FlyFast: A Scalable Approach to Probabilistic Model-Checking Based on Mean-Field Approximation

Diego Latella[1], Michele Loreti[2](✉), and Mieke Massink[1]

[1] CNR-ISTI, Pisa, Italy
{D.Latella,M.Massink}@cnr.it
[2] Università di Firenze, Firenze, Italy
michele.loreti@unifi.it

Abstract. Model-checking is an effective formal verification technique that has also been extended to quantitative logics and models such as PCTL and DTMCs as well as CSL and CTMCs/CTMDPs. Unfortunately, the state-space explosion problem of classical model-checking algorithms affects also quantitative extensions. Mean-field techniques provide approximations of the mean behaviour of *large* population models. These approximations are *deterministic*: a *unique* value of the fractions of agents in each state is computed for each time instant. A drastic reduction of the size of the model is obtained enabling the definition of an efficient model-checking algorithm. This paper is a survey of work we have done in the last few years in the area of mean-field approximated probabilistic model-checking. We start with a brief description of FlyFast, an on-the-fly model checker we have developed for approximated bounded PCTL model-checking, based on mean-field population DTMC approximation. Then we show an example of use of FlyFast in the context of Collective Adaptive Systems. We also discuss two additional interesting front-ends for FlyFast; the first one is a translation from CTMC-based population models and (a fragment of) CSL that allows for approximate probabilistic model-checking in the continuous stochastic time setting; the second one is a translation from a predicate-based process interaction language that allows for probabilistic model-checking of models based on components equipped both with behaviour and with attributes, on which predicates are defined that can be used in component interaction primitives.

Keywords: Probabilistic on-the-fly model-checking · Mean-field approximation · Discrete time Markov chains · Time bounded probabilistic computation tree logic · Collective Adaptive Systems

1 Introduction and Related Work

Model-checking is an effective, powerful, and successful formal verification technique for concurrent and distributed systems that has also been extended to

J.-P. Katoen et al. (Eds.): Brinksma Festschrift, LNCS 10500, pp. 254–275, 2017.
DOI: 10.1007/978-3-319-68270-9_13

quantitative logics and models. It consists of an efficient procedure that, given a model $\mathcal{M}$ of the system, typically composed of system states and related transitions, decides whether $\mathcal{M}$ satisfies a logical formula Φ, typically drawn from a temporal logic. Traditionally, model-checking approaches are divided into two broad categories: *global* approaches and *local* approaches.

In *global* model-checking approaches, the procedure determines the set of *all* states in $\mathcal{M}$ that satisfy Φ. Global model-checking algorithms are popular because of their computational efficiency and can be found in many model-checkers, both in a qualitative and in a probabilistic setting (see e.g. [2,3,11,16,36]). The set of states that satisfy a formula is constructed recursively in a *bottom-up* fashion following the syntactic structure of the formula. Moreover, for stochastic model-checking, the global model-checking algorithm relies on existing and well-known algorithms for Markov Chains, such as those for transient and steady-state analysis (see e.g. [2]). Despite their success, the scalability of model-checking algorithms has always remained a concern due to the potential combinatorial explosion of the state space that needs to be searched.

This is unfortunate since current trends in information technology, like the Internet of Things (IoT), include specifically systems composed of a large number of components, often acting collectively and adapting to changing conditions, the so called *Collective Adaptive Systems*[1] (CAS), like, for instance gossip protocols, self-organised collective decision making, computer epidemic and smart urban transportation systems and decentralised control strategies for smart grids [4,10,12,49]. Given that large portions of the IoT are intrinsically (part of) critical infrastructures, with safety, security, and, in general, high dependability requirements, it is of great importance that system designers have the possibility to perform formal analysis before developing and deploying them.

In order to mitigate the state space explosion problem, in the qualitative analysis domain, *local* model-checking algorithms have been proposed that, given a state s in $\mathcal{M}$, determine whether s satisfies Φ. Local model-checking approaches use the so called 'on-the-fly' paradigm (see e.g. [5,17,26,32]) and follow a *top-down* approach that does not require global knowledge of the complete state space. For each state that is encountered, starting from a given state, the outgoing transitions are followed to adjacent states, constructing step by step local knowledge of the state space until it is possible to decide whether the given state satisfies the formula. For qualitative model-checking, local model-checking algorithms have been shown to have the same worst-case complexity as the best existing global procedures for the above mentioned logics. However, in practice, they have better performance when only a subset of the system states need to be analysed to determine whether a system satisfies a formula. Furthermore, local model-checking may still provide some results in case of systems with a very large or even infinite state space where global model-checking approaches would be impossible to use.

In the context of probabilistic model-checking several on-the-fly approaches have been proposed, among which [21,29,40]. In [21], a probabilistic

[1] See, e.g. www.focas.eu/adaptive-collective-systems.

model-checker is shown for the *time bounded* fragment of the Probabilistic Computation Tree Logic (PCTL) [30]. An on-the-fly approach for *full* PCTL model-checking is proposed in [40] where, actually, a specific *instantiation* is presented of an algorithm which is *parametric* with respect to the specific probabilistic processes modelling language and logic, and their specific semantics. Finally, in [29] an on-the-fly approach is used for detecting a maximal relevant search depth in an infinite state space and then a *global* model-checking approach is used for verifying bounded Continuous Stochastic Logic (CSL) [1,2] formulas in a continuous time setting on the selected subset of states.

An on-the-fly approach by itself however, does not solve the challenging scalability problems that arise in truly large parallel systems, such as CAS. To address this type of scalability challenges in probabilistic model-checking, recently, several approaches have been proposed. In [28,31] approximate probabilistic model-checking is introduced. This is a form of statistical model-checking that consists in the generation of random executions of an *a priori* established maximal length [37]. On each execution the property of interest is checked and statistics are performed over the outcomes. The number of executions required for a reliable result depends on the maximal error-margin of interest. The approach relies on the analysis of individual execution traces rather than a full state space exploration and is therefore memory-efficient. However, the number of execution traces that may be required to reach a desired accuracy may be large and therefore time-consuming. The approach works for general models, i.e. models where stochastic behaviour can also be non Markovian and that do not necessarily model populations of similar objects. On the other hand, the approach is not independent from the number of objects involved.

In [38], we presented a scalable model-checking algorithm, based on mean-field approximation, for the verification of time bounded PCTL properties of an individual[2] in the context of a system consisting of a large number of interacting objects. Also this algorithm is actually an instantiation of the above mentioned parametric algorithm for (exact) probabilistic model-checking [40]. In this case, the parametric algorithm is instantiated on (time bounded PCTL and) the *approximate*, mean-field, semantics of a population process modelling language. The approach is based on the idea of fast simulation, as introduced in [46]. More specifically, the behaviour of a generic agent with S states in a clock-synchronous system with a *large* number N of instances of the agent at given step (i.e. time) t is approximated by $\mathbf{K}(\boldsymbol{\mu}(t))$ where $\mathbf{K}(\mathbf{m})$ is the $S \times S$ probability transition matrix of a (inhomogeneous) DTMC and $\boldsymbol{\mu}(t)$ is a vector of size S approximating the mean behaviour of the global system at t; each element of $\boldsymbol{\mu}(t)$ is associated with a distinct state of the agent, say C, and gives an approximation of (the average of) the fraction of the instances of the agent that are in state C in the

[2] The technique can be applied also to a finite selection of individuals; in addition, systems with several distinct types of individuals can be dealt with. For the sake of simplicity, in the present paper we consider systems with many instances of a single individual only and we focus in the model-checking a single individual in such a context.

global system, at step t. Note that such an approximation is *deterministic*, i.e. $\boldsymbol{\mu}$ is a *function* of the step t, computed iteratively, using (again) matrix $\mathbf{K}(\mathbf{m})$; the exact behaviour of the rest of the system would instead be a *large* DTMC in turn. Note furthermore, that $\mathbf{K}(\mathbf{m})$ does not depend on N; in other words, the cost of the analysis *is independent* from the number of objects involved, but only depends on the number of states of the single individual object. Our work is based on mean-field approximation in the *discrete time* setting; approximated mean-field model-checking in the *continuous time* setting has been presented in the literature as well. In the latter case, the deterministic approximation of the global system behaviour is formalised as an initial value problem using a set of differential equations. Preliminary ideas on the exploitation of mean-field convergence in continuous time for model-checking were informally sketched in a presentation at QAPL 2012[3] [34], but no model-checking algorithms were presented. Follow-up work on the above mentioned approach can be found in [33] which relies on earlier results on fluid model-checking by Bortolussi and Hillston [7], later published in [8]. Bortolussi and Hillston propose a *global CSL* model-checking procedure for the verification of properties of a selection of individuals in a population. The procedure relies on mean-field convergence and fast simulation results in a *continuous* time setting (see also [9,18,25] and references therein). The approach in [7,8] is based on an *interleaving* model of computation, rather than a clock-synchronous one. Furthermore, a *global* model-checking approach, rather than an on-the-fly approach is adopted; it is also worth noting that the treatment of nested formulas, whose truth value may change over time, turns out to be much more difficult in the interleaving, continuous time, global model-checking approach than in the clock-synchronous, discrete time, on-the-fly one.

We conclude this brief overview on related work by mentioning the approach of using techniques and tools developed for continuous signal monitoring as means for performing approximated global model checking of probabilistic models. In this approach, a deterministic, continuous, approximation of a population system model is first computed [9], and then monitoring techniques are applied on the resulting function of continuous time [23,24]. Recently, this approach has been extended in order to include also spatial features [50], as originally proposed in [13].

We finally note that one should keep in mind that mean-field/fluid procedures are based on *approximations* of the global behaviour of a system. Consequently, the techniques should be considered as *complementary* to other, more accurate analysis techniques for CAS, primarily those based on stochastic simulation, like for example statistical model-checking. In practice, given the high computational cost of simulation based techniques, especially when compared with the very low cost of the mean-field based techniques, the latter are more suitable for getting first ideas on the main features of the models at hand and a first screening

[3] Tenth Workshop on Quantitative Aspects of Programming Languages, March 31 - April 1, 2012, Tallinn, Estonia.

thereof. Then, when only a few options are left, more detailed analyses could be performed and more accurate techniques would be recommended.

In this paper, we present a survey of work we have carried out recently within the context of the EU project QUANTICOL[4] in the area of mean-field approximated probabilistic model-checking. We start with a brief description, in Sect. 2, of FlyFast, the on-the-fly model checker which implements the procedure we proposed in [38,42,43]. Then, in Sect. 3 we show a complete example of use of FlyFast in the context of Collective Adaptive Systems, taken from [39]. In Sect. 4 we discuss two additional interesting front-ends for FlyFast; the first one, originally presented in [41], is a translation to FlyFast from CTMC-based population models and (a fragment of) CSL that allows for approximate probabilistic model-checking in the continuous stochastic time setting; the second one, originally presented in [15], is a translation to FlyFast from a predicate-based process interaction language that allows for probabilistic model-checking of bounded PCTL formulas on models based on components equipped both with behaviour and with attributes; component interaction takes place via communication primitives using predicates over attributes for expressing the set of partners in multi-cast communication. We finally draw some conclusions in Sect. 5.

2 A Brief Overview of FlyFast

In this section we recall the main features of FlyFast.[5] The reader interested in details is referred to [38,42,43]. A FlyFast model specification characterises a system consisting of the *clock-synchronous product* of a (large) number of instances of a probabilistic process. Systems with several distinct types of processes can be specified as well; here we consider only the case of a single process type for the sake of simplicity. The size of the system is assumed constant during system evolution; FlyFast does not support explicit dynamic creation/deletion of processes. The behaviour of the probabilistic process is specified by a set Δ of state and action probability definitions. A state definition has the following syntax: **state** $\mathtt{C}\ \{\mathtt{a}_1.\mathtt{C}_1 + \ldots + \mathtt{a}_r.\mathtt{C}_r\}$ where $\mathtt{a}_i \in \mathcal{A}$—the set of FlyFast *actions*—$\mathtt{C}, \mathtt{C}_i \in$ —the set of FlyFast *states*—and, for $i, j = 1, \ldots, r$ $\mathtt{a}_i \neq \mathtt{a}_j$ if $i \neq j$; note that $\mathtt{C}_i = \mathtt{C}_j$ with $i \neq j$ is allowed instead. The informal meaning of the above definition is that, when in state $\mathtt{C}$, the process can jump to state $\mathtt{C}_1$, by executing (atomic) action $\mathtt{a}_1$, or to state $\mathtt{C}_2$, by executing (atomic) action $\mathtt{a}_2$, and so on. Each action has a probability assigned by means of an action probability definition of the form **action** $\mathtt{a}$: *exp* where *exp* is an expression involving constants and **frc** $(\mathtt{C})$ terms. Constants are floating point values or names associated to such values using the construct **const** `name` = `value`—we let A denote the set of such auxiliary definitions; **frc** $(\mathtt{C})$ denotes the element associated to state $\mathtt{C}$

[4] http://www.quanticol.eu.

[5] FlyFast (https://quanticol.github.io/jSAM/flyfast.html) is provided within the jSAM (java StochAstic Model Checker) framework which is an open source Eclipse plugin (https://quanticol.github.io/jSAM/).

in the current occupancy measure vector—*o.m.v.* in the sequel—that is a vector with as many elements as the number of states of the individual process; the element associated to a specific state gives the fraction of the subpopulation currently in that state over the size of the overall population; the *o.m.v.* is a compact abstract representation of the system global state, where process identity is lost. Thus, the probability of executing a transition of a process in the system may depend on the global distribution of the processes in their local states within the system; process interaction is thus *probabilistic* and *indirect*, via transition probabilities, i.e. functions of the *o.m.v.*. Note that, whenever the exit probability p of a state is smaller than 1, FlyFast implicitly inserts a self-loop in the state, associated with the residual probability $1-p$. The initial state of the system is specified by means of the **system** construct, followed by the name of the system model, and the vector $\mathbf{C_0}$ of the names of the initial state of all other instances, which implicitly specifies also the size N of the system. By convention, the first element $\mathbf{C_0}[1]$ of vector $\mathbf{C_0}$ refers to the individual process to analyse. In general more then one process can be specified for analysis; here we consider only the case of the single process for the sake of simplicity.

```
const N = 2000                                  state S {inf_ext.E + inf_int.E}
const alpha_e = 0.1                             state E {activate.I}
const alpha_i = 0.2                             state I {recover.R}
const alpha_r = 0.2                             state R {loss.S}
const alpha_a = 0.4
const alpha_l = 0.1
action inf_ext : alpha_e
action inf_int : alpha_i * frc (I)              system SEIR = ⟨S[N], E[0], I[0], R[0]⟩
action activate : alpha_a
action recover : alpha_r
action loss : alpha_l
```

Fig. 1. A FlyFast specification of an epidemic model

Example 1 (An epidemic system model). In Fig. 1 the FlyFast specification of the epidemic model discussed in [38] is shown. The system is composed of 2000 instances of a process with four states; when in state `S` (susceptible) the process can become exposed (state `E`) either via an external infection, with probability `alpha_e`, or via internal infection, with a probability that is proportional to the fraction of processes in the system that are already infected, i.e. `alpha_i` $*$ **frc** (`I`). The infection activates in an exposed process with probability `alpha_a`, leading to state `I`. An infected process may recover with probability `alpha_r` and then loose immunity with probability `alpha_l`. Initially, all 2000 instances are in state `S`[6].

Given specification $\langle \Delta, A, \mathbf{C_0} \rangle$ for a system model of size N, FlyFast generates a transition probability matrix $\mathbf{K}(\mathbf{m})$ such that $\mathbf{K}(\mathbf{m})_{c,c'}$ is the probability for the (individual) probabilistic process to jump from state `C` to `C'`, given the

[6] In FlyFast, the notation `C[n]` is used for indicating n copies of state `C`.

current *o.m.v.* $\mathbf{m}$. Thus, $\mathbf{K}(\mathbf{m})$ is a function of the *o.m.v.* $\mathbf{m}$; strictly speaking, $\mathit{\Delta}$ characterises an inhomogeneous DTMC. In [38,42] the details of the formal operational semantics definition for the model specification language are provided as well as the procedure for generating $\mathbf{K}(\mathbf{m})$; in the sequel we recall only the main steps. Let $\mathit{\Delta}$ be the set of states defined in $\mathit{\Delta}$, with $|\mathit{\Delta}| = S$, $\mathcal{U}^S = \{(m_1, \ldots, m_S) \in [0,1]^S | m_1 + \ldots + m_S = 1\}$ denote the unit simplex of dimension S, and $\mathcal{I} : \mathit{\Delta} \to \{1, \ldots, S\}$ be a bijection. For state $\mathtt{C} \in \mathit{\Delta}$, with $\mathcal{I}(\mathtt{C}) = c$, the interpretation $[\![\mathbf{frc}\,(\mathtt{C})]\!]_{\mathbf{m}}$ of $\mathbf{frc}\,(\mathtt{C})$ in the current *o.m.v.* $\mathbf{m} = (m_1, \ldots, m_S)$ is defined as expected: $[\![\mathbf{frc}\,(\mathtt{C})]\!]_{\mathbf{m}} = m_c$, i.e. $\mathbf{frc}\,(\mathtt{C})$ is the *fraction* of the subpopulation currently in state $\mathtt{C}$ over the size of the overall population, which, by definition of the *o.m.v.*, is exactly the element m_c of $\mathbf{m}$. The probability associated with an action $\mathtt{a}$ by action probability definition **action** $\mathtt{a} : E$ is a function $\pi_{\mathtt{a}}(\mathbf{m})$ of the *o.m.v.* $\mathbf{m}$, defined as $\pi_{\mathtt{a}}(\mathbf{m}) = [\![E]\!]_{\mathbf{m}}$, where the interpretation function $[\![\cdot]\!]$ is defined recursively on arithmetic expressions E involving **frc** and constants, in the obvious way. More precisely, letting $\mathtt{C} \overset{\mathtt{a}}{\rightarrowtail} \mathtt{C}'$ represent an $\mathtt{a}$-labelled transition in the operational semantics of the FlyFast modelling language and assuming $c = \mathcal{I}(\mathtt{C}) \neq c' = \mathcal{I}(\mathtt{C}')$, the probability matrix function $\mathbf{K} : \mathcal{U}^S \times \{1, \ldots, S\} \times \{1, \ldots, S\} \to [0,1]$ is defined as follows: $\mathbf{K}(\mathbf{m})_{c,c'} = \sum_{\mathtt{a}:\mathtt{C} \overset{\mathtt{a}}{\rightarrowtail} \mathtt{C}'} \pi_{\mathtt{a}}(\mathbf{m})$ and $\mathbf{K}(\mathbf{m})_{c,c} = 1 - \sum_{j \in \{1,\ldots,S\} \setminus \{c\}} \mathbf{K}(\mathbf{m})_{c,j}$. In other words, $\mathbf{K}(\mathbf{m})_{c,c'}$ is the *cumulative* probability of jumping from $\mathtt{C}$ to $\mathtt{C}'$, abstracting from the specific action performed by the process in the jump; this abstraction choice is typical of probabilistic, PCTL/DTMC-based approaches. Finally, note that, by construction, $\mathbf{K}(\mathbf{m})$ does *not* depend on N.

Example 2. It is easy to see that, for the model of Example 1, the resulting matrix is the following one, with $\mathbf{m} = (m_s, m_e, m_i, m_r)$ where m_s is the fraction of processes in state $\mathtt{S}$, m_e is the fraction in state $\mathtt{E}$, m_i is the fraction in state $\mathtt{I}$, and m_r is the fraction in state $\mathtt{R}$:

$$\mathbf{K}(m_s, m_e, m_i, m_r) = \begin{pmatrix} 1 - (0.1 + 0.2m_i) & 0.1 + 0.2m_i & 0 & 0 \\ 0 & 0.6 & 0.4 & 0 \\ 0 & 0 & 0.8 & 0.2 \\ 0.1 & 0 & 0 & 0.9 \end{pmatrix}$$

The exact probabilistic semantics of the complete system model is easily given as product of N instances of $\mathbf{K}$ with appropriate *o.m.v.* parameter and argument states. In other words, the transitions of different processes are intended as stochastically *independent*[7]. More precisely, for global system state $\mathbf{C} \in \mathit{\Delta}^N$, let the associated *o.m.v.* $\mathbf{M}(\mathbf{C})$ be defined as $\mathbf{M}(\mathbf{C}) = (M_1, \ldots, M_S)$ with $M_i = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}_{\{\mathbf{C}_{[n]} = \mathcal{I}^{-1}(i)\}}$ where $\mathbf{1}_{\{\alpha = \beta\}}$ is 1, if $\alpha = \beta$, and 0 otherwise. The probabilistic semantics of the system is the DTMC $X^{(N)}(t)$ with one-step transition probability $S^N \times S^N$ matrix $\mathbf{P}$ with $\mathbf{P}_{\mathbf{C},\mathbf{C}'} = \prod_{n=1}^{N} \mathbf{K}(\mathbf{M}(\mathbf{C}))_{\mathcal{I}(\mathbf{C}_{[n]}),\mathcal{I}(\mathbf{C}'_{[n]})}$

[7] It is worth stressing here that in the model of process interaction presented in [46], which FlyFast is based on, processes do *not synchronize* on actions explicitly (i.e. there is no notion of *randez-vous* here). Process interaction is only *indirect*, by means of the impact of the *o.m.v.* on individual transition probabilities.

and initial probability mass all in $\mathbf{C}_0$. FlyFast provides a standard stochastic simulation functionality based on the exact probabilistic semantics, namely matrix $\mathbf{P}$. In particular one can execute single runs or get averages of a user-specified number of runs. The output is given in the form of traces of the *o.m.v.* DTMC $\mathbf{M}^{(N)}(t) = \mathbf{M}(X^{(N)}(t))$. In addition, the tool can perform *exact*, on-the-fly (full) PCTL model checking using $\mathbf{P}$. FlyFast accepts state formulas built out of atomic propositions, negations, disjunctions and probabilistic quantification over path-formulas; the latter are next and until formulas. Of course, as opposed to approximate model-checking, exact PCTL model-checking of a formula can be used only if the portion of the state-space which needs to be generated and analysed for deciding satisfaction of the formula is not too large.

Example 3. For the epidemic model of Example 1, but with constant N set to 8, for a system with only 8 processes, we consider the following properties, where `tt` stands for *true*, `LowInf` is defined, using the **formula** construct of FlyFast, as follows: **formula** `LowInf` : (**frc** `I`) < 0.25, and `I` (`E`, respectively) labels all system states the first element of which is process state `I` (`E`, respectively):

P1 the worm will be active in the first component within k steps with a probability that is at most p: $\mathcal{P}_{\leq p}(\texttt{tt}\ \mathcal{U}^{\leq k}\ \texttt{I})$;

P2 the probability that the first component is infected, but latent, in the next k steps while the worm is active on less then 25% of the components is at most p: $\mathcal{P}_{\leq p}(\texttt{LowInf}\,\mathcal{U}^{\leq k}\ \texttt{E})$;

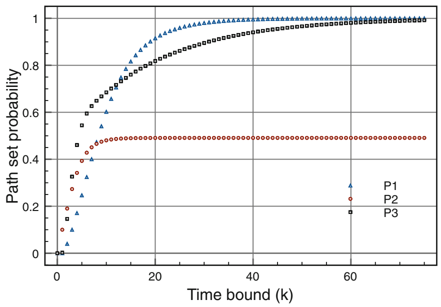
P3 the probability to reach, within k steps, a configuration where the first component is not infected but the worm will be activated with probability greater than 0.3 within 5 steps is at most p:

$$\mathcal{P}_{\leq p}(\texttt{tt}\ \mathcal{U}^{\leq k}\,(\texttt{!E} \wedge \texttt{!I} \wedge \mathcal{P}_{>0.3}(\texttt{tt}\ \mathcal{U}^{\leq 5}\ \texttt{I}))).$$

In Fig. 2 the result of exact PCTL model-checking of Example 1 is reported. On the left the probability of the set of paths that satisfy the path-formulae used in the three formulae above is shown for k from 0 to 70. On the right the time needed to perform the analysis using PRISM [36] and using FlyFast exact PCTL model checking are presented[8].

More interestingly, FlyFast can compute the *deterministic* limit of the *o.m.v.* DTMC, for $N \to \infty$, and execute time bounded PCTL model-checking using such a deterministic approximation. The approach has been inspired by *Fast Simulation*, proposed in [46] and is based on Theorem 4.1 of [46], actually on a simplified version of the theorem, thanks to the specific syntax of the expressions used in FlyFast action probability definitions. Informally, let $\mathbf{C_0}^{(N)}$ be the initial state of the FlyFast specification of a system with N processes and assume there exists $\boldsymbol{\mu_0} \in \mathcal{U}^S$ such that almost surely $\lim_{N\to\infty} \mathbf{M}(\mathbf{C_0}^{(N)}) = \boldsymbol{\mu_0}$. Let

[8] We used a 1.86 GHz Intel Core 2 Duo with 4 GB. State space generation time of PRISM is not counted. The experiments are available in the FlyFast web site, showing that the latter has comparable performance. Worst-case complexity of both algorithms are also comparable.



	PRISM	Exact on-the-fly
$P1$	$108.479s$	$29.587s$
$P2$	$51.816s$	$3.409s$
$P3$	$216.952s$	$85.579s$

Model parameter values:
$\alpha_e = 0.1$, $\alpha_i = 0.2$, $\alpha_r = 0.2$
$\alpha_a = 0.4$, $\alpha_l = 0.1$

Fig. 2. Exact model-checking results (left) and verification time (right).

function $\boldsymbol{\mu}(t)$ be defined as follows: $\boldsymbol{\mu}(0) = \boldsymbol{\mu}_{\mathbf{0}}$ and $\boldsymbol{\mu}(t+1) = \boldsymbol{\mu}(t)^T \cdot \mathbf{K}(\boldsymbol{\mu}(t))$, where, as usual, $\mathbf{m}^T$ is the transpose of vector $\mathbf{m}$. Then, for any fixed time τ, almost surely $\lim_{N\rightarrow\infty} \mathbf{M}^{(N)}(\tau) = \boldsymbol{\mu}(\tau)$—cfr. Theorem 4.1 of [46]. So, the matrix $\mathbf{K}(\mathbf{m})$ generated by FlyFast can be conveniently (re)used also for *approximating* the *o.m.v.*, which, we recall, is an abstract representation of the global system state; it is important to stress here that the *o.m.v.* $\mathbf{M}^{(N)}(t)$ is a stochastic process, whereas the approximation we use, $\boldsymbol{\mu}(t)$, is *deterministic*, i.e. just a function of the step (time) t. We consider now the stochastic process $\mathcal{H}(t)$ the generic state of which, at time t, is a pair $(\mathtt{C}, \boldsymbol{\mu}(t))$. The first component $\mathtt{C}$ is the current state of the selected process in the system we are interested in, and the second component $\boldsymbol{\mu}(t)$ represents the current global system state. It is easy to see that $\mathcal{H}(t)$ is a DTMC and that the probability of a jump from state $(\mathtt{C}, \boldsymbol{\mu}(t))$ to $(\mathtt{C}', \boldsymbol{\mu}(t+1))$ is $\mathbf{K}(\boldsymbol{\mu}(t))_{\mathcal{I}(\mathtt{C}),\mathcal{I}(\mathtt{C}')}$. $\mathcal{H}(t)$ is the *approximated* probabilistic semantics of the system model. By performing on-the-fly model-checking on $\mathcal{H}(t)$—where state labels of the selected process are exported to pair states $(\mathtt{C}, \boldsymbol{\mu})$—FlyFast provides an approximated, mean-field based, efficient *time bounded* PCTL model-checking functionality. In other words, for any fixed time τ, sample $\mathbf{C}$ of $X(t)$ at time τ and safe formula[9] Φ the following holds: $\mathbf{C} \models_{X(t)} \Phi$ if and only if $(\mathbf{C}[1], \boldsymbol{\mu}(\tau)) \models_{\mathcal{H}(t)} \Phi$. In the case of mean-field model-checking, the set of atomic propositions is the set of states of the single agent or assertions on the components of the *o.m.v.*; in addition one can assign a name to a formula and use it in larger formulas. Finally, note that FlyFast can provide, as a by-product, the plot of $\boldsymbol{\mu}(\tau)$ for τ ranging in a user-specified range.

Example 4. Figure 3 shows the result of mean-field, approximated model-checking by FlyFast on the model of Example 1 with formulas as in Example 3, for the first object of a large population of 2000 objects, each initially in state $\mathtt{S}$. In Fig. 3 (left) the same properties are considered as in Example 3. The analysis takes less than a second and is *insensitive* to the total population size. Figure 3 (right) shows how the probability measure of the set of paths satisfying the

[9] We refer to [38,42] for the characterisation of *safe* formulas and a related discussion.

formula $\texttt{tt}\ \mathcal{U}^{\leq k}\,(\texttt{!E}\ \wedge\texttt{!I}\wedge \mathcal{P}_{>0.3}(\texttt{tt}\ \mathcal{U}^{\leq 5}\ \ \texttt{I}))$ of property P3 on page 8, (for $k = 3$), changes for initial time $t0$ varying from 0 to 10.

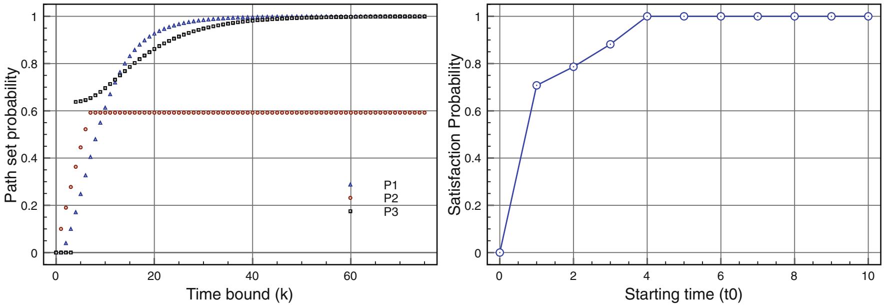


Fig. 3. Mean-field model-checking results.

We close this section by stressing that the exact full PCTL model-checker and the approximated mean-field time bounded PCTL model-checker are both instances of the *same* parametric implementation of an on-the-fly model-checking algorithm. Furthermore, the computation of the set of states to be analysed at the next step is a key operation of the on-the-fly procedure and, for exact model-checking, in the worst case the step returns S^N states, whereas for mean-field model-checking, the number of states returned in the worst case drops dramatically to S.

3 Predator-Prey Model of Lotka-Volterra in FlyFast

The next example we consider is a widely studied model for ecological competition, first independently investigated by the biophysicist Alfred Lotka and the Italian mathematician and physicist Vito Volterra in the twenties of the 19th century [48,51]. Since then, the model has been studied extensively by numerous other scientists and some of its elements are still at the basis of many population models that have been developed in the course of time, both in continuous time, e.g. [19,27] and references therein, and in discrete time settings, e.g. [22]. In its simplest form the model can be interpreted as a simplified and idealised description of two species in an ecosystem, often indicated as predator and prey, or foxes and rabbits for a concrete example.

In the variant we consider here we assume that each element of the two species can be in one of two states; it is either alive, or it is somehow 'dormant' waiting to get born again. We do this because the language we use does not provide explicit constructs for the dynamic creation of objects and is implicitly assuming that the total population size of all species remains constant. If we choose the size of the 'dormant' part of the population of each species large enough, this should not have any effect on the part of the population that is alive, since there are always enough dormant rabbits and foxes to get born.

As in the original version, we assume that the model depends on four parameters:

– The net probability ‘a’ of an increase in the size of the rabbit population which is the difference between the natural birth and death probabilities.
– The probability ‘b’ of rabbits that die because they are eaten by foxes
– The probability ‘e’ of extra foxes being born and surviving because they eat rabbits (efficiency).
– The net probability ‘c’ of the natural decrease in the population of foxes. Since the life of a fox depends on the availability of rabbits, there is a natural tendency of foxes to die when there a few or no rabbits.

A model in terms of difference equations of the populations of foxes and rabbits can then be given by:

$$\begin{aligned} RD(t+1) &= RD(t) + b\cdot h\cdot RL(t)\cdot FL(t) - a\cdot h\cdot RL(t)\\ RL(t+1) &= RL(t) + a\cdot h\cdot RL(t) - b\cdot h\cdot RL(t)\cdot FL(t)\\ FD(t+1) &= FD(t) - e\cdot h\cdot RL(t)\cdot FL(t) + c\cdot h\cdot FL(t)\\ FL(t+1) &= FL(t) + e\cdot h\cdot RL(t)\cdot FL(t) - c\cdot h\cdot FL(t) \end{aligned} \tag{1}$$

where t ranges over the set of the natural numbers, RD and RL are the fractions of ‘dormant’ and ‘alive’ rabbits, respectively, and FD and FL the fractions of ‘dormant’ and ‘alive’ foxes, respectively. The factor h is a rescaling factor for the duration of steps and $0 < h < 1$. The smaller the value of h the smaller the relative probabilities of the different events and the more accurate the results, but at the cost of an increase of the number of steps per time-unit in the model-checking procedure, which takes more time. For the model in this section we chose $h = 0.125$. Note that when this discrete model is interpreted as an approximation of the well-known continuous time model, i.e. in terms of differential equations, this approximation is not perfect, in the sense that the solution of the differential equations would give a perfect oscillating behaviour, whereas the solution of the difference equations will result in a small error in each step. This error has a cumulative effect resulting in oscillations with ever higher peaks, as can easily be observed in the results. A better approximation of the continuous model could be reached by using a more sophisticated integration method instead of the Euler method that is used implicitly in this case study.

The FlyFast specification of the Lotka-Volterra model is shown in Fig. 4. Assuming $\mathcal{I}(\texttt{RD}) = 1$, $\mathcal{I}(\texttt{RL}) = 2$, $\mathcal{I}(\texttt{FD}) = 3$, $\mathcal{I}(\texttt{FL}) = 4$, the 4×4 matrix $\mathbf{K} : \mathcal{U}^4 \times \{1,\ldots,4\} \times \{1,\ldots,4\} \to [0,1]$ generated by FlyFast is shown below, noting that the matrix is stochastic for the time interval of interest (and in particular $m_1 \neq 0 \neq m_3$):

$$\mathbf{K}(m_1,m_2,m_3,m_4) = \begin{pmatrix} 1 - a\cdot h\cdot\frac{m_2}{m_1} & a\cdot h\cdot\frac{m_2}{m_1} & 0 & 0\\ b\cdot h\cdot m_4 & 1 - b\cdot h\cdot m_4 & 0 & 0\\ 0 & 0 & 1 - e\cdot h\cdot m_2\cdot\frac{m_4}{m_3} & e\cdot h\cdot m_2\frac{m_4}{m_3}\\ 0 & 0 & c\cdot h & 1 - c\cdot h \end{pmatrix}$$

```
const a = 0.04
const b = 0.5
const c = 0.05
const e = 0.2
const h = 0.125
action rborn : a ∗ h ∗ frc (RL)/frc (RD)
action rdies : b ∗ h ∗ frc (FL)
action fborn : e ∗ h ∗ frc (RL) ∗ frc (FL)/frc (FD)
action fdies : c ∗ h

state RD {rborn.RL}
state RL {rdies.RD}
state FD {fborn.FL}
state FL {fdies.FD}

system LoVo = ⟨RD[5000], RL[1000], FD[3000], FL[1000]⟩
```

Fig. 4. A FlyFast specification of the Lotka-Volterra model

It is easy to see that by computing $\boldsymbol{\mu}(t+1)$ as $\boldsymbol{\mu}(t+1) = \boldsymbol{\mu}(t)^T \cdot \mathbf{K}(\boldsymbol{\mu}(t))$, where $\boldsymbol{\mu}(t) = (\mu_1(t), \mu_2(t), \mu_3(t), \mu_4(t))$, one obtains again the difference Eq. (1) of page 11, where, of course, $\mu_1(t)$ stands for $RD(t)$, $\mu_2(t)$ for $RL(t)$, $\mu_3(t)$ for $FD(t)$, and $\mu_4(t)$ for $FL(t)$.

As it is well known, the global behaviour of the (idealised) model shows oscillations in the populations of rabbits and foxes for certain values of the model parameters. In fact, the model has very interesting behaviour and is therefore widely studied, but in this paper we focus mainly on the illustration of the application of fast mean field model checking of an individual rabbit or fox in the context of the overall oscillating behaviour. For example, for the values of the parameters and initial state as in Fig. 4, we obtain the results for the occupancy measure varying over time shown in Fig. 5, which is the plot of the limit *o.m.v.* $\boldsymbol{\mu}(\tau)$ produded by FlyFast.

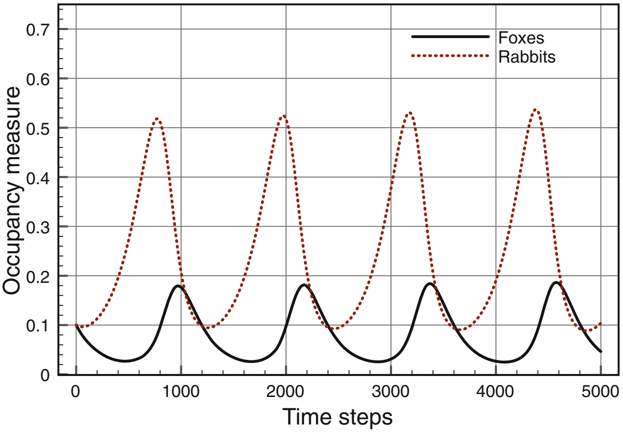


Fig. 5. Fraction of rabbit and fox populations.

In the predator-prey model one could furthermore be interested to know what is the probability that a rabbit survives for a certain amount of time, and how this probability changes over time with the oscillation of the population of foxes. Figure 6 shows the probability that a fox gets born or dies within time bound t

ranging from 0 to 3000 time steps. It also shows the results for a rabbit getting born or dying. The formula for the probability that a rabbit gets born within t time steps is $\mathcal{P}_{=?}(\texttt{RD}\ \mathcal{U}^{\leq t}\ \texttt{RL})$. The other formulas are similar. Figure 6 shows that both foxes and rabbits eventually get born and die when given enough time and starting from the initial state of the overall system. The curves also reflect the oscillations in the populations over time and consequently the change in probability to get born or die.

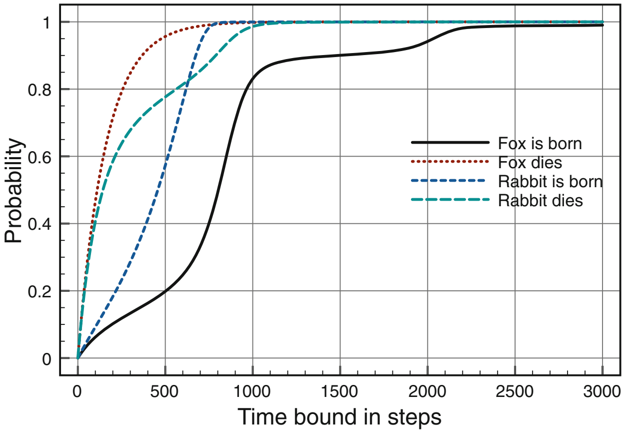


Fig. 6. Probability that a fox (rabbit) gets born or dies within time bound t ranging from 0 to 3000 time steps.

Figure 7 shows the time-dependent probability of a fox and a rabbit to get born or die in the next 10 time steps, starting from initial times ranging from 0 to 5000. The probability that a rabbit dies within 10 time units is obtained by evaluating the property $\mathcal{P}_{=?}(\texttt{RL}\ \mathcal{U}^{\leq 10}\ \texttt{RD})$, for different initial times. The other formulas are similar. In this oscillating system the time-dependence of these probabilities can be observed very well. The probabilities of a rabbit getting born (dying respectively) within 10 time units and a fox getting born follow closely the oscillations in the respective population sizes. The probability that a fox dies in this model is constant. The amplitude of the oscillations is slowly increasing. This is likely due to the accumulation of small errors in the computation due to the constant step size used in the computations. In fact it is well-known that a mean-field approximation may become less accurate on the longer run, in a discrete time setting. See e.g. [19] for a study of this aspect of the Lotka-Volterra model in the continuous time setting. We will come back to these issues in Sect. 4.1.

Finally, Fig. 8 shows the time dependent probability of reaching a state, within 100 time steps, in which the probability of an individual rabbit to die within 10 time steps is higher than 0.2. This probability is shown for different

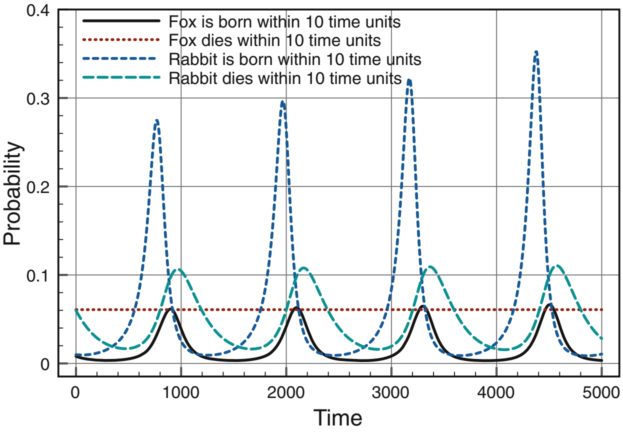


Fig. 7. Time dependent probability to get born or die in the next 10 time steps for different initial times from 0 to 5000.

initial times ranging from 0 to 5000. This is a typical example of a 'nested' formula involving two occurrences of the until operator. The formula is:

$$\mathcal{P}_{=?}(\texttt{tt}\ \mathcal{U}^{\leq 100}\ (\ \texttt{RL}\ \wedge\ \mathcal{P}_{>0.2}(\ \texttt{RL}\ \mathcal{U}^{\leq 10}\ \texttt{RD})))$$

The figure shows that there are indeed relatively short periods in which such states can be reached within 100 time steps. Nested formulas are relatively easy to handle due to the iterative and recursive way in which the FlyFast model-checker works.

4 Extending the Applicability of FlyFast

In the previous sections we have shown examples of the expected use of FlyFast, namely the development of a probabilistic, discrete time, population model of the system of interest and its analysis, mainly via bounded PCTL model-checking based on mean-field semantics. In this section we briefly describe two extensions of the applicability of the tool, both designed as additional *front-ends* for FlyFast, so that no modifications are required of the tool itself. The first extension concerns on-the-fly fluid CSL model-checking of *continuous* time population models; the second extends the FlyFast modelling language, and its underlying agent interaction paradigm, by adding *predicate* based communication primitives. Details on the first extension can be found in [41] while the second front end is described in detail in [15,44].

4.1 FlyFast Front-end for Fluid Model-checking of Continuous Time Population Models

Fluid model checking [7,8,33] relies on a global model checking approach for time-inhomogeneous Continuous Time Markov Chains (ICTMC) representing

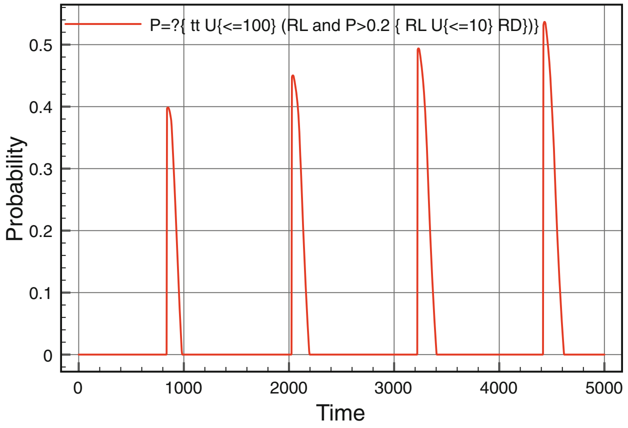


Fig. 8. Time dependent probability of reaching a state, within 100 time steps, in which the probability of an individual rabbit to die within 10 time steps is less than 0.2 for different initial times ranging from 0 to 5000.

an individual object in the context of a large CTMC population model. The rates of the individual may depend on the fraction of the population that is in a particular state. The algorithm relies on the deterministic approximation of the average stochastic behaviour of the system in *continuous time*, i.e. a fluid approximation [9,35]. Although the technical and mathematical foundations of the continuous time case are obviously different from those in the discrete case, at the intuitive/conceptual level, the two cases are similar.

Suppose you have system of N agents, each modelled by a ICTMC with states in $\{1,\ldots,S\}$, and $S \times S$ infinitesimal generator matrix $\mathbf{Q}^{(N)}(\boldsymbol{x})$ that may depend on the current *o.m.v.* $\boldsymbol{x} \in \mathcal{U}^S$; the *o.m.v.* process is a CTMC on the space $[0,1]^S$ with initial state $\boldsymbol{x}_0^{(N)}$ equal to the fraction of agents in each local state, in the initial global state. The average infinitesimal variation of the *o.m.v.* process, given that it is in state $\boldsymbol{x}$ is $F^{(N)}(\boldsymbol{x}) = \boldsymbol{x}^T \cdot \mathbf{Q}^{(N)}(\boldsymbol{x})$. If, for $N \rightarrow \infty$, $\mathbf{Q}^{(N)}(\boldsymbol{x})$ converges uniformly to the Lipschitz continuous generator matrix $\mathbf{Q}(\boldsymbol{x})$, and $\boldsymbol{x}_0^{(N)}$ to $\boldsymbol{x}_0$, and, furthermore, if $\boldsymbol{x}(t)$ is the solution of the ODE $\frac{d\boldsymbol{x}}{dt} = F(\boldsymbol{x}) = \boldsymbol{x}^T \cdot \mathbf{Q}(\boldsymbol{x})$ for initial condition $\boldsymbol{x}(0) = \boldsymbol{x}_0$, then, almost surely, in the limit, the *o.m.v.* process behaves the same as $\boldsymbol{x}(t)$, for any finite time horizon T [18,35].

This fundamental result has given rise to a *fast simulation* approach also in the continuous case. Assuming, again by convention and without loss of generality, that we are interested in the first of the N agents, let $Z^{(N)}(t)$ be the ICTMC on space $\{1,\ldots,S\}$ modelling the behaviour of such an agent. Let us furthermore consider the ICTMC on $\{1,\ldots,S\}$ $z(t)$ such that $\Pr\{z(t+dt) = j | z(t) = i\} = q_{i,j}(\boldsymbol{x}(t))dt$, and let $\mathbf{Q}_z(\boldsymbol{x}(t)) = (q_{i,j}(\boldsymbol{x}(t)))$. We then have that for any finite horizon T and $t \leq T$ the behaviour of the single object $Z^{(N)}(t)$ tends to the behaviour of the object that senses the rest of the system only through its limit behaviour given by $\boldsymbol{x}$, i.e. $z(t)$. On the basis of these results, in [7,8] a model-checking algorithm has been proposed for CSL robust[10] formulas.

In [41] we took an alternative approach, showing that, under suitable convergence and scaling assumptions[11], and for models that are not too stiff[12], fluid model checking can be performed exploiting on-the-fly mean field model checking. In particular, in [41] a mechanical translation is defined which derives a time-inhomogeneous DTMC and a bounded PCTL formula from the input ICTMC model and bounded CSL formula. FlyFast can then be used for performing on-the-fly mean-field model-checking of the derived formula on the derived IDTMC.

Our approach starts from the idea that we can interpret the difference equations obtained from a discrete time population model as an instance of the Euler forward method for approximating the solution of a set of ODEs. The set of ODEs we are interested in solving are those of a corresponding continuous population model. This, in turn, means that we need to derive suitable values for the probabilities from the rates in the continuous model. What we are actually interested in is to transform an ICTMC model of an individual (from which the ODEs can be derived) into an IDTMC model, with the same local states and jump structure as the ICTMC; from this IDTMC we get the set of difference equations that can be used to approximate the solution of the ODEs. Intuitively, the IDTMC is obtained from the ICTMC using an approach which is similar to CTMC uniformisation[13]; we define a probability matrix $\mathbf{K}$ such that $\mathbf{K} = \mathbf{I} + \frac{1}{q} \cdot \mathbf{Q}$, where $\mathbf{Q}$ is the infinitesimal rate matrix—which is a function of $\boldsymbol{x}(t)$—and q must not only satisfy the standard requirements for uniformisation, but also be such that absolute stability of the method as well as acceptable accuracy are guaranteed [47]. This procedure produces a discretisation of the continuous-time model; of course, also the logical formulas must be translated by consistently discretising them—and in particular the time bound of the bounded until operator.

We refer to [41] for the detailed definition of the translations and their correctness proof. Here we point out that such global fluid model checking algorithms, as described in [7,8], require the *a priori* calculation of discontinuity points, i.e. points in time in which the truth values of time-dependent (sub)-formulas of an until formula change. This is a non-trivial task and consists in finding all zeros of an analytic function. In the on-the-fly setting, instead, such points are detected automatically during the computation of the probabilities, up to a difference that is in the order of a small discrete step size; moreover, on-the-fly approaches are particularly efficient when verifying conditional reachability properties because in that case much fewer states need to be generated.

[10] We refer to [7,8] for the definition of formula robustness.

[11] See Theorem 5 of [7].

[12] Stiff models are those whose rates differ several orders of magnitude.

[13] More specifically, we use only the discretisation phase of uniformisation, and not the transient analysis part, that would require a further composition with a Poisson process.

On the other hand, our approach is ultimately based on an Euler forward method to solve differential equations. This poses certain limitations on the continuous time models that can be analysed efficiently this way, in particular they should not be too stiff. For non stiff models the results are promising as shown in [41] for the available benchmark models for which also some results for global fluid model checking and statistical model checking are available in the literature.

4.2 FlyFast Front-End for Predicate-Based Coordination

Recent proposals for CAS modelling and programming languages, like [6,20], typically assume any such a system be composed of a set of independent *components* where a component is a process equipped also with a set of *attributes* describing features of the component. Attributes can be used in *predicates* appearing in the language input/output primitives. *Predicate-based* output/input *multicast*, originally proposed in [45], forms the basis of interaction schemes in languages like SCEL [20] and Carma [6]. In [15] we proposed PiFF—Predicate-based Interaction for FlyFast—a front-end modelling language for FlyFast inspired by Carma, that provides *predicate-based* input/output *multicast* actions.

In PiFF, each component consists of a behaviour—modelled, like in FlyFast, as a DTMC-like agent—and a set of attributes. The attribute name-value correspondence is kept in the current *store* of the component. Associated to each action there is also an (atomic) probabilistic store-update. For instance, assume components have an attribute named loc which takes values in the set of points of a space, thus recording the current location of the component. The following action models a multi-cast via channel α to all components in the same location as the sender, making the latter change its location randomly: $\alpha^*[\mathsf{loc} = \mathbf{my}.\mathsf{loc}]\langle\rangle\mathtt{Jump}$. Here Jump is assumed to randomly update the store and, in particular attribute loc. The computational model of PiFF is *clock-synchronous*, as in FlyFast, but at the component level. In addition, each component is equipped with a local *outbox*. The effect of an output action $\alpha^*[\pi_r]\langle\rangle\sigma$ is to deliver output label $\alpha\langle\rangle$ to the local outbox, together with the predicate π_r, which (the store of) the receiver components will be required to satisfy, as well as the current store of the component executing the action; the current store is then updated according to update σ. Note that output actions are *non-blocking* and that successive output actions of the same component overwrite its outbox. An input action $\alpha^*[\pi_s]()\sigma$ by a component will be executed with a probability which is proportional to the *fraction* of all those components whose outboxes currently contain the label $\alpha\langle\rangle$, a predicate π_r which is satisfied by the component, and a store which satisfies predicate π_s in turn. If such a fraction is zero, then the input action will not take place (input is blocking), otherwise the action takes place, the store of the component is updated via σ, and its outbox cleared.

A PiFF model specification is compiled into a FlyFast model specification by means of a (purely mechanical) translation and related bounded PCTL formulas are mechanically translated as well. For the sake of simplicity, we do not describe the translation here; the interested reader can find its definition in [15], where the formal stochastic semantics of PiFF are also given and the translation is shown correct with respect to such semantics; optimisation of the translation is dealt with in [44]. In particular, in [44], a bisimilarity based state-reduction strategy for the target model specification is proposed.

5 Conclusions

Model-checking has proven to be an effective and successful formal verification technique. Initially developed for *qualitative* models and logics, it has been extended also to quantitative models and logics such as DTMCs and PCTL as well as CTMCs and CSL. It is well known that model-checking suffers from the state-space explosion problem, which makes the technique non-scalable and thus poorly applicable to large scale systems. On the other hand, current trends in information technology, like the Internet of Things, include systems composed of a large number of components, often acting collectively and adapting to changing conditions, the so called *Collective Adaptive Systems*. In this paper we have briefly described the work we have been carrying out in the area of approximated bounded PCTL model-checking of Population DTMC models. In particular we have given an introductory description of FlyFast, a mean-field, on-the-fly bounded PCTL model-checker, including an overview of its theoretical foundation, its main functionalities and a detailed example of application. A couple of extensions of the applicability of the tool have been shown as well, in the form of specific additional front-ends to the original tool; thus, the tool applicability is extended without actually modifying the tool.

There are several lines of future work of our interest. First of all, following approaches similar to those presented in [46], we plan to investigate the extension of the model-checking technique to systems with memory/rewards. Space and the spatial distribution of agents play a major role in CAS and, consequently, it should be a “first class” component of the modelling language and the underlying framework. For this reason, we have investigated *Closure Spaces*, a generalisation of Topological Spaces that includes discrete, graph-like, space structures, for which we have developed the *Spatial Logic for Closure Spaces*, SLCS and a specific model-checking algorithm [13,14]. A subject for future research is thus to incorporate a notion of space in the FlyFast modelling language and to integrate FlyFast and `topochecker`, the spatial model-checker for SLCS and its extensions. The investigation of different classes of interaction probability specifications in the FlyFast modelling language and of their implications on issues like model-reduction (see e.g. [44]) is also a promising subject for future research.

Acknowledgments. In the late 80’s of the previous century, Diego met Ed, who was chairing a Work Package of the EU Lotosphere project, in which Diego participated as well. At that time, Diego was fascinated by the early work on probabilistic process algebras by Scott Smolka, Kim Larsen and others and he was applying similar ideas to LOTOS, together with Paola Quaglia. At the same time, he was loving the work of Rom, supervised by Ed, on Bundle Event Structures as a mathematical model underlying

a truly concurrent semantics for LOTOS. The obvious step was to start thinking of probabilistic extensions of Bundle Event Structures. Accidentally, Diego and Mieke had met at a Lotosphere workshop in The Hague and they found themselves nicely synchronised in their professional interests, and beyond ...

Not surprisingly, Diego moved to Twente where he spent 12 months, from july 1992 to june 1993, and together with Ed, Rom and Joost-Pieter, started investigating probabilistic, deterministically timed and stochastically timed Bundle Event Structures. This was the start of a lively friendship of the four of them as well as of a series of headaches when trying to find finite graph-like representations of such structures suitable for analysis. They have been struggling together for years, searching for *cut-off* events in those slippery structures. Eventually, Mieke moved to Italy and joined the group of cut-off events hunters. It was fun! Maybe we did not manage to completely master the analysis of quantitative Bundle Event Structures, but we are aware of a couple of things: our current work on probabilistic systems is rooted back to those days (and headaches ...) and our friendship too. All this thanks to Ed, who accepted having Diego around in Twente in 1992–93.

References

1. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model checking continuous time Markov chains. ACM Trans. Comput. Log. **1**(1), 162–170 (2000)
2. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. IEEE Trans. Softw. Eng. IEEE CS **29**(6), 524–541 (2003)
3. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, Cambridge (2008)
4. Bernardo, M., De Nicola, R., Hillston, J. (eds.): Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems. LNCS, vol. 9700. Springer, Heidelberg (2016). ISBN 978-3-319-34095-1 (print), 978-3-319-34096-8 (online), ISSN 0302-9743(2016)
5. Bhat, G., Cleaveland, R., Grumberg, O.: Efficient on-the-fly model checking for CTL*. In: LICS, pp. 388–397. IEEE Computer Society (1995)
6. Bortolussi, L., De Nicola, R., Galpin, V., Gilmore, S., Hillston, J., Latella, D., Loreti, M., Massink, M.: CARMA: collective adaptive resource-sharing Markovian agents. In: Bertrand, N., Tribastone, M. (eds.) Proceedings Thirteenth Workshop on Quantitative Aspects of Programming Languages and Systems, QAPL 2015, London, UK, 11th–12th April 2015, EPTCS, vol. 194, pp. 16–31 (2015). http://dx.doi.org/10.4204/EPTCS.194.2
7. Bortolussi, L., Hillston, J.: Fluid model checking. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 333–347. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32940-1_24
8. Bortolussi, L., Hillston, J.: Model checking single agent behaviours by fluid approximation. Inf. Comput. **242**, 183–226 (2015)
9. Bortolussi, L., Hillston, J., Latella, D., Massink, M.: Continuous approximation of collective system behaviour: a tutorial. Perform. Eval. **70**(5), 317–349 (2013). http://www.sciencedirect.com/science/article/pii/S0166531613000023
10. Bradley, J.T., Gilmore, S.T., Hillston, J.: Analysing distributed internet worm attacks using continuous state-space approximation of process algebra models. J. Comput. Syst. Sci. **74**(6), 1013–1032 (2008)

11. Buchholz, P., Hahn, E.M., Hermanns, H., Zhang, L.: Model checking algorithms for CTMDPs. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 225–242. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_19
12. Chaintreau, A., Le Boudec, J.Y., Ristanovic, N.: The age of gossip: spatial mean field regime. In: Douceur, J.R., Greenberg, A.G., Bonald, T., Nieh, J. (eds.) SIGMETRICS/Performance, pp. 109–120. ACM (2009)
13. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Specifying and verifying properties of space. In: Diaz, J., Lanese, I., Sangiorgi, D. (eds.) TCS 2014. LNCS, vol. 8705, pp. 222–235. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44602-7_18. ISBN 978-3-662-44601-0 (print), 978-3-662-44602-7 (online), ISSN 0302-9743
14. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Model checking spatial logics for closure spaces. Log. Methods Comput. Sci. **12**(4), 1–51 (2016). doi:10.2168/LMCS-12(4:2)2016. Published online: 11 October 2016. ISSN 1860-5974
15. Ciancia, V., Latella, D., Massink, M.: On-the-fly mean-field model-checking for attribute-based coordination. In: Lluch Lafuente, A., Proença, J. (eds.) COORDINATION 2016. LNCS, vol. 9686, pp. 67–83. Springer, Cham (2016). doi:10.1007/978-3-319-39519-7_5. ISSN 0302-9743, ISBN 978-3-319-39518-0 (print), 978-3-319-39519-7 (online)
16. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. **8**(2), 244–263 (1986)
17. Courcoubetis, C., Vardi, M., Wolper, P., Yannakakis, M.: Memory-efficient algorithms for the verification of temporal properties. Form. Methods Syst. Des. **1**(2–3), 275–288 (1992)
18. Darling, R., Norris, J.: Differential equation approximations for Markov chains. Probab. Surv. **5**, 37–79 (2008)
19. Dayar, T., Mikeev, L., Wolf, V.: On the numerical analysis of stochastic Lotka-Volterra models. In: IMCSIT, pp. 289–296 (2010)
20. De Nicola, R., et al.: The SCEL language: design, implementation, verification. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems. LNCS, vol. 8998, pp. 3–71. Springer, Cham (2015). doi:10.1007/978-3-319-16310-9_1. ISBN 978-3-319-16309-3 (print), 978-3-319-16310-9 (online), ISSN 0302-9743
21. Della Penna, G., Intrigila, B., Melatti, I., Tronci, E., Zilli, M.V.: Bounded probabilistic model checking with the Murα verifier. In: Hu, A.J., Martin, A.K. (eds.) FMCAD 2004. LNCS, vol. 3312, pp. 214–229. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30494-4_16
22. Din, Q.: Dynamics of a discrete Lotka-Volterra model. Adv. Diff. Equ. **95**, 1–13 (2013)
23. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 264–279. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39799-8_19
24. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15297-9_9
25. Gast, N., Gaujal, B.: A mean field model of work stealing in large-scale systems. In: Misra, V., Barford, P., Squillante, M.S. (eds.) SIGMETRICS, pp. 13–24. ACM (2010)

26. Gnesi, S., Mazzanti, F.: An abstract, on the fly framework for the verification of service-oriented systems. In: Wirsing, M., Hölzl, M. (eds.) Rigorous Software Engineering for Service-Oriented Systems. LNCS, vol. 6582, pp. 390–407. Springer, Heidelberg (2011). doi:10.1007/978-3-642-20401-2_18
27. Goel, N.S., Maitra, S.C., Montroll, E.W.: On the volterra and other nonlinear models of interacting populations. Rev. Mod. Phys. **43**, 231–276 (1971). http://link.aps.org/doi/10.1103/RevModPhys.43.231
28. Guirado, G., Hérault, T., Lassaigne, R., Peyronnet, S.: Distribution, approximation and probabilistic model checking. In: PDMC 2005. LNCS, vol. 135. pp. 19–30. Springer, Heidelberg (2006)
29. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: INFAMY: an infinite-state Markov model checker. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 641–647. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02658-4_49
30. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Comput. **6**, 512–535 (1994)
31. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic Model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 73–84. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24622-0_8
32. Holzmann, G.J.: The SPIN Model Checker - Primer and Reference Manual. Addison-Wesley, Boston (2004)
33. Kolesnichenko, A., de Boer, P.T., Remke, A., Haverkort, B.R.: A logic for model-checking mean-field models. In: DSN13 (2013)
34. Kolesnichenko, A.V., Remke, A.K.I., de Boer, P.T., Haverkort, B.: A logic for model-checking of mean-field models. Technical report TR-CTIT-12-11 (2012). http://doc.utwente.nl/80267/
35. Kurtz, T.: Solutions of ordinary differential equations as limits of pure jump Markov processes. J. Appl. Probab. **7**, 49–58 (1970)
36. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic symbolic model checking using PRISM: a hybrid approach. STTT **6**(2), 128–142 (2004)
37. Larsen, K.G., Legay, A.: Statistical model checking: past, present, and future. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 3–15. Springer, Cham (2016). doi:10.1007/978-3-319-47166-2_1
38. Latella, D., Loreti, M., Massink, M.: On-the-fly fast mean-field model-checking. In: Abadi, M., Lluch Lafuente, A. (eds.) TGC 2013. LNCS, vol. 8358, pp. 297–314. Springer, Cham (2014). doi:10.1007/978-3-319-05119-2_17
39. Latella, D., Loreti, M., Massink, M.: On-the-fly PCTL fast mean-field model-checking for self-organising coordination - preliminary version. Technical report TR-QC-01-2013, QUANTICOL (2013)
40. Latella, D., Loreti, M., Massink, M.: On-the-fly probabilistic model checking. In: Lanese, I., Sokolova, A. (eds.) Proceedings of the 7th Interaction and Concurrency Experience (ICE 2014), 6 June 2014, Berlin, Germany. EPTCS, vol. 166, pp. 45–59 (2014). doi:10.4204/EPTCS.166.6, http://cgi.cse.unsw.edu.au/~rvg/eptcs/, ISSN 2075-2180
41. Latella, D., Loreti, M., Massink, M.: On-the-fly fluid model checking via discrete time population models. In: Beltrán, M., Knottenbelt, W., Bradley, J. (eds.) EPEW 2015. LNCS, vol. 9272, pp. 193–207. Springer, Cham (2015). doi:10.1007/978-3-319-23267-6_13. ISSN 0302-9743
42. Latella, D., Loreti, M., Massink, M.: On-the-fly PCTL fast mean-field approximated model-checking for self-organising coordination. Sci. Comput. Program. **110**, 23–50 (2015). doi:10.1016/j.scico.2015.06.009, ISSN 0167-6423

43. Latella, D., Loreti, M., Massink, M.: FlyFast: a mean field model checker. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 303–309. Springer, Heidelberg (2017). doi:10.1007/978-3-662-54580-5_18. ISSN 0302-9743
44. Latella, D., Massink, M.: Design and optimisation of the FlyFast front-end for attribute-based coordination. In: de Vink, E.P., Wiklicky, H. (eds.) Proceedings of the Fifteenth Workshop on Quantitative Aspects of Programming Languages (QAPL 2017). Electronic Proceedings in Theoretical Computer Science, EPTCS (2017, to appear). Available also as QUANTICOL Technical report TR-QC-01-2017 (2017)
45. Latella, D.: Comunicazione basata su proprietà nei sistemi decentralizzati [property-based inter-process communication in decentralized systems], December 1983. Graduation thesis. Istituto di Scienze dell'Informazione. Univ. of Pisa (in italian)
46. Le Boudec, J.Y., McDonald, D., Mundinger, J.: A generic mean field convergence result for systems of interacting objects. In: QEST07, pp. 3–18. IEEE Computer Society Press (2007). ISBN 978-0-7695-2883-0
47. LeVeque, R.J.: Finite Difference Methods for Ordinary and Partial Differential Equations. SIAM, Philadelphia (2007)
48. Lotka, A.J.: Elements of Mathematical Biology. Williams and Wilkins Company, Philadelphia (1924)
49. de Oca, M.A.M., Ferrante, E., Scheidler, A., Pinciroli, C., Birattari, M., Dorigo, M.: Majority-rule opinion dynamics with differential latency: a mechanism for self-organized collective decision-making. Swarm Intell. **5**(3–4), 305–327 (2011)
50. Nenzi, L., Bortolussi, L., Ciancia, V., Loreti, M., Massink, M.: Qualitative and quantitative monitoring of spatio-temporal properties. In: Bartocci, E., Majumdar, R. (eds.) RV 2015. LNCS, vol. 9333, pp. 21–37. Springer, Cham (2015). doi:10.1007/978-3-319-23820-3_2
51. Volterra, V.: Fluctuations in the abundance of a species considered mathematically. Nature **118**, 558–560 (1926)