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 \(K(\mu(t))\) where \(K(m)\) 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. \(\mu\) 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/
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 \( \alpha \) to all components in the same location as the sender, making it change location randomly: \( \alpha^*[\text{loc} = \text{my.loc}] \langle \rangle \text{Jump} \). 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 \( \pi_r \), 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 \( \sigma \). 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 \( \pi_r \) which is satisfied by the component, and a store which satisfies in turn predicate \( \pi_s \). 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 \( \sigma \), 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) [12] 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
Designing Attribute-based FlyFast

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] 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 \( \Upsilon = (\Delta_T, F_T, \Sigma_0)^{(N)} \) is a triple where \( F_T \) is the set of relevant function definitions (e.g. store updates, auxiliary constants and functions), \( \Delta_T \) is a set of state defining equations, and \( \Sigma_0 \) is the initial system state (an \( N \)-tuple of component states, each of which being a 3-tuple \( (C, \gamma, O) \) of agent state \( C \), store \( \gamma \) 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 \texttt{atttype < name > enum < id-list >}. \( < \text{id-list} > \) is a finite list of identifiers. Of course, attributes can also take floating point values.

In Figure 1 the attribute type \texttt{Space} is defined that consists of four values \( A, B, C, D \) modelling four...
Designing Attribute-based FlyFast

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; endfunc;
func N(x: Space): Space; case x of A:B:C:D endfunc;
func S(x: Space): Space; case x of A:D:B:C:C:D endfunc;
func E(x: Space): Space; case x of A:B:A:C:D:D endfunc;
func W(x: Space): Space; case x of A:B:B:C:C:D endfunc;

func pHr(x: Space): float; case x of A:H:B:0: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 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:

```plaintext
const < name > = < value >.
```

A PiFF store update definition has the following syntax.\(^3\)

**update upd**

```plaintext
my.a1 := e11,...,my.a_k := e_k1 with p1;
:my.a1 := e1n,...,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:

```plaintext
e := v_a | c_a | fn_a(e_1,\ldots,e_m) and
p := v_p | c_p | fn_p(e_1,\ldots,e_m).
```

In the above definition of attribute expressions \(v_a\) is an attribute value (drawn from finite set \(Y\) of attribute values), \(c_a\) is an attribute constant in \(Y\) 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 \(F_r\), 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:

```plaintext
afd := func fn_a(x_1 : T_1,\ldots,x_m : T_m) : T ; afd endfunc
```

```plaintext
afb := e case (x_1,\ldots,x_m) of(v_{a_{i1}},\ldots,v_{a_{in}}) : e_1 ; (v_{a_{i2}},\ldots,v_{a_{in2}}) : e_2 ; \ldots ; (v_{a_{ik}},\ldots,v_{a_{ink}}) : e_k
```

where \(fn_a\) is the name of the attribute function, \(x_1 : T_1,\ldots,x_m : T_m\) are its parameters and their relative types, \(T\) is the type of the result of \(fn_a\); \(e, e_i\) are attribute-expressions and \(v_{aij}\) are attribute-values.

In Figure 2 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 rights. 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 \(F_r\), which, when applied to attribute expressions \(e_1,\ldots,e_m\), returns a probability value. The syntax for store-probability function definitions \(pdf\) is similar to that of attribute functions:

```plaintext
pdf := func fn_p(x_1 : T_1,\ldots,x_m : T_m) : float ; pdf endfunc
```

```plaintext
pdf := p case (x_1,\ldots,x_m) of(v_{a_{i1}},\ldots,v_{a_{in1}}) : p_1 ; (v_{a_{i2}},\ldots,v_{a_{in2}}) : p_2 ; \ldots ; (v_{a_{ik}},\ldots,v_{a_{ink}}) : p_k
```

where \(fn_p\) is the name of the store-probability function, the result type is \(float\) (actually the range \([0, 1])\)

\(x_1 : T_1,\ldots,x_m : T_m\) are its parameters and their relative types, \(p, p_i\) are store-probability expressions and \(v_{aij}\) are attribute-values. In any store update definition it must be guaranteed that the values of \(p_1\ldots p_n\) sum up to 1. The informal meaning is clear. The store update will make attributes \(a_1,\ldots,a_k\) take the values of \(e_1,\ldots,e_k\), respectively with probability equal to the value of \(p_j\).

In Figure 3 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

\(^{\text{3}}\)In [\(\text{1}\)] a slightly different syntax for store updates has been used.

\(^{\text{4}}\)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.
current location.

**Example 1** A simplified version of the behaviour of the epidemic process discussed in [9] is shown in Figure 2. In Figure 7 we show a fragment of FS1 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 $\ell$ is to leave the value of loc unchanged with probability $p_{\text{HR}}(\ell)$, change it to the quadrant North of $\ell$ with probability $p_{\text{NI}}(\ell)$, and so on. Note that $H > L$ 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 ($H/2, L/2$) or self-loops ($H, L$). 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. $frc(I)$); it remains in state $S$ via the self-loop labelled by action nsc, with probability $frc(S) = 1 - frc(I)$. 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: $C := \sum_{j \in J} [g_j]p_j :: \text{act}_j.C_j$ 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: $b := T \mid \bot \mid e \lor e \mid \neg b \mid b \land b$ and $e := v_a \mid c_a$ where $T$ (\(\bot\)) denotes the constant true (false), $\llbracket \in \{\geq, >, \leq, <\}$, $v_a$ is an attribute value (drawn from finite set $Y$ of attribute values), $c_a$ is an attribute constant in $Y$ defined using the FlyFast const construct, and $a$ is the name of an attribute of the component.

- $p_j$ is a transition probability expression $p := v_p \mid c_p \mid frc(C) \mid frc(\pi) \mid \prod_{i \in I} p_i \mid \text{sum}_{i \in I} p_i \mid 1 - p$, for finite $I$, where $v_p \in (0, 1)$, $c_p$ a constant in $(0, 1)$ defined via the const construct, and $\pi$ is defined as $b$ above, but where expressions $e$ can also be attribute names $a$ (i.e. $e := v_a \mid c_a \mid my.a \mid a$); $frc(C)$ is the fraction of components currently in state $C$ over the total number $N$; similarly, $frc(\pi)$ is the fraction of components the current store of which satisfies $\pi$, over the total number $N$. Note that it must be guaranteed that $\prod_{i \in I} p_i \leq 1$ and $\text{sum}_{i \in I} p_i \leq 1$.

\[\text{Figure 2: } SI, \text{ a behavioural model.}\]
act\_j can be an output action \( \alpha^+ | \pi \rangle \sigma \) or an input action \( \alpha^+ | \pi \rangle () \sigma \), where \( \pi \) is as above and \( \sigma \) is the name of a store update. Note that in the case of an input action, \( \pi \) refers to the store of the partner component in the previous step of the computation.

If \([g_j]p_j = \text{rest}\), then \( act\_j \) must be an output action \( \alpha^+ | \pi \rangle () \sigma \), to be executed with the residual probability.

### 2.2 FlyFast

FlyFast accepts a specification \( \langle \Delta, A, C_0 \rangle^{(N)} \) 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 \( \Delta \). The (abstract) form of a state defining equation is the following \( C := \sum_{i=1}^{r} a_i C_i \) where \( a_i \in \mathcal{A} \)—the set of FlyFast actions—\( C, C_i \in \mathcal{J} \)—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 \( a :: \text{exp} \) where \( \text{exp} \) is an expression consisting of constants and \( \text{frc}(C) \) terms. Constants are floating point values or names associated to such values using the construct \( \text{const} < \text{name} > = < \text{value} > ; \text{frc}(C) \) denotes the element associated to state \( C \) in the current occupancy measure vector.\(^7\) So, strictly speaking, \( \Delta \) and \( A \) characterise an inhomogeneous DTMC whose probability matrix \( K(m) \) is a function of the occupancy measure vector \( m \) such that for each pair of states \( C, C' \), the matrix element \( K(m)_{C,C'} \) is the probability of jumping from \( C \) to \( C' \) given the current occupancy measure vector \( m \). Letting \( \mathcal{J}_A \) be the set of states of the agent, with \( |\mathcal{J}_A| = S \), and \( \mathbb{R}^S = \{ (m_1, \ldots, m_S) | m_1 + \ldots + m_S = 1 \} \) denote the unit simplex of dimension \( S \), we have \( K : \mathbb{R}^S \times \mathcal{J}_A \times \mathcal{J}_A \rightarrow [0,1] \). Matrix \( K \) is generated directly from the input specification \( \langle \Delta, A, C_0 \rangle^{(N)} \); the reader interested in the details of how to derive \( K \) is referred to \( [27, 29] \). Auxiliary function definitions can be specified in \( A \). The initial state \( C_0 \) is a vector of size \( N \) consisting of the initial state of each individual object. Finally, note that in matrix \( K(m) \) the information on specific actions is lost, which is common in PCTL/DTMC based approaches; furthermore, we note that, by construction, \( K(m) \) 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 \( \Upsilon = (\Delta\_\Upsilon, F\_\Upsilon, \Sigma_\Upsilon)^{(N)} \), the translation returns the FlyFast system specification \( \langle \Delta, A, C_0 \rangle^{(N)} \) 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 \( \Upsilon \) is annotated with an identifier which is unique within the specification. We let \( \mathcal{R}(\Upsilon) \) 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 \( \alpha() \) in outbox \( (\gamma, \pi, \alpha()) \) must match with any input

---

\(^{6}\)The concrete FlyFast syntax is: \text{state} \( C[a\_1.C\_1 + a\_2.C\_2 \ldots a\_r.C\_r] \).

\(^{7}\)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 \( \alpha() \) would actually correspond to \( (\alpha, 1)^*[\pi][] \) and \( \alpha() \) would actually correspond to \( (\alpha, 1)^*[\pi][] \). Apart from this detail, the probabilistic semantics as defined in \( \mathbb{9} \) 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 \( \mathbb{9} \) and is applied to \( \kappa(\mathcal{Y}) \). We let \( \mathcal{S}(\kappa(\mathcal{Y})) \) denote the result of the translation, namely the pure FlyFast system specification \( (\Delta, A, C_0)^{(\mathbb{N})} \).

We recall here some notation from \( \mathbb{9} \). We let \( \mathcal{S}_{\Delta} \) denote the set of states of \( \mathcal{Y} \); \( \Gamma_{\Delta} \) is the set of all stores defined over the attributes of \( \mathcal{Y} \)—a store is a finite mapping from the attributes of the component to a finite set of values \( \mathcal{V} \), thus \( \Gamma_{\Delta} \) is finite—and \( \mathcal{O}_{\Delta} \) the finite set of all outboxes of \( \mathcal{Y} \). A \( \mathcal{Y} \) component-state is a triple \( (C, \gamma, O) \in \mathcal{S}_{\Delta} \times \Gamma_{\Delta} \times \mathcal{O}_{\Delta} = \Omega_{\Delta} \). If the component-state is the target of a transition modelling the execution of an output action, then \( O = (\gamma', \pi, \alpha()) \), where \( \gamma' \) is the store of the (component-state) source of the transition, \( \pi \) is the predicate used in the action—actualised with \( \gamma' \)—and \( \alpha() \) the actual message sent by the action. If, instead, the component-state is the target of an input action, then \( O = () \), i.e. the empty outbox. Note that the set of component states of \( \kappa(\mathcal{Y}) \) is identical to that of \( \mathcal{Y} \). Also the set of all stores of \( \kappa(\mathcal{Y}) \) is the same as that of \( \mathcal{Y} \). 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 \( \text{PROD}\{t|\text{cond}(i)\} \), denoting the syntactical product \( t_1 * \ldots * t_n \) if \( \{t|\text{cond}(i) = \text{true}\} = \{t_1, \ldots, t_n\} \neq \emptyset \) and 1 otherwise. Similarly, \( \text{SUM}\{t|\text{cond}(i)\} \) denotes the syntactical sum \( t_1 + \ldots + t_n \) if \( \{t|\text{cond}(i) = \text{true}\} = \{t_1, \ldots, t_n\} \neq \emptyset \) and 0 otherwise. The translation algorithm uses a few auxiliary functions which we briefly discuss below:

- \( \mathcal{I}_{} : \Omega_{\Delta} \rightarrow \mathcal{S} \) is a total injection which maps every component state of \( \kappa(\mathcal{Y}) \) to a distinct state of \( \mathcal{S}(\kappa(\mathcal{Y})) \); we recall that \( \mathcal{S} \) denotes the set of state names of FlyFast models.
- \( \mathcal{I}_{\mathcal{S}} : (\mathcal{S}_{\Delta} \times \Gamma_{\Delta}) \times (\Lambda_{\Delta} \times \mathcal{I}_{\overline{K}}) \times \Omega_{\Delta} \rightarrow \mathcal{S} \) is a total injection where, as in \( \mathbb{9} \), \( \Lambda_{\Delta} \) is the set of action labels of \( \mathcal{Y} \) and \( \mathcal{I}_{\overline{K}} \) is the set of unique identifiers used in the first phase of the translation. We recall that \( \mathcal{S} \) 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 \( \Omega_{\Delta} \). 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 \( \Omega_{\Delta} \) to \( (\mathcal{S}_{\Delta} \times \Gamma_{\Delta}) \).
- The interpretation functions defined in Figure 3, namely those depending on stores only (and not on occupancy measure vectors); we assume \( \mathcal{E}_L[\cdot]_{\mathcal{Y}} \) extended to \( \mathcal{E}_L[fn]_{\mathcal{Y}} \) for defined function \( fn \), in the standard way. In Figure 3 \( \beta_T \) denotes the constant to value bindings generated by the const construct in the input model specification \( \mathcal{Y} \), whereas store update \( upd \) is defined as above.
- The translation function \( \mathcal{I}_{} \) 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, \((\text{inf}, 1)^*[\downarrow()]\) 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 \( \{\text{loc} \} \rightarrow \{A, B, C, D\} \). 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
on the action $\xi$ associated to local position $A$ (i.e. $\gamma = \operatorname{loc} \mapsto A$) and possible next position $B$ (i.e. $\gamma' = \operatorname{loc} \mapsto B$); the algorithm will generate the FlyFast probability function definition $\xi :: \rho W(A) * (\text{frc} \operatorname{II} + \ldots + \text{frc} \operatorname{In})$ as well as a transition leading to (a state which is the encoding, via $\mathcal{I}$, of) the component state with 1 as (proper) state, store $\gamma'$, and outbox ($\gamma, \bot, \inf \{\}$. 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 $\gamma$. 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 \textbf{rest} case is straightforward. Note that for every $\xi :: r * q \in A_{\gamma}$, $r$ 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 $\xi_j :: r_i * q \in A_{\gamma}$, then also $\xi_j :: r_j * q \in A_{\gamma}$ for all $j = 1, \ldots, n$, $j \neq i$ with $\sum_{j=1}^{n} r_j = 1$. Thus the remaining probability is $q_j = (1 - \text{SUM}\{q | \xi_j :: r * q \in A_{\gamma}\})$, where $q$ is either a term $\mathcal{I}(p_j)_{\gamma}$, with $p_j$ occurring in a summand of the state defining equation (see step 1), or a term $\mathcal{I}(p_j)_{\gamma} \times \text{SUM}\{\text{frc} (\mathcal{I}(\mathcal{S})) | \ldots \}$ (see step 2). It worth

\footnote{Here we assume that $\mathcal{I}(\{(C, \gamma, O) | C = 1\}) = \{I_1, \ldots, I_n\}$.}
\[
\mathcal{J}_\gamma(v_p) = v_p \\
\mathcal{J}_\gamma(c_p) = \beta_\gamma(c_p) \\
\mathcal{J}_\gamma(\text{fr}(c)) = \text{SUM}\{\text{fr}(\mathcal{J}_\gamma((C', \gamma', O')))) | (C', \gamma', O') \in \Omega_\Lambda_f \}
\]

\[
\mathcal{J}_\gamma(\prod_{i \in I} p_i) = \text{PROD}\{\mathcal{J}_\gamma(p_i) | i \in I\}
\]

\[
\mathcal{J}_\gamma(\sum_{i \in I} p_i) = \text{SUM}\{\mathcal{J}_\gamma(p_i) | i \in I\}
\]

**Figure 4:** Transition probability expressions translation function definition

---

For each state equation \(C := \sum_{i \in J} \delta_j / p_j :: \text{act}, C_j \) in \(\Lambda_f\):

1. For each output action \((\alpha, \gamma)^* [\pi]/\sigma = \text{act}\_k \in J\) with \(k \in J\) and \([g_k]p_k \neq \text{rest}\), for each \(\gamma \in \Gamma_{\Lambda_f}\) s.t. \(E_k[Eu_{\gamma}] = \text{tt}\) and \((C, \gamma, O) \in \Omega_{\Lambda_f}\), for some \(O \in \Theta_{\Lambda_f}\) for each \(\gamma' \in \Gamma_{\Lambda_f}\) s.t. \((C, \gamma', \gamma, E_k[Eu_{\gamma}]) \in \Omega_{\Lambda_f}\) and \(E_k[\sigma]_{\gamma'}^{(\gamma')} > 0\), let \(\xi = \mathcal{J}_\gamma((C, \gamma, \gamma, \gamma', (\gamma, \gamma, E_k[Eu_{\gamma}])) \in \Omega_{\Lambda_f}\) and add the following action probability function definition in the FlyFast model specification \(\mathcal{J}(\mathcal{R}(\gamma)) = (\Delta.A, \mathcal{C}_0)^{(\gamma)}\) and add the following action probability function definition in \(A\): \(\xi := E_k[\sigma]_{\gamma'}^{(\gamma')} \ast \mathcal{J}_\gamma(p_k)_{\gamma}^{\gamma'}\)

Moreover, for each output \(O \in \Theta_{\Lambda_f}\) s.t. \((C, \gamma, O) \in \Omega_{\Lambda_f}\), the following summand is added to the equation in \(\Lambda\) for state \(\mathcal{J}_\gamma((C, \gamma, O)) \in \Lambda\): \(\xi := E_k[\sigma]_{\gamma'}^{(\gamma')} \ast \mathcal{J}_\gamma(p_k)_{\gamma}^{\gamma'}\)

2. For each input action \((\alpha, \gamma)^* [\pi]/\sigma = \text{act}\_k \in J\) with \(k \in J\) and \([g_k]p_k \neq \text{rest}\), for each \(\gamma \in \Gamma_{\Lambda_f}\) s.t. \(E_k[Eu_{\gamma}] = \text{tt}\) and \((C, \gamma, O) \in \Omega_{\Lambda_f}\) for some \(O \in \Theta_{\Lambda_f}\) for each \(\gamma' \in \Gamma_{\Lambda_f}\) s.t. \((C, \gamma', \gamma, E_k[Eu_{\gamma}]) \in \Omega_{\Lambda_f}\) and \(E_k[\sigma]_{\gamma'}^{(\gamma')} > 0\),

   - Let \(\xi = \mathcal{J}_\gamma((C, \gamma, \gamma, \gamma', (\gamma, \gamma, E_k[Eu_{\gamma}])) \in \Omega_{\Lambda_f}\)
   - Let \(\xi := E_k[\sigma]_{\gamma'}^{(\gamma')} \ast \mathcal{J}_\gamma(p_k)_{\gamma}^{\gamma'}\)

Moreover, for each output \(O \in \Theta_{\Lambda_f}\) s.t. \((C, \gamma, O) \in \Omega_{\Lambda_f}\), the following summand is added to the equation in \(\Lambda\) for state \(\mathcal{J}_\gamma((C, \gamma, O)) \in \Lambda\): \(\xi := E_k[\sigma]_{\gamma'}^{(\gamma')} \ast \mathcal{J}_\gamma(p_k)_{\gamma}^{\gamma'}\)

3. If there exists \(k \in J\) s.t. \([g_k]p_k = \text{rest}\), and \(\text{act}\_k = (\alpha, \gamma)^* [\pi]/\sigma\) for each \(\gamma \in \Gamma_{\Lambda_f}\) s.t. \((C, \gamma, O) \in \Omega_{\Lambda_f}\) for some \(O \in \Theta_{\Lambda_f}\), let \(\Lambda_f\) be the set of probability function definitions which has been constructed in steps (1) and (2) above. Let \(q_F\) be defined by \(q_F = (1 - \text{SUM}\{\xi : r * q \in \Lambda_f\})\). For all \(\gamma \in \Gamma_{\Lambda_f}\) s.t. \((C, \gamma, \gamma, E_k[Eu_{\gamma}]) \in \Omega_{\Lambda_f}\), let \(\xi = \mathcal{J}_\gamma((C, \gamma, \gamma, E_k[Eu_{\gamma}])) \in \Omega_{\Lambda_f}\), be a fresh new action in the FlyFast model specification \(\mathcal{J}(\mathcal{R}(\gamma)) = (\Delta.A, \mathcal{C}_0)^{(\gamma)}\) and add the following action probability function definition in \(A\): \(\xi := E_k[\sigma]_{\gamma'}^{(\gamma')} \ast \mathcal{J}_\gamma(p_k)_{\gamma}^{\gamma'}\)

Moreover, for each output \(O \in \Theta_{\Lambda_f}\) s.t. \((C, \gamma, O) \in \Omega_{\Lambda_f}\), the following summand is added to the equation in \(\Lambda\) for state \(\mathcal{J}_\gamma((C, \gamma, O)) \in \Lambda\): \(\xi := E_k[\sigma]_{\gamma'}^{(\gamma')} \ast \mathcal{J}_\gamma(p_k)_{\gamma}^{\gamma'}\)

4. No other action probability function definition and transition is included and the initial state \(\mathcal{C}_0\) of \(\mathcal{J}(\gamma)\) is defined as \(\mathcal{C}_0 = \mathcal{J}(\mathcal{C}_0)\).

---

**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 DTMCS 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 \(\Omega_{\Lambda_f}, \Gamma_{\Lambda_f}\) and \(\Theta_{\Lambda_f}\) are used. Of course, an alternative approach could be one which considers only the set \(\Omega_{\Lambda_f}\) of component states which are reachable from a given initial component state and, consequently, the sets \(\Gamma_{\Lambda_f}\) and \(\Theta_{\Lambda_f}\) 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 $\langle \Delta, A, C_0 \rangle^{(N)}$. The restricted syntax for transition probability expressions $p$ we use in this section is the following: $p := e_p \mid e_p \cdot \text{frc} (C) \mid e_p \cdot \text{frc} (\pi)$ and $e_p := v_p \, \cdot \, c_p$ where $v_p$ and $c_p$ and $\pi$ are defined as in Section 2.

By inspection of the FlyFast translation as defined in Section 3 and recalling that the set $\mathcal{S}_\Delta$ of the states of the resulting FlyFast model, ranged over by $z, z_1, \ldots$, 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 $\xi :: k \cdot \text{SUM}\{\text{frc}(z_i) \mid i \in I\}$ where $k$ is a FlyFast constant. Moreover, if $p$ was of the form $e_p \cdot \text{frc}(C)$, then index set $I \subseteq \{1, \ldots, S\}$ identifies those states in $\mathcal{S}_\Delta$ that represent (via $\mathcal{I}_\rho$) component states with proper local state $C$; if instead, $p$ was of the form $e_p \cdot \text{frc}(\pi)$, then $I \subseteq \{1, \ldots, S\}$ identifies those states in $\mathcal{S}_\Delta$ that represent (via $\mathcal{I}_\rho$) component states with a store satisfying $\pi$ in the relevant store. At the FlyFast semantics level, recalling that $\text{frc}(z_i)$ is exactly the $i$-th component $m_i$ of the occupancy measure vector $m = (m_1, \ldots, m_S)$ of the model, we can rewrite the above as $k$ or $k \cdot \Sigma_{i \in I} m_i$.

Similarly, the probability action definition in the result of a translation of a generic input action $(\alpha, 1)^+ [\pi'](\gamma')$ (executed in local store $\gamma'$) will necessarily be of the form $k \cdot (\Sigma_{j \in I} m_j)$ or of the form $k \cdot (\Sigma_{j \in I} m_j \cdot (\Sigma_{j \in I'} m_j))$, for index sets $I$ as above and $I'$ as follows:

$$I' = \{i \in \{1, \ldots, S\} \mid \exists C, \gamma, \pi, \pi', \text{s.t. } z_i = \mathcal{I}_\rho((C, \gamma, (\pi, \pi', \alpha')))) \land E_R[\pi'] \gamma' = E_R[\pi'] \gamma = \text{tt}\}$$

An immediate consequence of using the above mentioned restricted syntax for the probability function definitions is that, letting $K : \forall^S \times \mathcal{I} \times \mathcal{I} \to [0, 1]$ be the transition probability matrix for the FlyFast translation of a model specification, we have that $K(m_1, \ldots, m_S)_{z, z'}$ 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 $\mathcal{I}$, with $|\mathcal{I}| = S$, let $K : \forall^S \times \mathcal{I} \times \mathcal{I} \to [0, 1]$ and, for $z \in \mathcal{I}$ and $Q \subseteq \mathcal{I}$, write $K(m)_{z, Q}$ for $\Sigma_{z' \in Q} K(m)_{z, z'}$. Let furthermore $\mathcal{L} : \mathcal{I} \to 2^{\forall P}$ be a state-labelling function, for a given set $\forall P$ of atomic propositions. An equivalence relation $R \subseteq \mathcal{I} \times \mathcal{I}$ is called a bisimulation relation if and only if $z_1 R z_2$ implies: (i) $\mathcal{L}(z_1) = \mathcal{L}(z_2)$ and (ii) $K(m)_{z_1, Q} = K(m)_{z_2, Q}$, for all $m \in \forall^S$ and $Q \in \mathcal{I}/R$. The bisimulation equivalence on $\mathcal{I}$ is the largest bisimulation relation $R \subseteq \mathcal{I} \times \mathcal{I}$.

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.

---

10With a little notational abuse using $k$ also as the actual value in $[0, 1]$ of the FlyFast constant $k$. 
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 \( K(m_1, \ldots, m_S)_{z_i,Q} = K(m_1, \ldots, m_S)_{z_i,Q} \) for all \((m_1, \ldots, m_S) \in \mathcal{U}^S \) 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. \( m_1 < \ldots < m_S \)) and get expressions of the general form \( (\sum_{i=1}^S \sum_{j \geq i}^S h_{ij} \cdot m_i \cdot m_j) + (\sum_{i=1}^S h_i \cdot m_i) + h \) for suitable \( h_{ij}, h_i, h \). Actually, such expressions can always be rewritten in the form \( (\sum_{i=1}^S \sum_{j \geq i}^S u_{ij} \cdot m_i \cdot m_j) + u \) for suitable \( u_{ij}, u \) since, recalling that \( \sum_{i=1}^S m_i = 1 \), we get \( \sum_{i=1}^S h_i \cdot m_i = (\sum_{i=1}^S m_i) \cdot (\sum_{i=1}^S h_i \cdot m_i) \) which, by simple algebraic manipulation, yields an expression of the following form: \( (\sum_{i=1}^S \sum_{j \geq i}^S u_{ij} \cdot m_i \cdot m_j) \); finally, we get \( (\sum_{i=1}^S \sum_{j \geq i}^S u_{ij} \cdot m_i \cdot m_j) + u \) by letting \( u_{ij} = h_{ij} + u'_{ij} \) and \( u = h \). The following proposition thus establishes decidability of \( K(m_1, \ldots, m_S)_{z_i,Q} = K(m_1, \ldots, m_S)_{z_i,Q} \) for all \((m_1, \ldots, m_S) \in \mathcal{U}^S \) for transition probability matrices as in Section 4.

**Proposition 1**

Let \( A(m_1, \ldots, m_S) = (\sum_{i=1}^S \sum_{j \geq i}^S a_{ij} \cdot m_i \cdot m_j) + a \) and \( B(m_1, \ldots, m_S) = (\sum_{i=1}^S \sum_{j \geq i}^S b_{ij} \cdot m_i \cdot m_j) + b \) 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: \( (\forall m_1, \ldots, m_S)(A(m_1, \ldots, m_S) = B(m_1, \ldots, m_S)) \iff ((\forall i, j = 1, \ldots, S \text{ with } i \leq j)(a_{ij} = b_{ij}) \land a = b) \).

The above results can be used for reducing 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 [3]. 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.
The application of the translation to the specification of Example 1 generates an agent model with 8 states, say \( \mathcal{A} = \{SA, SB, SC, SD, IA, IB, IC, ID\} \) with associated IDTMC probability transition matrix as shown in Figure 7 where \( m_{xy} \) represents the fraction of objects currently in state \( x \) and with \( loc = y \) in the original specification of Fig 2—so that \( m = (m_{SA}, m_{SB}, m_{SC}, m_{SD}, m_{IA}, m_{IB}, m_{IC}, m_{ID}) \) is the occupancy measure vector. In Figure 7 functions \( \phi_2 \) and \( \phi_1 \) are used as abbreviations in the obvious way: \( \phi_2(m) = m_{SA} + m_{SB} + m_{SC} + m_{SD} \) and \( \phi_1(m) = m_{IA} + m_{IB} + m_{IC} + m_{ID} \). 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 \( Sh, I, Sl \) and \( II \) and a labelling \( \mathcal{L} \) such that \( \mathcal{L}(SA) = \mathcal{L}(SC) = \{Sh\}, \mathcal{L}(IA) = \mathcal{L}(IC) = \{I\}, \mathcal{L}(SB) = \mathcal{L}(SD) = \{Sl\}, \) and \( \mathcal{L}(IB) = \mathcal{L}(ID) = \{II\} \).

Consider relation \( R \) on \( \mathcal{A} \) defined as \( R = I_{\mathcal{A}} \cup \{(SA, SC), (SB, SD), (IA, IC), (IB, ID)\} \cup \{(SC, SA), (SD, SB), (IC, IA), (ID, IB)\} \) where \( I_{\mathcal{A}} \) is the identity relation on \( \mathcal{A} \). It is very easy to show that \( R \) is a bisimulation according to Definition 1. Clearly \( R \) is an equivalence relation and its quotient \( \mathcal{A}/R \) 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 \( z_1, z_2 \in \mathcal{A}/R \), whenever \( z_1 R z_2 \), we have \( \mathcal{L}(z_1) = \mathcal{L}(z_2) \) and \( \mathcal{K}(m)_{z_1} = \mathcal{K}(m)_{z_2} \) for all \( Q \in \mathcal{A}/R \) and for all \( m \), as one can easily check; clearly, \( R \) is also the largest bisimulation relation on \( \mathcal{A} \).

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 \( \mathcal{K}(m_{Q_{sh}}, m_{Q_{sl}}, m_{Q_{ih}}, m_{Q_{il}}) \) shown in Figure 9. It corresponds to the FlyFast agent specification \( \hat{\Delta} \) 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. \( \mathcal{L}(SA) = \mathcal{L}(SC) = \mathcal{L}'(IA) = \mathcal{L}'(IC) = \{h\}, \mathcal{L}'(SB) = \mathcal{L}'(SD) = \mathcal{L}'(IB) = \mathcal{L}'(ID) = \{l\} \) would make the model collapse into one with only two states, \( Q_h \) and \( Q_l \), with probabilities \( H: Q_h \rightarrow Q_h, H: Q_l \rightarrow Q_h, L: Q_h \rightarrow Q_l \) and \( L: Q_h \rightarrow Q_l \), where only the location would be modelled whereas

---

11For instance, on page 227 of \( \mathcal{L} \), line 12, \( v(x, S) = v(y, S) \) should be replaced with \( L(x) = L(y) \land v(x, S) = v(y, S) \) and in line 13, \( v(x, S) \neq v(y, S) \) should be replaced with \( L(x) \neq L(y) \lor v(x, S) \neq v(y, S) \), in order to take state labels into consideration as well. Of course \( v(x, S) \) (\( L \), respectively) is to be intended as \( \mathcal{K}(m)_z \) (\( \mathcal{L}' \), respectively), using the notation we introduced above for Bisimilarity.

12Actually 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.
Designing Attribute-based FlyFast

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 \( \hat{\Delta} \)

|        | \( Q_{Sh} \) | \( Q_{Sl} \) | \( Q_{Ih} \) | \( Q_{Il} \) |
|--------|-------------|-------------|-------------|-------------|
| \( Q_{Sh} \) | H \cdot (m_{QSh} + m_{QSh}) | L \cdot (m_{QSh} + m_{QSh}) | H \cdot (m_{QSh} + m_{QSh}) | L \cdot (m_{QSh} + m_{QSh}) |
| \( Q_{Sl} \) | H \cdot (m_{QSh} + m_{QSh}) | L \cdot (m_{QSh} + m_{QSh}) | H \cdot (m_{QSh} + m_{QSh}) | L \cdot (m_{QSh} + m_{QSh}) |
| \( Q_{Ih} \) | H \cdot ir | L \cdot ir | H \cdot ir | L \cdot ir |
| \( Q_{Il} \) | H \cdot ir | L \cdot ir | H \cdot ir | L \cdot ir |

Figure 9: IDTMC transition probability matrix function \( \tilde{K}(\mathbf{m}) \), for \( \mathbf{m} \) in \( \mathcal{M}^4 \).

information on the infection status would be lost.

7 Conclusions

Pi\text{FF} [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 Pi\text{FF} 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 continuous-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 Evaluaition 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 Tizzi & Andrea Vandin (2015): *The SCEL Language: Design, Implementation, Verification*. In Martin Wirsing, Matthias M. Hötzl, 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 Muralpha 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. Havorkert (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:10.1007/978-3-540-24622-0_8.

[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, ISO-LA 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 Martin 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
A Appendix

Proof of Proposition

$\iff$: Trivial.

$\implies$:

We first prove that $a = b$:

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

$\implies \{ \text{Logic} \}$

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

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

$a = b$

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

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

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

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

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

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

$\implies \{ a = b \text{ (see above)} \}$

$a_{ii} = b_{ii}$

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

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

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

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

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

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

$\implies \{ a = b \text{ (see above)} \}$

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

$\implies \{ \text{Algebra} \}$

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

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

$a_{ij} = b_{ij}$