Design and Optimisation of the FlyFast Front-end for Attribute-based Coordination

Diego Latella Mieke Massink
Consiglio Nazionale delle Ricerche
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Diego.Latella@isti.cnr.it, Mieke.Massink@isti.cnr.it

Collective Adaptive Systems (CAS) consist of a large number of interacting objects. The design of such systems requires scalable analysis tools and methods, which have necessarily to rely on some form of approximation of the system's actual behaviour. Promising techniques are those based on mean-field approximation. The FlyFast model-checker uses an on-the-fly algorithm for bounded PCTL model-checking of selected individual(s) in the context of very large populations whose global behaviour is approximated using deterministic limit mean-field techniques. Recently, a front-end for FlyFast has been proposed which provides a modelling language, PiFF in the sequel, for the *Predicate-based* Interaction for FlyFast. In this paper we present details of PiFF design and an approach to state-space reduction based on probabilistic bisimulation for inhomogeneous DTMCs.

1 Introduction

Collective Adaptive Systems (CAS) consist of a large number of entities with decentralised control and varying degrees of complex autonomous behaviour. They form the basis of many modern *smart city* critical infrastructures. Consequently, their design requires support from formal methods and scalable automatic tools based on solid mathematical foundations. In [29, 27], Latella et al. presented a scalable mean-field model-checking procedure for verifying bounded Probabilistic Computation Tree Logic (PCTL, [19]) properties of an individual[1] in the context of a system consisting of a large number of interacting objects. The model-checking procedure is implemented in the tool FlyFast[2]. The procedure performs on-the-fly, mean-field, approximated model-checking based on the idea of fast simulation, as introduced in [8]. More specifically, the behaviour of a generic agent with S states in a system with a *large* number N of instances of the agent at given step (i.e. time) t is approximated by $\mathbf{K}(\mu(t))$ where $\mathbf{K}(\mathbf{m})$ is the $S \times S$ probability transition matrix of an (inhomogeneous) DTMC and $\mu(t)$ is a vector of size S approximating the mean behaviour of (the rest of) the system at t; each element of $\mu(t)$ is associated with a distinct state of the agent, say C, and gives an approximation of the fraction of instances of the agent that are in state C in the global system, at step t. Note that such an approximation is a *deterministic* one, i.e. μ is a *function* of the step t (the exact behaviour of the rest of the system would instead be a *large* DTMC in turn); note furthermore, that the above transition matrix does not depend on N [29, 27].

Recently, modelling and programming languages have been proposed specifically for autonomic computing systems and CAS [12, 5]. Typically, in such frameworks, a system is 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. The attributes of a component can be *updated* during its execution so

[1] 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.

[2] http://j-sam.sourceforge.net/

De Vink and Wiklicky (Eds.): QAPL 2017
EPTCS 250, 2017, pp. 92–110, doi:10.4204/EPTCS.250.6

that the association between attribute *names* and attribute *values* is maintained in the dynamic *store* of the component. Attributes can be used in *predicates* appearing in language constructs for component interaction. The latter is thus typically modelled using *predicate-based* output/input *multicast*, originally proposed in [26], and playing a fundamental role in the interaction schemes of languages like SCEL [12] and CARMA [5]. In fact, predicate-based communication can be used by components to dynamically organise themselves into ensembles and as a means to dynamically select partners for interaction. Furthermore, it provides a way for representing component features, like for instance component location in space, which are fundamental for systems distributed in space, such as CAS [30].

In [9] we proposed a front-end modelling language for FlyFast that provides constructs for dealing with *components* and *predicate-based interaction*; in the sequel, the language—which has been inspired by CARMA— will be referred to as PiFF, which stands for for Predicate-based Interaction for FlyFast. Components interact via predicate-based communication. Each component consists of a behaviour, modelled as a DTMC-like agent, like in FlyFast, and a set of attributes. The attribute name-value correspondence is kept in the current store of the component. Actions are *predicate based multi-cast* output and input primitives; predicates are defined over attributes. 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 it change 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 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 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 in turn predicate π_s. 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.

Related Work CAS are typically *large* systems, so that the formal analysis of models for such systems hits invariantly the state-space explosion problem. In order to mitigate this problem, the so called 'on-the-fly' paradigm is often adopted (see e.g. [10, 4, 21, 16]).

In the context of probabilistic model-checking several on-the-fly approaches have been proposed, among which [13], [28] and [18]. In [13], a probabilistic model-checker is shown for the *time bounded* fragment of PCTL. An on-the-fly approach for *full* PCTL model-checking is proposed in [28] 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 [18] 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 [20, 17] 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 [25]. 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. As recalled above, in [27] a scalable model-checking algorithm is presented that is based on mean-field approximation, for the verification of time bounded PCTL properties of an individual in the context of a system consisting of a large number of interacting objects. Correctness of the algorithm with respect to exact probabilistic model-checking has been proven in [27] as well. Also this algorithm is actually an instantiation of the above mentioned parametric algorithm for (exact) probabilistic model-checking [28], but the algorithm is instantiated on (time bounded PCTL and) the *approximate*, mean-field, semantics of a population process modelling language. It is worth pointing out that FlyFast allows users to perform simulations of their system models and to analyse the latter using their *exact* probabilistic semantics and *exact* PCTL model-checking. In addition, the tool provides *approximate* model-checking for bounded PCTL, using the model semantics based on mean-field.

The work of Latella et al. [27] 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, where 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 [23], but no model-checking algorithms were presented. Follow-up work on the above mentioned approach can be found in [24] which relies on earlier results on fluid model-checking by Bortolussi and Hillston [6], later published in [7], where a *global CSL* model-checking procedure is proposed for the verification of properties of a selection of individuals in a population, which relies on fast simulation results. This work is perhaps closest related to [27, 29]; however their procedure exploits mean-field convergence and fast simulation [11, 15] in a *continuous* time setting—using a set of differential equations—rather than in a discrete time setting—where an inductive definition is used. Moreover, that approach 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.

PiFF has been originally proposed in [9], where the complete formal, exact probabilistic, semantics of the language have been defined. The semantics definition consists of three transition rules—one for transitions associated with output actions, one for those associated with input actions, and one for transitions to be fired with residual probability. The rules induce a transition relation among component states and compute the relevant probabilities. From the component transition relation, a component one-step transition probability matrix is derived, the elements of which may depend on the fractions of the components in the system which are in a certain state. The system-wide one-step transition probability matrix is obtained by product—due to independence assumptions—using the above mentioned component probability matrix and the actual fractions in the current system global state. In [9] a translation of PiFF to the model specification language of FlyFast has also been presented which makes PiFF an additional front-

end for FlyFast extending its applicability to models of systems based on predicate-based interaction. In the above mentioned paper, correctness of the translation has been proved as well. In particular, it has been shown that the probabilistic semantics of any PiFF model are isomorphic to those of the translation of the model. In other words, the transition probability matrix of (the DTMCs of) the two models is the same. A companion translation of bounded PCTL formulas is also defined [9] and proven correct.

The notion of the outbox used in PiFF is reminiscent of the notion of the *ether* in PALOMA [14] in the sense that the collection of all outboxes together can be thought of as a kind of ether; but such a collection is intrinsically distributed among the components so that it cannot represent a bottleneck in the execution of the system neither a singularity point in the deterministic approximation.

We are not aware of other proposals, apart from [9], of probabilistic process languages, equipped both with standard, DTMC-based, semantics and with mean-field ones, that provide a predicate-based interaction framework, and that are fully supported by a tool for probabilistic simulation, exact and mean-field model-checking.

We conclude this section recalling 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, possibly more accurate but often not as scalable, analysis techniques for CAS, primarily those based on stochastic simulation, such as statistical model-checking.

In this paper we present some details of PiFF, a translation to FlyFast which simplifies that proposed in [9] and an approach to model reduction based on probabilistic bisimulation for Inhomogeneous DTMCs. In Section 2 we briefly present the main ingredients of the PiFF syntax and informal semantics, and we recall those features of FlyFast directly relevant for understanding the translation of PiFF to the FlyFast input language proposed in [9]. A revised and simplified version of the translation is described in Section 3. In Section 4 we introduce a simplified language for the definition of transition-probabilities in PiFF that allows us to define in Section 5 a model reduction procedure of the translation result, based on a notion of bisimulation for the kind of IDTMCs of interest, introduced in Section 5 as well. An example of application of the procedure is presented in Section 6. Some conclusions are drawn in Section 7. A formal proof of decidability of the cumulative probability test for state-space reduction based on bisimulation is provided in the Appendix.

2 Summary on PiFF and FlyFast

In the following we present the main ingredients of PiFF and the features of FlyFast relevant for the present paper.

2.1 PiFF

A PiFF system model specification $\Upsilon = (\Delta_\Upsilon, F_\Upsilon, \Sigma_0)^{(N)}$ is a triple where F_Υ is the set of relevant function definitions (e.g. store updates, auxiliary constants and functions), Δ_Υ is a set of state defining equations, and Σ_0 is the initial system state (an N-tuple of component states, each of which being a 3-tuple (C, γ, O) of agent state C, store γ and outbox O). We describe the relevant details below referring to [9] for the formal definition probabilistic semantics of the language.

The PiFF type system consists of floating point values and operations, as in FlyFast, plus simple enumeration types for attributes, declared according to the syntax **attype** $<$ `name` $>$ **enum** $<$ `id` $-$ `list` $>$. $<$ `id` $-$ `list` $>$ is a finite list of identifiers. Of course, attributes can also take floating point values.

In Figure 1 the attribute type `Space` is defined that consists of four values `A`,`B`,`C`,`D` modelling four

```
attype Space enum A,B,C,D;
⋮
const H = 0.6;
const L = 1 − H;
const Hdiv2 = H/2;
const Ldiv2 = L/2;
⋮
attribute loc : Space;
⋮
func Hr(x : Space) : Space; x endfunc;
func N(x : Space) : Space; case x of A : A;B : B;C : B;D : A endfunc;
func S(x : Space) : Space; case x of A : D;B : C;C : C;D : D endfunc;
func E(x : Space) : Space; case x of A : A;B : A;C : D;D : D endfunc;
func W(x : Space) : Space; case x of A : B;B : B;C : C;D : C endfunc;
⋮
func pHr(x : Space) : float; case x of A : H;B : L;C : H;D : L endfunc;
func pN(x : Space) : float; case x of A : 0;B : 0;C : Ldiv2;D : Hdiv2 endfunc;
func pS(x : Space) : float; case x of A : Ldiv2;B : Hdiv2;C : 0;D : 0 endfunc;
func pE(x : Space) : float; case x of A : 0;B : Hdiv2;C : Ldiv2;D : 0 endfunc;
func pW(x : Space) : float; case x of A : Ldiv2;B : 0;C : 0;D : Hdiv2 endfunc;
⋮
update Jump
my.loc := Hr(my.loc) with pHr(my.loc);
my.loc := N(my.loc) with pN(my.loc);
my.loc := S(my.loc) with pS(my.loc);
my.loc := E(my.loc) with pE(my.loc);
my.loc := W(my.loc) with pW(my.loc)
endupdate
```

Figure 1: A fragment of F_{SI}.

locations. Some auxiliary constants are defined, using the **const** construct inherited from FlyFast: **const** $<$ `name` $>$ $=$ $<$ `value` $>$.

A PiFF store update definition has the following syntax[3]:

update *upd*
my.$a_1 := e_{11}, \ldots,$ **my**.$a_k := e_{k1}$ **with** p_1;
$\vdots$
my.$a_1 := e_{1n}, \ldots,$ **my**.$a_k := e_{kn}$ **with** p_n
endupdate

where *upd* is the update name (unique within the system model specification), $a_1, \ldots, a_k$ are the attribute names of the component, $e_{11}, \ldots, e_{kn}$ and $p_1, \ldots, p_n$ are attribute/store-probability expressions respectively, with syntax defined according to the grammars $e ::= v_a \mid c_a \mid$ **my**.$a \mid fn_a(e_1, \ldots, e_m)$ and $p ::= v_p \mid c_p \mid fn_p(e_1, \ldots, e_m)$. In the above definition of attribute expressions v_a is an attribute value (drawn from finite set $\mathscr{V}$ of attribute values), c_a is an attribute constant in $\mathscr{V}$ defined using the **const**; $a \in \{a_1, \ldots, a_k\}$ is an attribute name and fn_a is an attribute function defined by the user in F_Υ, which, when applied to attribute expressions $e_1, \ldots, e_m$ returns an attribute value; the syntax for such function definitions *afd* is given below:

afd ::= **func** $fn_a(x_1 : T1, \ldots, x_m : Tm) : T$;*afb* **endfunc**
afb ::= e |**case** $(x_1, \ldots, x_m)$ **of**$(v_{a_{11}}, \ldots, v_{a_{m1}}) : e_1; (v_{a_{12}}, \ldots, v_{a_{m2}}) : e_2; \ldots (v_{a_{1k}}, \ldots, v_{a_{mk}}) : e_k$

where fn_a is the name of the attribute function, $x_1 : T1, \ldots, x_m : Tm$ are its parameters and their relative types, T is the type of the result of fn_a; e, e_i are attribute-expressions and $v_{a_{ij}}$ are attribute-values.

In Figure 1 attribute functions `N`, `S`, `E`, `W` are defined for North, South, East, and West, such that `Space` models the Cartesian space with four quadrants: `A` $=$ `N(D)` $=$ `E(B)`, `B` $=$ `N(C)` $=$ `W(A)`, and so on, as shown diagrammatically in Figure 2 *right*. Function `Hr` is the identity on `Space`.

In the definition of store-probability expressions $v_p \in (0,1]$, c_p is a store-probability constant in $(0,1]$ defined using the FlyFast **const** construct, and fn_p is a store-probability function defined by the user in F_Υ, which, when applied to attribute expressions $e_1, \ldots, e_m$ returns a probability value.The syntax for store-probability function definitions *pfd* is similar to that of attribute functions:

pfd ::= **func** $fn_p(x_1 : T1, \ldots, x_m : Tm) :$ **float**;*pfb* **endfunc**
pfb ::= p |**case** $(x_1, \ldots, x_m)$ **of**$(v_{a_{11}}, \ldots, v_{a_{m1}}) : p_1; (v_{a_{12}}, \ldots, v_{a_{m2}}) : p_2; \ldots (v_{a_{1k}}, \ldots, v_{a_{mk}}) : p_k$

where fn_p is the name of the store-probability function, the result type is **float** (actually the range $[0,1]$) $x_1 : T1, \ldots, x_m : Tm$ are its parameters and their relative types, p, p_i are store-probability expressions and $v_{a_{ij}}$ are attribute-values. In any store update definition it must be guaranteed that the values of $p_1 \ldots p_n$ sum up[4] to 1. The informal meaning is clear. The store update will make attributes $a_1, \ldots, a_k$ take the values of $e_{1i}, \ldots, e_{ki}$ respectively with probability equal to the value of p_i.

In Figure 1 store-probability functions `pHr`, `pN`, `pS`, `pE`, `pW` are defined that give the probabilities of not moving (`pHr`), or of jumping to North (`pN`), South (`pS`), East (`pE`), West (`pW`), as functions of the

[3] In [9] a slightly different syntax for store updates has been used.

[4] In this version of the translation we allow only flat updates, i.e. the specific probability of each combination of values assigned to the attributes must be given explicitly. Other possibilities could be defined using *combinations* of (independent) probability distributions.

$$
\begin{array}{lcl}
\mathtt{S} & := & \mathsf{frc}(\mathtt{I}) :: \mathtt{inf}^*[\bot]\langle\rangle\mathtt{Jump.I} \; + \\
 & & \mathsf{frc}(\mathtt{S}) :: \mathtt{nsc}^*[\bot]\langle\rangle\mathtt{Jump.S} \\
 & & \\
\mathtt{I} & := & \mathit{ii} :: \mathtt{inf}^*[\bot]\langle\rangle\mathtt{Jump.I} \; + \\
 & & \mathit{ir} :: \mathtt{rec}^*[\bot]\langle\rangle\mathtt{Jump.S}
\end{array}
$$

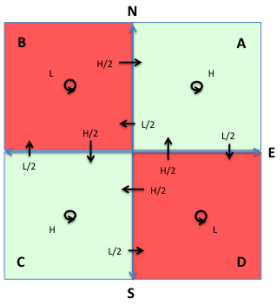

Figure 2: *SI*, a behavioural model.

current location.

Example 1 *A simplified version of the behaviour of the epidemic process discussed in [9] is shown in Figure 2* left[5]*. In Figure 1 we show a fragment of F_{SI} defining store update* `Jump` *together with the relevant type, constant and function definitions as introduced above. The component has just one attribute, named* loc*, with values in* `Space`*. The effect of* `Jump` *executed by a component in which* loc *is bound to quadrant ℓ is to leave the value of* loc *unchanged with probability* $\mathtt{pHr}(\ell)$*, change it to the quadrant North of ℓ with probability* $\mathtt{pN}(\ell)$*, and so on. Note that* $\mathtt{H} > \mathtt{L}$ *and this implies that higher probability is assigned to* `A` *and* `C` *and low probability to* `B` *and* `D`*. This is represented in Figure 2* right *where higher probability locations are shown in green and lower probability ones are shown in red; moreover, the relevant probabilities are represented as arrows (*$\mathtt{H}/2, \mathtt{L}/2$*) or self-loops (*$\mathtt{H}, \mathtt{L}$*). A susceptible (state* `S`*) component becomes infected (state* `I`*) via an* `inf` *action which takes place with probability equal to the fraction of components in the system which are currently infected (i.e.* $\mathsf{frc}(\mathtt{I})$*); it remains in state* `S` *via the self-loop labelled by action* `nsc`*, with probability* $\mathsf{frc}(\mathtt{S}) = 1 - \mathsf{frc}(\mathtt{I})$*. An infected node (state* `I`*) may recover, entering state* `S` *with action* `rec` *and probability* `ir`*; while infected, it keeps executing action* `inf`*, with probability* `ii`*. Note that, for the sake of simplicity, we use only* internal *actions, modelled by means of* output *actions with predicate false ($\bot$). We assume that in the initial global state all outboxes are non-empty; each contains the initial store of the specific component (i.e., its initial location), predicate $\bot$ and the empty tuple $\langle\rangle$).*

A PiFF state defining equation has the following (abstract) form: $C := \sum_{j\in J}[g_j]p_j :: act_j.C_j$ where either $[g_j]p_j$ is the keyword **rest** or:

- g_j is a boolean expression b which may depend on the current store, but not on the current occupancy measure vector: $b ::= \top \mid \bot \mid e \bowtie e \mid \neg b \mid b \wedge b$ and $e ::= v_a \mid c_a \mid \mathbf{my}.a$ where $\top$ ($\bot$) denotes the constant `true` (`false`), $\bowtie \in \{\geq, >, \leq, <\}$, v_a is an attribute value (drawn from finite set $\mathscr{V}$ of attribute values), c_a is an attribute constant in $\mathscr{V}$ defined using the FlyFast **const** construct, and a is the name of an attribute of the component.
- p_j is a transition probability expression $p ::= v_p \mid c_p \mid \mathsf{frc}(C) \mid \mathsf{frc}(\pi) \mid \prod_{i\in I} p_i \mid \sum_{i\in I} p_i \mid 1 - p$, for finite I, where $v_p \in (0,1]$, c_p a constant in $(0,1]$ defined via the **const** construct, and π is defined as b above, but where expressions e can also be attribute names a (i.e. $e ::= v_a \mid c_a \mid \mathbf{my}.a \mid a$); $\mathsf{frc}(C)$ is the fraction of components *currently* in state C over the total number N; similarly, $\mathsf{frc}(\pi)$ is the fraction of components the *current* store of which satisfies π, over the total number N. Note that it must be guaranteed that $\prod_{i\in I} p_i \leq 1$ and $\sum_{i\in I} p_i \leq 1$.

[5]We focus only on those features that are most relevant for the present paper. In [9] also other features are shown like, e.g. the use of (predicate-based) input actions, which are not the main subject of this paper.

- act_j can be an output action $\alpha^*[\pi]\langle\rangle\sigma$ or an input action $\alpha^*[\pi]()\sigma$, where π is as above and σ is the name of a store update. Note that in the case of an input action, π refers to the store of the partner component in the *previous* step of the computation.

If $[g_j]p_j = \mathbf{rest}$, then act_j must be an output action $\alpha^*[\pi]\langle\rangle\sigma$, to be executed with the residual probability.

2.2 FlyFast

FlyFast accepts a specification $\langle\Delta, A, \mathbf{C}_0\rangle^{(N)}$ of a model of a system consisting of the clock-synchronous product of N instances of a probabilistic agent. The states of the DTMC-like agent model are specified by a set of state-defining equations Δ. The (abstract) form of a state defining equation is the following $C := \sum_{i=1}^{r} a_i.C_i$ where $a_i \in \mathscr{A}$—the set of FlyFast *actions*—$C, C_i \in \mathscr{S}$—the set of FlyFast *states*—and, for $i, j = 1, \ldots, r$ $a_i \neq a_j$ if $i \neq j$; note that $C_i = C_j$ with $i \neq j$ is allowed instead[6]. Each action has a probability assigned by means of an action probability function definition in A of the form $a :: exp$ where exp is an expression consisting of constants and $\mathsf{frc}(C)$ terms. Constants are floating point values or names associated to such values using the construct **const** $<$ `name` $>$ $=$ $<$ `value` $>$; $\mathsf{frc}(C)$ denotes the element associated to state C in the current occupancy measure vector[7]. So, strictly speaking, Δ and A characterise an inhomogeneous DTMC whose probability matrix $\mathbf{K}(\mathbf{m})$ is a function of the occupancy measure vector $\mathbf{m}$ such that for each pair of states C, C', the matrix element $\mathbf{K}(\mathbf{m})_{C,C'}$ is the probability of jumping from C to C' given the current occupancy measure vector $\mathbf{m}$. Letting $\mathscr{S}_\Delta$ be the set of states of the agent, with $|\mathscr{S}_\Delta| = S$, and $\mathscr{U}^S = \{(m_1, \ldots, m_S) | m_1 + \ldots + m_S = 1\}$ denote the unit simplex of dimension S, we have $\mathbf{K} : \mathscr{U}^S \times \mathscr{S}_\Delta \times \mathscr{S}_\Delta \to [0,1]$. Matrix $\mathbf{K}$ is generated directly from the input specification $\langle\Delta, A, \mathbf{C}_0\rangle^{(N)}$; the reader interested in the details of how to derive $\mathbf{K}$ is referred to [27, 29]. Auxiliary function definitions can be specified in A. The initial state $\mathbf{C}_0$ is a vector of size N consisting of the initial state of each individual object. Finally, note that in matrix $\mathbf{K}(\mathbf{m})$ the information on specific actions is lost, which is common in PCTL/DTMC based approaches; furthermore, we note that, by construction, $\mathbf{K}(\mathbf{m})$ does not depend on N (see [27, 29] for details).

3 A revised translation

As in [9], we define a translation such that, given a PiFF system specification $\Upsilon = (\Delta_\Upsilon, F_\Upsilon, \Sigma_0)^{(N)}$, the translation returns the FlyFast system specification $\langle\Delta, A, \mathbf{C}_0\rangle^{(N)}$ preserving probabilistic semantics. The predicate-based FlyFast front-end is then completed with a simple translation at the PCTL level, for which we refer to [9].

The system model specification translation consists of two phases. In the first phase, each action in the input system model specification Υ is annotated with an identifier which is unique within the specification. We let $\aleph(\Upsilon)$ denote the resulting specification. These annotations will make action names unique specification-wide thus eliminating complications which may arise from multiple occurrences of the same action, in particular when leading to the same state (see [9] for details). Of course, these annotations are disregarded in the probabilistic semantics, when considering the interaction model of components. In other words, an output action $\alpha\langle\rangle$ in outbox $(\gamma, \pi, \alpha\langle\rangle)$ must match with any input

[6] The concrete FlyFast syntax is: `state C{a_1.C_1 + a_2.C_2 ...a_r.C_r}`.

[7] The occupancy measure vector is a vector with as many elements as the number of states of an individual agent; 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 occupancy measure vector is a compact representation of the system global state.

action $\alpha()$ even if $\alpha\langle\rangle$ would actually correspond to $(\alpha,\iota)^*[\pi]\langle\rangle$ and $\alpha()$ would actually correspond to $(\alpha,\eta)^*[\pi']()$. Apart from this detail, the probabilistic semantics as defined in [9] remain unchanged.

The second phase is defined by the translation algorithm defined in Figure 5, which is a revised and simplified version of that presented in [9] and is applied to $\aleph(\Upsilon)$. We let $\mathscr{I}(\aleph(\Upsilon))$ denote the result of the translation, namely the pure FlyFast system specification $\langle\Delta,A,\mathbf{C}_0\rangle^{(N)}$.

We recall here some notation from [9]. We let $\mathscr{S}_{\Delta_\Upsilon}$ denote the set of states of Υ; Γ_{Δ_Υ} is the set of all stores defined over the attributes of Υ—a store is a finite mapping from the attributes of the component to a finite set of values $\mathscr{V}$, thus Γ_{Δ_Υ} is finite—and $\mathscr{O}_{\Delta_\Upsilon}$ the finite set of all outboxes of Υ. A Υ *component-state* is a triple $(C,\gamma,O)\in\mathscr{S}_{\Delta_\Upsilon}\times\Gamma_{\Delta_\Upsilon}\times\mathscr{O}_{\Delta_\Upsilon}=\Omega_{\Delta_\Upsilon}$. If the component-state is the target of a transition modelling the execution of an *output* action, then $O=(\gamma',\pi,\alpha\langle\rangle)$, where γ' is the store of the (component-state) source of the transition, π is the predicate used in the action—actualised with γ'—and $\alpha\langle\rangle$ the actual message sent by the action. If, instead, the component-state is the target of a transition for an *input* action, then $O=\langle\rangle$, i.e. the empty outbox. Note that the set of component states of $\aleph(\Upsilon)$ is identical to that of Υ. Also the set of all stores of $\aleph(\Upsilon)$ is the same as that of Υ. In the algorithm of Figure 5 by $t*t'$ we mean the *syntactical* term representing the product of terms t and t'; the notation is extended to $\mathsf{PROD}\{t|\mathsf{cond}(t)\}$, denoting the *syntactical* product $t_1*\ldots*t_n$ if $\{t|\mathsf{cond}(t)=\mathrm{tt}\}=\{t_1,\ldots,t_n\}\neq\emptyset$ and 1 otherwise. Similarly, $\mathsf{SUM}\{t|\mathsf{cond}(t)\}$ denotes the *syntactical* sum $t_1+\ldots+t_n$ if $\{t|\mathsf{cond}(t)=\mathrm{tt}\}=\{t_1,\ldots,t_n\}\neq\emptyset$ and 0 otherwise. The translation algorithm uses a few auxiliary functions which we briefly discuss below:

- $\mathscr{I}_{\mathscr{S}}:\Omega_{\Delta_\Upsilon}\to\mathscr{S}$ is a total injection which maps every component state of $\aleph(\Upsilon)$ to a distinct state of $\mathscr{I}(\aleph(\Upsilon))$; we recall that $\mathscr{S}$ denotes the set of state names of FlyFast models.
- $\mathscr{I}_{\mathscr{A}}:(\mathscr{S}_{\Delta_\Upsilon}\times\Gamma_{\Delta_\Upsilon})\times(\Lambda_{\Delta_\Upsilon}\times I_\aleph)\times\Omega_{\Delta_\Upsilon}\to\mathscr{A}$ is a total injection where, as in [9], $\Lambda_{\Delta_\Upsilon}$ is the set of action labels of Υ and $I_\aleph$ is the set of unique identifiers used in the first phase of the translation. We recall that $\mathscr{A}$ is the set of action names of FlyFast. The mapping of actions is a bit more delicate because we have to respect FlyFast static constraints and, in particular, we have to avoid multiple probability function definitions for the same action. A first source of potential violations (i.e. multiple syntactical occurrences of the same action) has been removed by action annotation in the first phase of the translation. A second source is the fact that the same action can take place in different contexts (for example with different stores) or leading to different target component states (maybe with different probabilities). To that purpose, we could distinguish different occurrences of the same action in different transitions, each characterised by its source component-state and its target component-state in Ω_{Δ_Υ}. In practice, since an action of a component cannot be influenced by the current outbox of the component, it is sufficient to restrict the first component of the domain from Ω_{Δ_Υ} to $(\mathscr{S}_{\Delta_\Upsilon}\times\Gamma_{\Delta_\Upsilon})$.
- The interpretation functions defined in Figure 3, namely those depending on stores only (and not on occupancy measure vectors); we assume $\mathbf{E_L}[\![\cdot]\!]_\gamma$ extended to $\mathbf{E_L}[\![fn]\!]_\gamma$ for defined function fn, in the standard way. In Figure 3 β_Υ denotes the constant to value bindings generated by the **const** construct in the input model specification Υ, whereas store update *upd* is defined as above.
- The translation function $\mathscr{I}_{\mathscr{P}}$ for transition probability expressions p_j, defined in Figure 4.

Output actions are dealt with in step 1 of the algorithm of Figure 5. Let us consider, for example, $(\mathtt{inf},1)^*[\bot]\langle\rangle\,\mathtt{Jump}$ in the definition of state `S` in Figure 2 (assuming annotations are integer values and the action has been annotated with 1). We know that the possible values for locations are `A,B,C,D`, so that the set of all stores is $\{\mathsf{loc}\}\to\{\mathtt{A},\mathtt{B},\mathtt{C},\mathtt{D}\}$. The algorithm generates 12 actions[8]. Let us focus

[8]Diagonal jumps are not contemplated in the model; technically this comes from the actual probability values used in the

$$
\begin{array}{lcl}
\mathbf{E_L}[\![\top]\!]_\gamma & = & \mathrm{tt} \\
\mathbf{E_L}[\![\bot]\!]_\gamma & = & \mathrm{ff} \\
\mathbf{E_L}[\![e_1 \bowtie e_2]\!]_\gamma & = & \mathbf{E_L}[\![e_1]\!]_\gamma \bowtie \mathbf{E_L}[\![e_2]\!]_\gamma \\
\mathbf{E_L}[\![\neg b]\!]_\gamma & = & \neg \mathbf{E_L}[\![b]\!]_\gamma \\
\mathbf{E_L}[\![b_1 \ \wedge \ b_2]\!]_\gamma & = & \mathbf{E_L}[\![b_1]\!]_\gamma \ \wedge \ \mathbf{E_L}[\![b_2]\!]_\gamma \\
\mathbf{E_L}[\![v_a]\!]_\gamma & = & v_a \\
\mathbf{E_L}[\![c_a]\!]_\gamma & = & \beta_\Upsilon(c_a) \\
\mathbf{E_L}[\![v_p]\!]_\gamma & = & v_p \\
\mathbf{E_L}[\![c_p]\!]_\gamma & = & \beta_\Upsilon(c_p) \\
\mathbf{E_L}[\![a]\!]_\gamma & = & a \\
\mathbf{E_L}[\![\mathbf{my}.a]\!]_\gamma & = & \gamma(a) \\
\mathbf{E_L}[\![fn_a(e_1,\ldots,e_m)]\!]_\gamma & = & \mathbf{E_L}[\![fn_a]\!]_\gamma(\mathbf{E_L}[\![e_1]\!]_\gamma,\ldots,\mathbf{E_L}[\![e_m]\!]_\gamma) \\
\mathbf{E_L}[\![fn_p(e_1,\ldots,e_m)]\!]_\gamma & = & \mathbf{E_L}[\![fn_p]\!]_\gamma(\mathbf{E_L}[\![e_1]\!]_\gamma,\ldots,\mathbf{E_L}[\![e_m]\!]_\gamma) \\
\mathbf{E_U}[\![upd]\!]_\gamma & = & \lambda\gamma'.\mathtt{dom}(\gamma') \neq \{a_1,\ldots,a_k\} \to 0; \\
 & & \quad \gamma'(a_1) = \mathbf{E_L}[\![e_{11}]\!]_\gamma \ \wedge \ \ldots \ \wedge \ \gamma'(a_k) = \mathbf{E_L}[\![e_{k1}]\!]_\gamma \to \mathbf{E_L}[\![p_1]\!]_\gamma; \\
 & & \quad \vdots \\
 & & \quad \gamma'(a_1) = \mathbf{E_L}[\![e_{1n}]\!]_\gamma \ \wedge \ \ldots \ \wedge \ \gamma'(a_k) = \mathbf{E_L}[\![e_{kn}]\!]_\gamma \to \mathbf{E_L}[\![p_n]\!]_\gamma; \\
 & & \quad \mathbf{otherwise} \ \to \ 0 \\
\mathbf{E_R}[\![\top]\!]_\gamma & = & \mathrm{tt} \\
\mathbf{E_R}[\![\bot]\!]_\gamma & = & \mathrm{ff} \\
\mathbf{E_R}[\![e_1 \bowtie e_2]\!]_\gamma & = & \mathbf{E_R}[\![e_1]\!]_\gamma \bowtie \mathbf{E_R}[\![e_2]\!]_\gamma \\
\mathbf{E_R}[\![\neg b]\!]_\gamma & = & \neg \mathbf{E_R}[\![b]\!]_\gamma \\
\mathbf{E_R}[\![b_1 \ \wedge \ b_2]\!]_\gamma & = & \mathbf{E_R}[\![b_1]\!]_\gamma \ \wedge \ \mathbf{E_R}[\![b_2]\!]_\gamma \\
\mathbf{E_R}[\![v_a]\!]_\gamma & = & v_a \\
\mathbf{E_R}[\![c_a]\!]_\gamma & = & \beta_\Upsilon(c_a) \\
\mathbf{E_R}[\![a]\!]_\gamma & = & \gamma(a)
\end{array}
$$

Figure 3: Interpretation functions relevant for the translation

on the action ξ associated to local position A (i.e. $\gamma = [\mathsf{loc} \mapsto \mathtt{A}]$) and possible next position B (i.e. $\gamma' = [\mathsf{loc} \mapsto B]$); the algorithm will generate the FlyFast probability function definition $\xi :: \mathtt{pW}(A) * (\mathsf{frc}(\mathtt{I1}) + \ldots + \mathsf{frc}(\mathtt{In}))$[9] as well as a transition leading to (a state which is the encoding, via $\mathscr{I}_{\mathscr{S}}$, of) the component state with I as (proper) state, store γ', and outbox $(\gamma, \bot, \mathtt{inf}\langle\rangle)$. Since the action is not depending on the current outbox, in practice a copy of such a transition is generated *for each* component state sharing the same proper state S and the same store γ. The translation scheme for input actions is defined in case 2 and is similar, except that one has also to consider the sum of the fractions of the possible partners. The translation of the **rest** case is straightforward. Note that for every $\zeta :: r * q \in A_\gamma$, r is a probability value associated to a store update; since any store update characterizes a probability distribution over stores, assuming the range of such a distribution is $\{r_1,\ldots,r_n\}$ if $\zeta_i :: r_i * q \in A_\gamma$, then also $\zeta_j :: r_j * q \in A_\gamma$ for all $j = 1,\ldots,n,\ j \neq i$ with $\sum_{i=1}^{n} r_j = 1$. Thus the remaining probability is $q_\gamma = (1 - \mathsf{SUM}\{q | \zeta :: r * q \in A_\gamma\})$, where q is either a term $\mathscr{I}_{\mathscr{P}}(p_j)_\gamma$, with p_j occurring in a summand of the state defining equation (see step 1), or a term $\mathscr{I}_{\mathscr{P}}(p_j)_\gamma * \mathsf{SUM}\{\mathsf{frc}(\mathscr{I}_{\mathscr{S}}(\Sigma)) | \ldots\}$ (see step 2). It worth

definition of Jump.

[9] Here we assume that $\mathscr{I}_{\mathscr{S}}(\{(C,\gamma,O) \in \Omega_{\Delta_\Upsilon} | C = \mathtt{I}\}) = \{\mathtt{I1},\ldots,\mathtt{In}\} \subset \mathscr{S}$.

$$
\begin{array}{lcl}
\mathscr{I}_{\mathscr{P}}(v_p)_\gamma & = & v_p \\
\mathscr{I}_{\mathscr{P}}(c_p)_\gamma & = & \beta_\Upsilon(c_p) \\
\mathscr{I}_{\mathscr{P}}(\mathsf{frc}(C))_\gamma & = & \mathsf{SUM}\{\mathsf{frc}(\mathscr{I}_{\mathscr{S}}((C',\gamma',O'))) \mid (C',\gamma',O') \in \Omega_{\Delta_\Upsilon} \text{ and } C' = C\} \\
\mathscr{I}_{\mathscr{P}}(\mathsf{frc}(\pi))_\gamma & = & \mathsf{SUM}\{\mathsf{frc}(\mathscr{I}_{\mathscr{S}}((C',\gamma',O'))) \mid (C',\gamma',O') \in \Omega_{\Delta_\Upsilon} \text{ and } \mathbf{E_R}[\![\mathbf{E_L}[\![\pi]\!]_\gamma]\!]_{\gamma'} = \mathrm{tt}\} \\
\mathscr{I}_{\mathscr{P}}(\prod_{i\in I}\ p_i)_\gamma & = & \mathsf{PROD}\{\mathscr{I}_{\mathscr{P}}(p_i)_\gamma \mid i \in I\} \\
\mathscr{I}_{\mathscr{P}}(\sum_{i\in I}\ p_i)_\gamma & = & \mathsf{SUM}\{\mathscr{I}_{\mathscr{P}}(p_i)_\gamma \mid i \in I\}
\end{array}
$$

Figure 4: Transition probability expressions translation function definition

For each state equation $C := \sum_{j\in J}[g_j]p_j :: act_j.C_j$ in Δ_Υ:

1. For each *output* action $(\alpha,\iota)^*[\pi]\langle\rangle\sigma = act_k$ with $k \in J$ and $[g_k]p_k \neq \mathbf{rest}$,
   for each $\gamma \in \Gamma_{\Delta_\Upsilon}$ s.t. $\mathbf{E_L}[\![g_k]\!]_\gamma = \mathrm{tt}$ and $(C,\gamma,O) \in \Omega_{\Delta_\Upsilon}$ for some $O \in \mathscr{O}_{\Delta_\Upsilon}$, for each $\gamma' \in \Gamma_{\Delta_\Upsilon}$ s.t. $(C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)) \in \Omega_{\Delta_\Upsilon}$ and $\mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') > 0$, let $\xi = \mathscr{I}_{\mathscr{A}}((C,\gamma),(\alpha\langle\rangle,\iota),(C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)))$ be a fresh new action in the FlyFast model specification $\mathscr{I}(\aleph(\Upsilon)) = \langle\Delta,A,\mathbf{C}_0\rangle^{(N)}$ and add the following action probability function definition in A: $\xi :: \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') * \mathscr{I}_{\mathscr{P}}(p_k)_\gamma$.
   Moreover, for each outbox $O \in \mathscr{O}_{\Delta_\Upsilon}$ s.t. $(C,\gamma,O) \in \Omega_{\Delta_\Upsilon}$, the following summand is added to the equation in Δ for state $\mathscr{I}_{\mathscr{S}}((C,\gamma,O))$: $\xi.\mathscr{I}_{\mathscr{S}}((C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)))$;

2. For each *input* action $(\alpha,\iota)^*[\pi]()\sigma = act_k$, with $k \in J$ and $[g_k]p_k \neq \mathbf{rest}$,
   for each $\gamma \in \Gamma_{\Delta_\Upsilon}$ s.t. $\mathbf{E_L}[\![g_k]\!]_\gamma = \mathrm{tt}$ and $(C,\gamma,O) \in \Omega_{\Delta_\Upsilon}$ for some $O \in \mathscr{O}_{\Delta_\Upsilon}$, for each $\gamma' \in \Gamma_{\Delta_\Upsilon}$ s.t. $(C_k,\gamma',\langle\rangle) \in \Omega_{\Delta_\Upsilon}$ and $\mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') > 0$,
   let $\xi = \mathscr{I}_{\mathscr{A}}((C,\gamma),(\alpha(),\iota),(C_k,\gamma',\langle\rangle))$, be a fresh new action in the FlyFast model specification $\mathscr{I}(\aleph(\Upsilon)) = \langle\Delta,A,\mathbf{C}_0\rangle^{(N)}$ and add the following action probability function definition in A:
   $\xi :: \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') * \mathscr{I}_{\mathscr{P}}(p_k)_\gamma *$
   $*\mathsf{SUM}\{\mathsf{frc}(\mathscr{I}_{\mathscr{S}}(\Sigma)) \mid \Sigma = (C'',\gamma'',(\overline{\gamma},\overline{\pi},\alpha\langle\rangle)) \in \Omega_{\Delta_\Upsilon} \ \wedge$
   $\wedge\ \mathbf{E_R}[\![\overline{\pi}]\!]_\gamma = \mathbf{E_R}[\![\mathbf{E_L}[\![\pi]\!]_\gamma]\!]_{\overline{\gamma}} = \mathrm{tt}\}$.
   Moreover, for each outbox $O \in \mathscr{O}_{\Delta_\Upsilon}$ s.t. $(C,\gamma,O) \in \Omega_{\Delta_\Upsilon}$, the following summand is added to the equation in Δ for state $\mathscr{I}_{\mathscr{S}}((C,\gamma,O))$: $\xi.\mathscr{I}_{\mathscr{S}}((C_k,\gamma',\langle\rangle))$;

3. If there exists $k \in J$ s.t. $[g_k]p_k = \mathbf{rest}$, and $act_k = (\alpha,\iota)^*[\pi]\langle\rangle\sigma$, for each $\gamma \in \Gamma_{\Delta_\Upsilon}$ s.t. $(C,\gamma,O) \in \Omega_{\Delta_\Upsilon}$ for some $O \in \mathscr{O}_{\Delta_\Upsilon}$, let A_γ be the set of probability function definitions which has been constructed in steps (1) and (2) above. Let q_γ be defined by $q_\gamma = (1 - \mathsf{SUM}\{q \mid \zeta :: r * q \in A_\gamma\})$. For all $\gamma' \in \Gamma_{\Delta_\Upsilon}$ s.t. $(C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)) \in \Omega_{\Delta_\Upsilon}$, let $\xi = \mathscr{I}_{\mathscr{A}}((C,\gamma),(\alpha,\iota)\langle\rangle,(C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle))) \in \Omega_{\Delta_\Upsilon}$, be a fresh new action in the FlyFast model specification $\mathscr{I}(\aleph(\Upsilon)) = \langle\Delta,A,\mathbf{C}_0\rangle^{(N)}$ and add the following action probability function definition in A: $\xi :: \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') * q_\gamma$.
   Moreover, for each outbox $O \in \mathscr{O}_{\Delta_\Upsilon}$ s.t. $(C,\gamma,O) \in \Omega_{\Delta_\Upsilon}$, the following summand is added to the equation in Δ for state $\mathscr{I}_{\mathscr{S}}((C,\gamma,O))$: $\xi.\mathscr{I}_{\mathscr{S}}((C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)))$;

4. No other action probability function definition and transition is included and the initial state $\mathbf{C}_0$ of $\mathscr{I}(\Upsilon)$ is defined as $\mathbf{C}_0 = \mathscr{I}_{\mathscr{S}}(\Sigma_0)$.

Figure 5: The translation algorithm

pointing out here that the translation of Figure 5 is essentially the same as that presented in [9], when the latter is applied to the sublanguage of PiFF where one requires that each action occurs at most once. The annotations performed in the first phase of the translation ensure that this requirement is fulfilled; as we noted above, these annotations are purely syntactical and are disregarded at the semantics level. We also recall that in probabilistic, pure DTMC process language semantics, actions are in the end dropped and, for each pair of states, the cumulative probability of such actions is assigned to the single transition from one of the states to the other one. Consequently, correctness of the translation, proved in [9], is preserved by the simplified version presented in this paper.

We note that in the algorithm sets Ω_{Δ_Υ}, Γ_{Δ_Υ} and $\mathscr{O}_{\Delta_\Upsilon}$ are used. Of course, an alternative approach could be one which considers only the set $\overline{\Omega_{\Delta_\Upsilon}}$ of component states which are *reachable* from a given initial component state and, consequently, the sets $\overline{\Gamma_{\Delta_\Upsilon}}$ and $\overline{\mathscr{O}_{\Delta_\Upsilon}}$ of *used* stores and outboxes. In this way, the size of the resulting FlyFast model specification would be smaller (for example in terms of number of

states). On the other hand, this approach might require recompilation for each model-checking session starting from a different initial component state.

4 A simplified language for Bisimulation-based optimisation

In this section we consider a simplified language for transition probability expressions appearing in state defining equations that will allow us to perform bisimulation based optimisation of the result $\langle \Delta, A, \mathbf{C}_0 \rangle^{(N)}$. The restricted syntax for transition probability expressions p we use in this section is the following: $p ::= e_p \,|\, e_p \cdot \mathsf{frc}(C) \,|\, e_p \cdot \mathsf{frc}(\pi)$ and $e_p ::= v_p \,|\, c_p$ where v_p and c_p and π are defined as in Section 2.

By inspection of the FlyFast translation as defined in Section 3, and recalling that the set $\mathscr{S}_\Delta$ of the states of the resulting FlyFast model, ranged over by $z, z_i, \ldots$, has cardinality S, it is easy to see that the probability action definition in the result of a translation of a generic *output* action is either of the form $\xi :: k$, or it is of the form $\xi :: k * \mathsf{SUM}\{\mathsf{frc}(z_i) | i \in I\}$ where k is a FlyFast constant. Moreover, if p was of the form $e_p \cdot \mathsf{frc}(C)$, then index set $I \subseteq \{1, \ldots, S\}$ identifies those states in $\mathscr{S}_\Delta$ that represent (via $\mathscr{I}_\mathscr{S}$) component states with proper local state C; if instead, p was of the form $e_p \cdot \mathsf{frc}(\pi)$, then $I \subseteq \{1, \ldots, S\}$ identifies those states in $\mathscr{S}_\Delta$ that represent (via $\mathscr{I}_\mathscr{S}$) component states with a store satisfying π in the relevant store. At the FlyFast semantics level, recalling that $\mathsf{frc}(z_i)$ is exactly the i-th component m_i of the occupancy measure vector $\mathbf{m} = (m_1, \ldots, m_S)$ of the model, we can rewrite[10] the above as k or $k \cdot \sum_{i \in I} m_i$.

Similarly, the probability action definition in the result of a translation of a generic *input* action $(\alpha, \iota)^*[\pi']()$ (executed in local store γ') will necessarily be of the form $k \cdot \left(\sum_{j \in I'} m_j\right)$ or of the form $k \cdot \left(\sum_{j \in I} m_j\right) \cdot \left(\sum_{j \in I'} m_j\right)$, for index sets I as above and I' as follows:

$$I' = \{i \in \{1, \ldots, S\} | \exists C, \gamma, \overline{\gamma}, \overline{\pi}, s.t. \\ z_i = \mathscr{I}_\mathscr{S}((C, \gamma, (\overline{\gamma}, \overline{\pi}, \alpha\langle\rangle))) \ \wedge \ \mathbf{E_R}[\![\overline{\pi}]\!]_{\gamma'} = \mathbf{E_R}[\![\mathbf{E_L}[\![\pi']\!]_\gamma]\!]_{\overline{\gamma}} = \mathrm{tt}\}$$

An immediate consequence of using the above mentioned restricted syntax for the probability function definitions is that, letting $\mathbf{K} : \mathscr{U}^S \times \mathscr{S} \times \mathscr{S} \to [0,1]$ be the transition probability matrix for the FlyFast translation of a model specification, we have that $\mathbf{K}(m_1, \ldots, m_S)_{z,z'}$ is a polynomial function of degree at most 2 in variables $m_1, \ldots, m_S$.

5 Bisimilarity and State-space Reduction

The following definition generalises standard probabilistic bisimilarity for state labelled DTMCs to the case in which transition probabilities are *functions* instead of constant values.

Definition 1 *For finite set of states $\mathscr{S}$, with $|\mathscr{S}| = S$, let $\mathbf{K} : \mathscr{U}^S \times \mathscr{S} \times \mathscr{S} \to [0,1]$ and, for $z \in \mathscr{S}$ and $Q \subseteq \mathscr{S}$, write $\mathbf{K}(\mathbf{m})_{z,Q}$ for $\sum_{z' \in Q} \mathbf{K}(\mathbf{m})_{z,z'}$. Let furthermore $\mathscr{L} : \mathscr{S} \to 2^{AP}$ be a state-labelling function, for a given set AP of atomic propositions. An equivalence relation $R \subseteq \mathscr{S} \times \mathscr{S}$ is called a* bisimulation relation *if and only if $z_1 \, R \, z_2$ implies: (i) $\mathscr{L}(z_1) = \mathscr{L}(z_2)$ and (ii) $\mathbf{K}(\mathbf{m})_{z_1,Q} = \mathbf{K}(\mathbf{m})_{z_2,Q}$, for all $\mathbf{m} \in \mathscr{U}^S$ and $Q \in \mathscr{S}/R$. The* bisimulation equivalence *on $\mathscr{S}$ is the largest bisimulation relation $R \subseteq \mathscr{S} \times \mathscr{S}$.*

We point out that the notion of bisimilarity does *not* introduce any approximation, and consequently error, in a model and related analyses. Bisimilarity is only a way for abstracting from details that are irrelevant

[10] With a little notational abuse using k also as the actual value in $[0,1]$ of the FlyFast constant k.

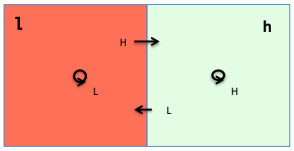

Figure 6: *SI* in two quadrants

for the specific analyses of interest. In particular, it is useful to remark that bisimilarity preserves also state labels, which are directly related to the atomic propositions of logic formulas for which model-checking is performed. Actually, it is well known that probabilistic bisimilarity coincides with PCTL equivalence, i.e. the equivalence induced on system states by their satisfaction of PCTL formulas, for finitely branching systems [3].

Note that $\mathbf{K}(m_1,\ldots,m_S)_{z_1,Q} = \mathbf{K}(m_1,\ldots,m_S)_{z_2,Q}$ for all $(m_1,\ldots,m_S) \in \mathscr{U}^S$ is in general not decidable. If instead we consider only transition probability matrices as in Section 4, we see that each side of the above equality is a polynomial function of degree at most 2 in variables $m_1,\ldots,m_S$ and one can define a normal form for the polynomial expressions in $m_1,\ldots,m_S$ supported by an ordering relation on the variable names (e.g. $m_1 \prec \ldots \prec m_S$) and get expressions of the general form $\left(\sum_{i=1}^{S}\sum_{j\geq i}^{S} h_{ij}\cdot m_i\cdot m_j\right) + \left(\sum_{i=1}^{S} h_i\cdot m_i\right) + h$ for suitable h_{ij},h_i,h. Actually, such expressions can always be rewritten in the form $\left(\sum_{i=1}^{S}\sum_{j\geq i}^{S} u_{ij}\cdot m_i\cdot m_j\right) + u$ for suitable u_{ij},u since, recalling that $\sum_{i=1}^{S} m_i = 1$, we get $\sum_{i=1}^{S} h_i\cdot m_i = \left(\sum_{i=1}^{S} m_i\right)\cdot\left(\sum_{i=1}^{S} h_i\cdot m_i\right)$ which, by simple algebraic manipulation, yields an expression of the following form: $\left(\sum_{i=1}^{S}\sum_{j\geq i}^{S} u'_{ij}\cdot m_i\cdot m_j\right)$; finally, we get $\left(\sum_{i=1}^{S}\sum_{j\geq i}^{S} u_{ij}\cdot m_i\cdot m_j\right) + u$ by letting $u_{ij} = h_{ij} + u'_{ij}$ and $u = h$. The following proposition thus establishes decidability of $\mathbf{K}(m_1,\ldots,m_S)_{z_1,Q} = \mathbf{K}(m_1,\ldots,m_S)_{z_2,Q}$ for all $(m_1,\ldots,m_S) \in \mathscr{U}^S$ for transition probability matrices as in Section 4:

Proposition 1
Let $A(m_1,\ldots,m_S) = \left(\sum_{i=1}^{S}\sum_{j\geq i}^{S} a_{ij}\cdot m_i\cdot m_j\right) + a$ *and* $B(m_1,\ldots,m_S) = \left(\sum_{i=1}^{S}\sum_{j\geq i}^{S} b_{ij}\cdot m_i\cdot m_j\right) + b$ *with* $a_{ij},b_{ij},a,b \in \mathbb{R}$, *where* $m_1,\ldots,m_S$ *are variables taking values over* $\mathbb{R}_{\geq 0}$ *with* $\sum_{i=1}^{S} m_i = 1$. *The following holds:* $(\forall m_1,\ldots,m_S.A(m_1,\ldots,m_S) = B(m_1,\ldots,m_S)) \Leftrightarrow ((\forall i,j = 1,\ldots,S \text{ with } i \leq j.a_{ij} = b_{ij}) \wedge a = b)$.

The above results can be used for reduction of the state-space of the individual agent, i.e. the resulting FlyFast model specification, after the application of the translation described in Section 3, by using for instance the standard probabilistic relational coarsest set partition problem algorithm (see e.g. [22], page 227) with slight obvious modifications due to the presence of state-labels and the need of symbolic computation capabilities required for checking (degree 2) polynomial expressions equality.[11] It is worth mentioning that state aggregation via bisimilarity is effective only if there is some sort of compatibility between (i) state labelling—and, consequently, the specific PCTL atomic propositions one uses—and (ii) the way probabilities are assigned to transitions—and, consequently, the cumulative probabilities to equivalence classes. We will come back on this issue in the following section.

	SA	*SB*	*SC*	*SD*	*IA*	*IB*	*IC*	*ID*
SA	$H\phi_S(\mathbf{m})$	$\frac{L}{2}\phi_S(\mathbf{m})$	0	$\frac{L}{2}\phi_S(\mathbf{m})$	$H\phi_I(\mathbf{m})$	$\frac{L}{2}\phi_I(\mathbf{m})$	0	$\frac{L}{2}\phi_I(\mathbf{m})$
SB	$\frac{H}{2}\phi_S(\mathbf{m})$	$L\phi_S(\mathbf{m})$	$\frac{H}{2}\phi_S(\mathbf{m})$	0	$\frac{H}{2}\phi_I(\mathbf{m})$	$L\phi_I(\mathbf{m})$	$\frac{H}{2}\phi_I(\mathbf{m})$	0
SC	0	$\frac{L}{2}\phi_S(\mathbf{m})$	$H\phi_S(\mathbf{m})$	$\frac{L}{2}\phi_S(\mathbf{m})$	0	$\frac{L}{2}\phi_I(\mathbf{m})$	$H\phi_I(\mathbf{m})$	$\frac{L}{2}\phi_I(\mathbf{m})$
SD	$\frac{H}{2}\phi_S(\mathbf{m})$	0	$\frac{H}{2}\phi_S(\mathbf{m})$	$L\phi_S(\mathbf{m})$	$\frac{H}{2}\phi_I(\mathbf{m})$	0	$\frac{H}{2}\phi_I(\mathbf{m})$	$L\phi_I(\mathbf{m})$
IA	Hir	$\frac{L}{2}ir$	0	$\frac{L}{2}ir$	Hii	$\frac{L}{2}ii$	0	$\frac{L}{2}ii$
IB	$\frac{H}{2}ir$	Lir	$\frac{H}{2}ir$	0	$\frac{H}{2}ii$	Lii	$\frac{H}{2}ii$	0
IC	0	$\frac{L}{2}ir$	Hir	$\frac{L}{2}ir$	0	$\frac{L}{2}ii$	Hii	$\frac{L}{2}ii$
ID	$\frac{H}{2}ir$	0	$\frac{H}{2}ir$	Lir	$\frac{H}{2}ii$	0	$\frac{H}{2}ii$	Lii

Figure 7: IDTMC transition probability matrix $\mathbf{K}(\mathbf{m})$, for $\mathbf{m}$ in $\mathscr{U}^8$.

6 Example

The application of the translation to the specification of Example 1 generates an agent model with 8 states, say $\mathscr{S}_\Delta = \{SA, SB, SC, SD, IA, IB, IC, ID\}$[12] with associated IDTMC probability transition matrix as shown in Figure 7 where m_{xy} represents the fraction of objects currently in state xy for $x \in \{S, I\}$ and $y \in \{A, B, C, D\}$—i.e. the components in state x and with $\mathsf{loc} = y$ in the original specification of Fig 2—so that $\mathbf{m} = (m_{SA}, m_{SB}, m_{SC}, m_{SD}, m_{IA}, m_{IB}, m_{IC}, m_{ID})$ is the occupancy measure vector. In Figure 7, functions ϕ_S and ϕ_I are used as abbreviations in the obvious way: $\phi_S(\mathbf{m}) = m_{SA} + m_{SB} + m_{SC} + m_{SD}$ and $\phi_I(\mathbf{m}) = m_{IA} + m_{IB} + m_{IC} + m_{ID}$. Let us assume now that we are interested in checking PCTL formulas on the model of Fig 2 which distinguish components located in A or C from those located in B or D, and those in state S from those in state I, that is we consider atomic propositions Sh, Ih, Sl and Il and a labelling $\mathscr{L}$ such that $\mathscr{L}(SA) = \mathscr{L}(SC) = \{Sh\}$, $\mathscr{L}(IA) = \mathscr{L}(IC) = \{Ih\}$, $\mathscr{L}(SB) = \mathscr{L}(SD) = \{Sl\}$, and $\mathscr{L}(IB) = \mathscr{L}(ID) = \{Il\}$.

Consider relation R on $\mathscr{S}_\Delta$ defined as $R = I_{\mathscr{S}_\Delta} \cup \{(SA, SC), (SB, SD), (IA, IC), (IB, ID)\} \cup \{(SC, SA), (SD, SB), (IC, IA), (ID, IB)\}$ where $I_{\mathscr{S}_\Delta}$ is the identity relation on $\mathscr{S}_\Delta$. It is very easy to show that R is a bisimulation according to Definition 1. Clearly R is an equivalence relation and its quotient $\mathscr{S}_\Delta/R$ is the set $\{Q_{Sh}, Q_{Sl}, Q_{Ih}, Q_{Il}\}$ with $Q_{Sh} = \{SA, SC\}, Q_{Sl} = \{SB, SD\}, Q_{Ih} = \{IA, IC\}, Q_{Il} = \{IB, ID\}$. In addition, for all $z_1, z_2 \in \mathscr{S}_\Delta$, whenever $z_1 R z_2$, we have $\mathscr{L}(z_1) = \mathscr{L}(z_2)$ and $\mathbf{K}(\mathbf{m})_{z_1,Q} = \mathbf{K}(\mathbf{m})_{z_2,Q}$ for all $Q \in \mathscr{S}_\Delta/R$ and for all $\mathbf{m}$, as one can easily check; clearly, R is also the largest bisimulation relation on $\mathscr{S}_\Delta$. The relationship between the two occupancy measure vectors is: $m_{Q_{Sh}} = m_{SA} + m_{SC}$, $m_{Q_{Sl}} = m_{SB} + m_{SD}$, $m_{Q_{Ih}} = m_{IA} + m_{IC}$, and $m_{Q_{Il}} = m_{IB} + m_{ID}$. We can thus use the reduced IDTMC defined by matrix $\widehat{\mathbf{K}}(m_{Q_{Sh}}, m_{Q_{Sl}}, m_{Q_{Ih}}, m_{Q_{Il}})$ shown in Figure 9. It corresponds to the FlyFast agent specification $\widehat{\Delta}$ given in Figure 8. In a sense, the high probability locations A and C, in the new model, have collapsed into a single one, namely h and the low probability ones (B and D) have collapsed into l, as shown in Figure 6. We point out again the correspondence between the symmetry in the space jump probability on one side and the definition of the state labelling function on the other side. Finally, note that a coarser labelling like, e.g. $\mathscr{L}'(SA) = \mathscr{L}'(SC) = \mathscr{L}'(IA) = \mathscr{L}'(IC) = \{h\}$, $\mathscr{L}'(SB) = \mathscr{L}'(SD) = \mathscr{L}'(IB) = \mathscr{L}'(ID) = \{l\}$ would make the model collapse into one with only two states, Q_h and Q_l, with probabilities $H : Q_h \to Q_h, H : Q_l \to Q_h$, $L : Q_h \to Q_l$ and $L : Q_h \to Q_l$ where only the location would be modelled whereas

[11] For instance, on page 227 of [22], line 12, $v(x,S) = v(y,S)$ should be replaced with $L(x) = L(y) \wedge v(x,S) = v(y,S)$ and in line 13, $v(x,S) \neq v(y,S)$ should be replaced with $L(x) \neq L(y) \vee v(x,S) \neq v(y,S)$, in order to take state labels into consideration as well. Of course $v(x,S)$ (L, respectively)is to be intended as $\mathbf{K}(\mathbf{m})_{z,Q}$ ($\mathscr{L}$, respectively), using the notation we introduced above for Bisimilarity.

[12] Actually the agent resulting from the translation of Figure 5 has a higher number of states due to the different possibilities for outbox values. Many of these states are unreachable from the initial state since the agent has no input action and we assume an initial unreachable state pruning has been performed.

```
action QSh_inf_QIh: H*(frc(QIh)+frc(QIl));        action QSh_nsc_QSh: H*(frc(QSh)+frc(QSl));
action QSh_inf_QIl: L*(frc(QIh)+frc(QIl));        action QSh_nsc_QSl: L*(frc(QSh)+frc(QSl));
action QSl_inf_QIh: H*(frc(QIh)+frc(QIl));        action QSl_nsc_QSh: H*(frc(QSh)+frc(QSl));
action QSl_inf_QIl: L*(frc(QIh)+frc(QIl));        action QSl_nsc_QSl: L*(frc(QSh)+frc(QSl));

action QIh_inf_QIh: H*ii;                         action QIh_rec_QSh: H*ir;
action QIh_inf_QIl: L*ii;                         action QIh_rec_QSl: L*ir;
action QIl_inf_QIh: H*ii;                         action QIl_rec_QSh: H*ir;
action QIl_inf_QIl: L*ii;                         action QIl_rec_QSl: L*ir;

state QSh{QSh_inf_QIh.QIh + QSh_inf_QIl.QIl + QSh_nsc_QSh.QSh + QSh_nsc_QSl.QSl}
state QSl{QSl_inf_QIh.QIh + QSl_inf_QIl.QIl + QSl_nsc_QSh.QSh + QSl_nsc_QSl.QSl}
state QIh{QIh_inf_QIh.QIh + QIh_inf_QIl.QIl + QIh_rec_QSh.QSh + QIh_rec_QSl.QSl}
state QIl{QIl_inf_QIh.QIh + QIl_inf_QIl.QIl + QIl_rec_QSh.QSh +QIl_rec_QSl.QSl}
```

Figure 8: Reduced agent specification $\widehat{\Delta}$

	Q_{Sh}	Q_{Sl}	Q_{Ih}	Q_{Il}
Q_{Sh}	$H \cdot (m_{Q_{Sh}} + m_{Q_{Sl}})$	$L \cdot (m_{Q_{Sh}} + m_{Q_{Sl}})$	$H \cdot (m_{Q_{Ih}} + m_{Q_{Il}})$	$L \cdot (m_{Q_{Ih}} + m_{Q_{Il}})$
Q_{Sl}	$H \cdot (m_{Q_{Sh}} + m_{Q_{Sl}})$	$L \cdot (m_{Q_{Sh}} + m_{Q_{Sl}})$	$H \cdot (m_{Q_{Ih}} + m_{Q_{Il}})$	$L \cdot (m_{Q_{Ih}} + m_{Q_{Il}})$
Q_{Ih}	$H \cdot ir$	$L \cdot ir$	$H \cdot ii$	$L \cdot ii$
Q_{Il}	$H \cdot ir$	$L \cdot ir$	$H \cdot ii$	$L \cdot ii$

Figure 9: IDTMC transition probability matrix function $\widehat{\mathbf{K}}(\mathbf{m})$, for $\mathbf{m}$ in $\mathscr{U}^4$.

information on the infection status would be lost.

7 Conclusions

PiFF [9] is a language for a predicate-based front-end of FlyFast, an on-the-fly mean-field model-checking tool. In this paper we presented a simplified version of the translation proposed in [9] together with an approach for model reduction that can be applied to the result of the translation. The approach is based on probabilistic bisimilarity for inhomogeneous DTMCs. An example of application of the procedure has been shown. The implementation of a compiler for PiFF mapping the language to FlyFast is under development as an add-on of FlyFast. We plan to apply the resulting extended tool to more as well as more complex models, in order to get concrete insights on the practical applicability of the framework and on the actual limitations imposed by the restrictions necessary for exploiting bisimilarity-based state-space reduction. Investigating possible ways of relaxing some of such restrictions will also be an interesting line of research.

Acknowledgments Research partially funded by EU Project n. 600708 *A Quantitative Approach to Management and Design of Collective and Adaptive Behaviours* (QUANTICOL).

References

[1] Adnan Aziz, Kumud Sanwal, Vigyan Singhal & Robert K. Brayton (2000): *Model-checking continous-time Markov chains*. *ACM Trans. Comput. Log.* 1(1), pp. 162–170, doi:10.1145/343369.343402.

[2] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns & Joost-Pieter Katoen (2003): *Model-Checking Algorithms for Continuous-Time Markov Chains*. *IEEE Trans. Software Eng.* 29(6), pp. 524–541, doi:10.1109/TSE.2003.1205180.

[3] Christel Baier & Joost-Pieter Katoen (2008): *Principles of model checking*. MIT Press.

[4] Girish Bhat, Rance Cleaveland & Orna Grumberg (1995): *Efficient On-the-Fly Model Checking for CTL**. In: *Proceedings, 10th Annual IEEE Symposium on Logic in Computer Science, San Diego, California, USA, June 26-29, 1995*, IEEE Computer Society, pp. 388–397, doi:10.1109/LICS.1995.523273.

[5] L. Bortolussi, G. Cabri, G. Di Marzo Serugendo, V. Galpin, J. Hillston, R. Lanciani, M. Massink & D. Tribastone, M. Weyns (2015): *Verification of CAS*. In J. Hillston, J. Pitt, M. Wirsing & F. Zambonelli, editors: *Collective Adaptive Systems: Qualitative and Quantitative Modelling and Analysis*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany, pp. 91–102. Dagstuhl Reports. Vol. 4, Issue 12. Report from Dagstuhl Seminar 14512. ISSN 2192-5283.

[6] Luca Bortolussi & Jane Hillston (2012): *Fluid Model Checking*. In Maciej Koutny & Irek Ulidowski, editors: *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, *Lecture Notes in Computer Science* 7454, Springer, pp. 333–347, doi:10.1007/978-3-642-32940-1_24.

[7] Luca Bortolussi & Jane Hillston (2015): *Model checking single agent behaviours by fluid approximation*. *Inf. Comput.* 242, pp. 183–226, doi:10.1016/j.ic.2015.03.002.

[8] Jean-Yves Le Boudec, David D. McDonald & Jochen Mundinger (2007): *A Generic Mean Field Convergence Result for Systems of Interacting Objects*. In: *Fourth International Conference on the Quantitative Evaluaiton of Systems (QEST 2007), 17-19 September 2007, Edinburgh, Scotland, UK*, IEEE Computer Society, pp. 3–18, doi:10.1109/QEST.2007.8.

[9] Vincenzo Ciancia, Diego Latella & Mieke Massink (2016): *On-the-Fly Mean-Field Model-Checking for Attribute-Based Coordination*. In Alberto Lluch-Lafuente & José Proença, editors: *Coordination Models and Languages - 18th IFIP WG 6.1 International Conference, COORDINATION 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*, *Lecture Notes in Computer Science* 9686, Springer, pp. 67–83, doi:10.1007/978-3-319-39519-7_5.

[10] Costas Courcoubetis, Moshe Y. Vardi, Pierre Wolper & Mihalis Yannakakis (1992): *Memory-Efficient Algorithms for the Verification of Temporal Properties*. *Formal Methods in System Design* 1(2/3), pp. 275–288, doi:10.1007/BF00121128.

[11] R.W.R. Darling & J.R. Norris (2008): *Differential equation approximations for Markov chains*. *Probability Surveys* 5, pp. 37–79, doi:10.1214/07-PS121.

[12] Rocco De Nicola, Diego Latella, Alberto Lluch-Lafuente, Michele Loreti, Andrea Margheri, Mieke Massink, Andrea Morichetta, Rosario Pugliese, Francesco Tiezzi & Andrea Vandin (2015): *The SCEL Language: Design, Implementation, Verification*. In Martin Wirsing, Matthias M. Hölzl, Nora Koch & Philip Mayer, editors: *Software Engineering for Collective Autonomic Systems - The ASCENS Approach*, *Lecture Notes in Computer Science* 8998, Springer, pp. 3–71, doi:10.1007/978-3-319-16310-9_1.

[13] Giuseppe Della Penna, Benedetto Intrigila, Igor Melatti, Enrico Tronci & Marisa Venturini Zilli (2004): *Bounded Probabilistic Model Checking with the Mur*alpha *Verifier*. In Alan J. Hu & Andrew K. Martin, editors: *Formal Methods in Computer-Aided Design, 5th International Conference, FMCAD 2004, Austin, Texas, USA, November 15-17, 2004, Proceedings*, *Lecture Notes in Computer Science* 3312, Springer, pp. 214–229, doi:10.1007/978-3-540-30494-4_16.

[14] Cheng Feng & Jane Hillston (2014): *PALOMA: A Process Algebra for Located Markovian Agents*. In Gethin Norman & William H. Sanders, editors: *Quantitative Evaluation of Systems - 11th International Conference, QEST 2014, Florence, Italy, September 8-10, 2014. Proceedings*, *Lecture Notes in Computer Science* 8657, Springer, pp. 265–280, doi:10.1007/978-3-319-10696-0_22.

[15] Nicolas Gast & Bruno Gaujal (2010): *A mean field model of work stealing in large-scale systems*. In Vishal Misra, Paul Barford & Mark S. Squillante, editors: *SIGMETRICS 2010, Proceedings of the 2010 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, New York, New York, USA, 14-18 June 2010*, ACM, pp. 13–24, doi:10.1145/1811039.1811042.

[16] Stefania Gnesi & Franco Mazzanti (2011): *An Abstract, on the Fly Framework for the Verification of Service-Oriented Systems*. In Martin Wirsing & Matthias M. Hölzl, editors: *Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing*, *Lecture Notes in Computer Science* 6582, Springer, pp. 390–407, doi:10.1007/978-3-642-20401-2_18.

[17] Guillaume Guirado, Thomas Hérault, Richard Lassaigne & Sylvain Peyronnet (2006): *Distribution, Approximation and Probabilistic Model Checking*. *Electr. Notes Theor. Comput. Sci.* 135(2), pp. 19–30, doi:10.1016/j.entcs.2005.10.016.

[18] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter & Lijun Zhang (2009): *INFAMY: An Infinite-State Markov Model Checker*. In Ahmed Bouajjani & Oded Maler, editors: *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, *Lecture Notes in Computer Science* 5643, Springer, pp. 641–647, doi:10.1007/978-3-642-02658-4_49.

[19] Hans Hansson & Bengt Jonsson (1994): *A Logic for Reasoning about Time and Reliability*. *Formal Asp. Comput.* 6(5), pp. 512–535, doi:10.1007/BF01211866.

[20] Thomas Hérault, Richard Lassaigne, Frédéric Magniette & Sylvain Peyronnet (2004): *Approximate Probabilistic Model Checking*. In Bernhard Steffen & Giorgio Levi, editors: *Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004, Venice, January 11-13, 2004, Proceedings*, *Lecture Notes in Computer Science* 2937, Springer, pp. 73–84, doi:10.1007/978-3-540-24622-0_8.

[21] Gerard J. Holzmann (2004): *The SPIN Model Checker - primer and reference manual*. Addison-Wesley.

[22] D. Huynh & L. Tian (1992): *On some equivalence relations for probabilistic processes*. *Fundamenta Informaticae* 17, pp. 211–234.

[23] A. Kolesnichenko, A. Remke & P.-T. de Boer (2012): *A logic for model-checking of mean-field models*. Technical Report TR-CTIT-12-11, http://doc.utwente.nl/80267/.

[24] Anna Kolesnichenko, Pieter-Tjerk de Boer, Anne Remke & Boudewijn R. Haverkort (2013): *A logic for model-checking mean-field models*. In: *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Budapest, Hungary, June 24-27, 2013*, IEEE Computer Society, pp. 1–12, doi:10.1109/DSN.2013.6575345.

[25] Kim G. Larsen & Axel Legay (2016): *Statistical Model Checking: Past, Present, and Future*. In Tiziana Margaria & Bernhard Steffen, editors: *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I*, *Lecture Notes in Computer Science* 9952, pp. 3–15, doi:10.1007/978-3-319-47166-2_1.

[26] D. Latella (1983): *Comunicazione basata su proprietà nei sistemi decentralizzati*. [Property-based interprocess communication in decentralized systems] Graduation Thesis. Istituto di Scienze dell'Informazione. Univ. of Pisa, Italy (in italian).

[27] Diego Latella, Michele Loreti & Mieke Massink (2013): *On-the-fly Fast Mean-Field Model-Checking*. In Martín Abadi & Alberto Lluch-Lafuente, editors: *Trustworthy Global Computing - 8th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers*, *Lecture Notes in Computer Science* 8358, Springer, pp. 297–314, doi:10.1007/978-3-319-05119-2_17.

[28] Diego Latella, Michele Loreti & Mieke Massink (2014): *On-the-fly Probabilistic Model Checking*. In Ivan Lanese, Alberto Lluch-Lafuente, Ana Sokolova & Hugo Torres Vieira, editors: *Proceedings 7th Interaction and Concurrency Experience, ICE 2014, Berlin, Germany, 6th June 2014.*, *EPTCS* 166, pp. 45–59, doi:10.4204/EPTCS.166.6.

[29] Diego Latella, Michele Loreti & Mieke Massink (2015): *On-the-fly PCTL fast mean-field approximated model-checking for self-organising coordination*. *Sci. Comput. Program.* 110, pp. 23–50, doi:10.1016/j.scico.2015.06.009.

[30] Michele Loreti & Jane Hillston (2016): *Modelling and Analysis of Collective Adaptive Systems with CARMA and its Tools*. In Marco Bernardo, Rocco De Nicola & Jane Hillston, editors: *Formal Methods for the*

Quantitative Evaluation of Collective Adaptive Systems - 16th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2016, Bertinoro, Italy, June 20-24, 2016, Advanced Lectures, *Lecture Notes in Computer Science* 9700, Springer, pp. 83–119, doi:10.1007/978-3-319-34096-8_4.

A Appendix

Proof of Proposition 1
$\Leftarrow$: Trivial.
$\Rightarrow$:
We first prove that $a = b$:

$\forall m_1,\ldots,m_S.A(m_1,\ldots,m_S) = B(m_1,\ldots,m_S)$

$\Rightarrow$ $\{$Logic$\}$

$A(0,\ldots,0) = B(0,\ldots,0)$

$\Rightarrow$ $\{$Def. of $A(m_1,\ldots,m_S)$ and $B(m_1,\ldots,m_S)\}$

$a = b$

Now we prove that $a_{ii} = b_{ii}$ for $i = 1,\ldots,S$:

$\forall m_1,\ldots,m_S.A(m_1,\ldots,m_S) = B(m_1,\ldots,m_S)$

$\Rightarrow$ $\{$Take the $S-tuple$ $(\bar{m}_1,\ldots,\bar{m}_S)$ where $\bar{m}_k = 1$ if $k = i$ and 0 otherwise$\}$

$A(\bar{m}_1,\ldots,\bar{m}_S) = B(\bar{m}_1,\ldots,\bar{m}_S)$

$\Rightarrow$ $\{$Def. of $A(m_1,\ldots,m_S)$ and $B(m_1,\ldots,m_S)\}$

$a_{ii} + a = b_{ii} + b$

$\Rightarrow$ $\{a = b$ (see above)$\}$

$a_{ii} = b_{ii}$

Finally we prove that $a_{ij} = b_{ij}$, for $i, j = 1,\ldots,S, j > i$:

$\forall m_1,\ldots,m_S.A(m_1,\ldots,m_S) = B(m_1,\ldots,m_S)$

$\Rightarrow$ $\{(\tilde{m}_1,\ldots,\tilde{m}_S)$ where $\tilde{m}_k = 0.5$ if $k \in \{i, j\}$ and 0 otherwise$\}$

$A(\tilde{m}_1,\ldots,\tilde{m}_S) = B(\tilde{m}_1,\ldots,\tilde{m}_S)$

$\Rightarrow$ $\{$Def. of $A(m_1,\ldots,m_S)$ and $B(m_1,\ldots,m_S)\}$

$0.25a_{ii} + 0.25a_{ij} + 0.25a_{jj} + a = 0.25b_{ii} + 0.25b_{ij} + 0.25b_{jj} + b$

$\Rightarrow$ $\{a = b$ (see above)$\}$

$0.25a_{ii} + 0.25a_{ij} + 0.25a_{jj} = 0.25b_{ii} + 0.25b_{ij} + 0.25b_{jj}$

$\Rightarrow$ $\{$Algebra$\}$

$a_{ii} + a_{ij} + a_{jj} = b_{ii} + b_{ij} + b_{jj}$

$\Rightarrow$ $\{a_{ii} = b_{ii}$ and $a_{jj} = b_{jj}$ (see above)$\}$

$a_{ij} = b_{ij}$ •