Process Algebras. In: Donatiello, L., Nelson, R.D. (eds.) Performance Evaluation of Computer and Communication Systems, Joint Tutorial Papers of Performance'93 and Sigmetrics'93, Santa Clara, CA, USA. Lecture Notes in Computer Science, vol. 729, pp. 121–146. Springer (May 1993). https://doi.org/10.1007/BFB0013851

26. Gros, T.P.: Markov Automata Taken by Storm. Master's thesis, Saarland University, Germany (Jan 2018)
27. Guck, D., Han, T., Katoen, J.P., Neuhäußer, M.R.: Quantitative Timed Analysis of Interactive Markov Chains. In: Goodloe, A., Person, S. (eds.) Proceedings of the 4th International NASA Formal Methods Symposium (NFM'12), Norfolk, VA, USA. Lecture Notes in Computer Science, vol. 7226, pp. 8–23. Springer (Apr 2012). https://doi.org/10.1007/978-3-642-28891-3_4
28. Hahn, E.M., 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
29. Hartmanns, A., Hermanns, H.: The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification. In: Ábrahám, E., Havelund, K. (eds.) Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14), Grenoble, France. Lecture Notes in Computer Science, vol. 8413, pp. 593–598. Springer (Apr 2014). https://doi.org/10.1007/978-3-642-54862-8_51
30. Hensel, C., Junges, S., Katoen, J.P., 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
31. Hermanns, H.: Construction and Verification of Performance and Reliability Models. Bulletin of the EATCS **74**, 135–154 (2001)
32. Hermanns, H.: Interactive Markov Chains: The Quest for Quantified Quality, Lecture Notes in Computer Science, vol. 2428. Springer (2002)
33. 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
34. 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
35. Hermanns, H., Herzog, U., Mertsiotakis, V.: Stochastic Process Algebras – Between LOTOS and Markov Chains. Computer Networks **30**(9-10), 901–924 (1998). https://doi.org/10.1016/S0169-7552(97)00133-5
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)
37. Hermanns, H., Katoen, J.P.: Automated compositional markov chain generation for a plain-old telephone system. Sci. Comput. Program. (2000)
38. Hermanns, H., Katoen, J.P.: The How and Why of Interactive Markov Chains. In: de Boer, F.S., Bonsangue, M.M., Hallerstede, S., Leuschel, M. (eds.) Revised

Selected Papers of the 8th International Symposium on Formal Methods for Components and Objects (FMCO'09), Eindhoven, The Netherlands. Lecture Notes in Computer Science, vol. 6286, pp. 311–337. Springer (Nov 2009)
39. Hermanns, H., Rettelbach, M.: Syntax, Semantics, Equivalences, and Axioms for MTIPP. In: Herzog, U., Rettelbach, M. (eds.) Proceedings of the 2nd Workshop on Process Algebras and Performance Modelling (PAPM'94), Erlangen, Germany. Lecture Notes in Computer Science, vol. 1601, pp. 71–88. University of Erlangen-Nürnberg, Germany (Jul 1994)
40. Herzog, U.: Formal Description, Time and Performance Analysis: A Framework. In: Härder, T., Wedekind, H., Zimmermann, G. (eds.) Entwurf und Betrieb verteilter Systeme, Proceedings Fachtagung des Sonderforschungsbereiche 124 und 182 (Dagstuhl, Germany). Informatik-Fachberichte, vol. 264, pp. 172–190. Springer (Sep 1990). https://doi.org/10.1007/978-3-642-76309-0_10
41. 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
42. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
43. Johr, S.: Model Checking Compositional Markov Systems. Ph.D. thesis, Saarland University (Aug 2007)
44. Kemeny, J.G., Snell, J.L.: Finite Markov Chains. Undergraduate Texts in Mathematic, Springer (1976)
45. Kordon, F., Garavel, H., Hillah, L.M., Paviot-Adet, E., Jezequel, L., Rodríguez, C., Hulin-Hubard, F.: MCC'2015 – The Fifth Model Checking Contest. Transactions on Petri Nets and Other Models of Concurrency **XI**, 262–273 (2016)
46. 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
47. Mateescu, R.: Specification and Analysis of Asynchronous Systems using CADP. In: Merz, S., Navet, N. (eds.) Modeling and Verification of Real-Time Systems – Formalisms and Software Tools, chap. 5, pp. 141–170. ISTE publishing / John Wiley (2008), http://hal.inria.fr/inria-00264235
48. Mateescu, R., Garavel, H.: XTL: A Meta-Language and Tool for Temporal Logic Model-Checking. In: Margaria, T. (ed.) Proceedings of the International Workshop on Software Tools for Technology Transfer (STTT'98), Aalborg, Denmark. pp. 33–42. BRICS (Jul 1998)
49. 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)
50. 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
51. Neuhäußer, M.R.: Model Checking Nondeterministic and Randomly Timed Systems. Ph.D. thesis, RWTH Aachen University, Germany (2010)
52. Queille, J.P., Sifakis, J.: Specification and Verification of Concurrent Systems in CESAR. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) Proceedings of the 5th International Symposium on Programming, Torino, Italy. Lecture Notes in Computer Science, vol. 137, pp. 337–351. Springer (1982)

53. Zhang, L., Neuhäußer, M.R.: Model Checking Interactive Markov Chains. In: Esparza, J., Majumdar, R. (eds.) Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10), Paphos, Cyprus. Lecture Notes in Computer Science, vol. 6015, pp. 53–68. Springer (Mar 2010). https://doi.org/10.1007/978-3-642-12002-2_5
54. Zidouni, M.: Modélisation et analyse des performances de la bibliothèque MPI en tenant compte de l'architecture matérielle. Ph.D. thesis, Université de Grenoble (May 2010), http://hal.inria.fr/tel-00526164/en