QUANTICOL

A Quantitative Approach to Management and Design of Collective and Adaptive Behaviours

quantiCOL

http://www.quanticol.eu

TR-QC-01-2017

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

Preliminary version

Revision: 0.0; January 16, 2017

Author(s): Diego Latella (CNR), Mieke Massink (CNR)

Publication date: January 16, 2017

Funding Scheme: Small or medium scale focused research project (STREP)
Topic: ICT-2011 9.10: FET-Proactive 'Fundamentals of Collective Adaptive Systems' (FOCAS)
Project number: 600708

Coordinator: Jane Hillston (UEDIN)
e-mail: Jane.Hillston@ed.ac.uk
Fax: +44 131 651 1426

Part. no.	Participant organisation name	Acronym	Country
1 (Coord.)	University of Edinburgh	UEDIN	UK
2	Consiglio Nazionale delle Ricerche – Istituto di Scienza e Tecnologie della Informazione "A. Faedo"	CNR	Italy
3	Ludwig-Maximilians-Universität München	LMU	Germany
4	Ecole Polytechnique Fédérale de Lausanne	EPFL	Switzerland
5	IMT Lucca	IMT	Italy
6	University of Southampton	SOTON	UK
7	Institut National de Recherche en Informatique et en Automatique	INRIA	France

COOPERATION

Contents

Abstract

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 [8, 7], Latella et al. presented a scalable mean-field model-checking procedure for verifying bounded Probabilistic Computation Tree Logic (PCTL, [4]) 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, approximated, model-checking based on the idea of fast simulation, as introduced in [9]. 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 $\mathbf{K}(\boldsymbol{\mu}(t))$ where $\mathbf{K}(\mathbf{m})$ is the $S \times S$ probability transition matrix of a (inhomogeneous) DTMC and $\boldsymbol{\mu}(t)$ is a vector of size S approximating the mean behaviour of the rest of the system at t; each element of $\boldsymbol{\mu}(t)$ is associated with a distinct state of the agent, say C, and gives an approximation of the 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 *determinstic* one, i.e. $\boldsymbol{\mu}$ is a *function* of the step t (the exact behaviour of the rest of the system

[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/`

would instead be a *large* DTMC in turn); note furthermore, that the above transition matrix does not depend on N [8, 7].

Recently, modelling and programming languages have been proposed specifically for autonomic computing systems and CAS [3, 1]. 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 that the association between attribute *names* and attribute *values* is mantained 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 [6], and forming the basis of interaction schemes in languages like SCEL [3] and CARMA [1].

In [2] 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, 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 γ 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 γ 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 rewrite 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 γ 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.

In this paper we present some details of PiFF, a translation to FlyFast which simplifies that proposed in [2] and an approach to state-space reduction based on probabilistic bisimulation for Inhomogeneous DTMCs. In Section 2 we present the main ingredients of the PiFF definition and we recall those features of FlyFast directly relevant for understanding the translation PiFF to the FlyFast input language proposed in [2]. 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 state 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.

⋮
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 : B; D : B **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}.

2.1 PiFF

An PiFF system model specification $\Upsilon = (\Delta_\Upsilon, F_\Upsilon, \mathbf{\Sigma_0})^{(N)}$ 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 $\mathbf{\Sigma_0}$ 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 [2] for additional information.
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 locations. Some auxiliary constants are defined, using the **const** construct inherited from FlyFast: **const** $<$ name $>$ = $<$ value $>$.

An PiFF store update definition has the following sytax[3]:

[3] In [2] a slightly different syntax for store updates has been used.

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 $e ::= v_a \mid c_a \mid \mathbf{my}.a \mid fn_a(e_1, \ldots, e_m)$ 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 $\mathcal{V}$ of attribute values), c_a is an attribute constant in $\mathcal{V}$ 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 attribite-values.

In Figure 1 attribute functions $\mathtt{N}, \mathtt{S}, \mathtt{E}, \mathtt{W}$ are defined for North, South, East, and West, such that `Space` models the Cartesian space with four quadrants: $\mathtt{A} = \mathtt{N}(\mathtt{D}) = \mathtt{E}(\mathtt{B})$, $\mathtt{B} = \mathtt{N}(\mathtt{C}) = \mathtt{W}(\mathtt{A})$, and so on, as shown diagramatically 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 an store-probability function defined by the user in F_Υ, which, when applied to attribute expressions $e_1, \ldots, e_m$ returs 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 to 1[4]. 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 $\mathtt{pHr}, \mathtt{pN}, \mathtt{pS}, \mathtt{pE}, \mathtt{pW}$ are defined that give the probability of not moving or of jumping to North, South, East, West, respectively, as a function of the current location.

Example 1 *A simplified version of the behaviour of the epidemic process discussed in [2] is shown in Fig. 2* left[5]*. In Fig. 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

[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.

[5]We focus only on those features that are most relevant for the present paper. In [2] also other features are shown like, e.g. the use of (predicate-based) input actions, which are not the main subject of this paper.

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

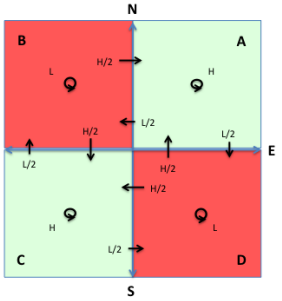

Figure 2: SI, a behavioural model.

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 the condition* $\mathtt{H} > \mathtt{L}$ *implies that higher probability is assigned to* A *and* C *and low probability to* B *and* D *(see Figure 1* right*). 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.* $\mathtt{frc}(\mathtt{I})$*); it remains in state* S *via the self-loop labelled by action* nsc*, with probability* $\mathtt{frc}(\mathtt{S}) = 1 - \mathtt{frc}(\mathtt{I})$*. 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 on the specific component (i.e., its initial location), predicate $\bot$ and the empty tuple $\langle\rangle$).*

An 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 $\mathcal{V}$ of attribute values), c_a is an attribute constant in $\mathcal{V}$ 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. $e ::= v_a \mid c_a \mid \mathbf{my}.a \mid a$). Note that, in probability expression p_j, $\mathsf{frc}\,(C)$ ($\mathsf{frc}\,(\pi)$, respectively) is the fraction of components *currently* in state C (the *current* store of which satisfies π, respectively), over the total number N and that it must be guaranteed that $\prod_{i\in I} p_i \leq 1$ and $\sum_{i\in I} p_i \leq 1$.

- 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 $[g_j]p_j$ = **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 $\langle\Delta, A, \mathbf{C}_0\rangle^{\langle N\rangle}$ 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 $a_i \in \mathcal{A}$—the set of FlyFast *actions*—$C, C_i \in \mathcal{S}$—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 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 characterizes 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 $\mathcal{S}_\Delta$ be the set of states of the agent, with $|\mathcal{S}_\Delta| = S$, and $\mathcal{U}^S = \{(m_1, \ldots, m_S) | m_1 + \ldots + m_S = 1\}$ denote the unit simplex of dimension S, we have $\mathbf{K} : \mathcal{U}^S \times \mathcal{S}_\Delta \times \mathcal{S}_\Delta \to [0, 1]$. 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 [7, 8] for details).

3 A revised translation

As in [2], we define a translation such that, given an PiFF system specification $\Upsilon = (\Delta_\Upsilon, F_\Upsilon, \mathbf{\Sigma_0})^{(N)}$, 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 [2].

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 [2] 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 action $\alpha()$ even if $\alpha\langle\rangle$ would actually correspond to $(\alpha, \iota)^*[\pi]\langle\rangle$ and $\alpha()$ would actually correspond to $(\alpha, \eta)^*[\pi'](\rangle$. Apart from this detail, the probabilistic semantics as defined in [2] remain unchanged.

The second phase is defined by the translation algorithm defined in Fig. 5, which is a revised and simplified version of that presented in [2] and is applied to $\aleph(\Upsilon)$. We let $\mathcal{I}(\aleph(\Upsilon))$ 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 [2]. We let $\mathcal{S}_{\Delta_\Upsilon}$ 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 $\mathcal{V}$, thus Γ_{Δ_Υ} is finite—and $\mathcal{O}_{\Delta_\Upsilon}$ the finite set of all outboxes of Υ. A Υ *component-state* is a triple $(C, \gamma, O) \in \mathcal{S}_{\Delta_\Upsilon} \times \Gamma_{\Delta_\Upsilon} \times \mathcal{O}_{\Delta_\Upsilon} = \Omega_{\Delta_\Upsilon}$. 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—actualized 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 Fig. 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

[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 subpolulation currently in that state over the size of the overall population. The occupancy measure vector is a compact representation of the system global state.

translation algorithm uses a few auxiliary functions which we briefly discuss below:

- $\mathcal{I}_{\mathcal{S}} : \Omega_{\Delta_\Upsilon} \to \mathcal{S}$ is a total injection which maps every component state of $\aleph(\Upsilon)$ to a distinct state of $\mathcal{I}(\aleph(\Upsilon))$; we recall that $\mathcal{S}$ denotes the set of state names of FlyFast models.

- $\mathcal{I}_{\mathcal{A}} : (\mathcal{S}_{\Delta_\Upsilon} \times \Gamma_{\Delta_\Upsilon}) \times (\Lambda_{\Delta_\Upsilon} \times I_{\aleph}) \times \Omega_{\Delta_\Upsilon} \to \mathcal{A}$ is a total injection where, as in [2], $\Lambda_{\Delta_\Upsilon}$ 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 $\mathcal{A}$ 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 characterized 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 $(\mathcal{S}_{\Delta_\Upsilon} \times \Gamma_{\Delta_\Upsilon})$.

- The interpretation functions defined in Fig. 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 Fig. 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 $\mathcal{I}_{\mathcal{P}}$ for transition probability expressions p_j, defined in Fig. 4.

Output actions are dealt with in step 1 of the algorithm of Fig. 5. Take for example $(\mathtt{inf}, 1)^*[\bot]\langle\rangle\,\mathtt{Jump}$ in the definition of state `S` in Fig. 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 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 $\mathcal{I}_{\mathcal{S}}$, 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 straighforward. 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 $\mathcal{I}_{\mathcal{P}}(p_j)_\gamma$, with p_j occurring in a summand of the state defining equation (see step 1), or a term $\mathcal{I}_{\mathcal{P}}(p_j)_\gamma * \mathsf{SUM}\{\mathsf{frc}\,(\mathcal{I}_{\mathcal{S}}(\Sigma)) | \ldots\}$ (see step 2).

We note that in the algorithm sets Ω_{Δ_Υ}, Γ_{Δ_Υ} and $\mathcal{O}_{\Delta_\Upsilon}$ 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 $\overline{\mathcal{O}_{\Delta_\Upsilon}}$ 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.

[8] Diagonal jumps are not contemplated in the example; technically this comes from the actual probability values used in the definition of `Jump`.

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

$$
\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

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 which 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 as defined 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 $z, z_i, \ldots$, 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

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

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 $O \in \mathcal{O}_{\Delta_\Upsilon}$, 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 $\xi = \mathcal{I}_\mathcal{A}((C,\gamma),(\alpha\langle\rangle,\iota),(C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)))$ be a fresh new action on the FlyFast model specification $\mathcal{I}(\aleph(\Upsilon)) = \langle\Delta,A,\mathbf{C}_0\rangle^{(N)}$ and add the following action probability function definition in A: $\xi :: \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') * \mathcal{I}_\mathcal{P}(p_k)_\gamma$.
Moreover, for each outbox $O \in \mathcal{O}_{\Delta_\Upsilon}$ s.t. $(C,\gamma,O) \in \Omega_{\Delta_\Upsilon}$, the following summand is added to the equation in Δ for state $\mathcal{I}_\mathcal{S}((C,\gamma,O))$: $\xi.\mathcal{I}_\mathcal{S}((C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)))$;

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

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 $O \in \mathcal{O}_{\Delta_\Upsilon}$, 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 $q_\gamma = (1 - \mathsf{SUM}\{q|\zeta :: r * q \in A_\gamma\})$. 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 $\xi = \mathcal{I}_\mathcal{A}((C,\gamma),(\alpha,\iota)\langle\rangle,(C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle))) \in \Omega_{\Delta_\Upsilon}$, be a fresh new action on the FlyFast model specification $\mathcal{I}(\aleph(\Upsilon) = \langle\Delta,A,\mathbf{C}_0\rangle^{(N)}$ and add the following action probability function definition in A: $\xi :: \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') * q_\gamma$.
Moreover, for each outbox $O \in \mathcal{O}_{\Delta_\Upsilon}$ s.t. $(C,\gamma,O) \in \Omega_{\Delta_\Upsilon}$, the following summand is added to the equation in Δ for state $\mathcal{I}_\mathcal{S}((C,\gamma,O))$: $\xi.\mathcal{I}_\mathcal{S}((C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)))$;

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

Figure 5: The translation algorithm

was of the form $e_p \cdot \mathsf{frc}\,(C)$, then index set $I \subseteq \{1,\ldots,S\}$ identifies those states in $\mathcal{S}_\Delta$ that represent (via $\mathcal{I}_\mathcal{S}$) 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 $\mathcal{S}_\Delta$ that represent (via $\mathcal{I}_\mathcal{S}$) 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:

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

An immediate consequence of using the above, restricted, syntax for the probability function definitions is that, letting $\mathbf{K} : \mathcal{U}^S \times \mathcal{S} \times \mathcal{S} \to [0,1]$ 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 $m_1,\ldots,m_S$.

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

5 Bisimilarity and State-space Reduction

The following definition generalizes 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 $\mathcal{S}$, let $\mathbf{K} : \mathcal{U}^S \times \mathcal{S} \times \mathcal{S} \to [0,1]$ and, for $z \in \mathcal{S}$ and $Q \subseteq \mathcal{S}$, write $\mathbf{K}(\mathbf{m})_{z,Q}$ for $\sum_{z' \in Q} \mathbf{K}(\mathbf{m})_{z,z'}$. Let furthermore $\mathcal{L} : \mathcal{S} \to 2^{AP}$ be a state-labelling function, for a given set AP of atomic propositions. An equivalence relation $R \subseteq \mathcal{S} \times \mathcal{S}$ 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) $\mathbf{K}(\mathbf{m})_{z_1,Q} = \mathbf{K}(\mathbf{m})_{z_2,Q}$, for all $\mathbf{m} \in \mathcal{U}^S$ and $Q \in \mathcal{S}/R$. The* bisimulation equivalence *on $\mathcal{S}$ is the largest bisimulation relation $R \subseteq \mathcal{S} \times \mathcal{S}$.*

Of course $\mathbf{K}(m_1, \ldots, m_S)_{z_1,Q} = \mathbf{K}(m_1, \ldots, m_S)_{z_2,Q}$ for all $(m_1, \ldots, m_S) \in \mathcal{U}^S$ 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. $m_1 \prec \ldots \prec m_S$) 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 calculations, 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 $(m_1, \ldots, m_S) \in \mathcal{U}^S$ 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. [5], 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, conseqently, 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.

6 Example

The application of the translation to the specification of Example 1 generates an agent model with 8 states, say $\mathcal{S}_\Delta = \{SA, SB, SC, SD, IA, IB, IC, ID\}$[12] with associated IDTMC probability transition

[11] For instance, on page 227 of [5], 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 $\mathbf{K}(\mathbf{m})_{z,Q}$ ($\mathcal{L}$, respectively), using the notation we introduced above for Bisimilarity.

[12] Actually the agent resulting from the translation of Fig. 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

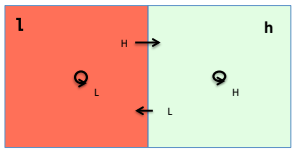

Figure 6: SI in two quadrants

	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 $\mathcal{U}^8$.

matrix as shown in Fig. 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 Fig. 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 $\mathcal{L}$ such that $\mathcal{L}(SA) = \mathcal{L}(SC) = \{Sh\}$, $\mathcal{L}(IA) = \mathcal{L}(IC) = \{Ih\}$, $\mathcal{L}(SB) = \mathcal{L}(SD) = \{Sl\}$, and $\mathcal{L}(IB) = \mathcal{L}(ID) = \{Il\}$.

Consider relation R on $\mathcal{S}_\Delta$ defined as $R = I_{\mathcal{S}_\Delta} \cup \{(SA, SC), (SB, SD), (IA, IC), (IB, ID)\}$ where $I_{\mathcal{S}_\Delta}$ is the identity relation on $\mathcal{S}_\Delta$. It is very easy to show that R is a bisimulation according to Definition 1. Clearly R is an equivalence relation and its quotient $\mathcal{S}_\Delta/R$ 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 $z_1, z_2 \in \mathcal{S}_\Delta$, whenever $z_1 \, R \, z_2$, we have $\mathcal{L}(z_1) = \mathcal{L}(z_2)$ and $\mathbf{K}(\mathbf{m})_{z_1,Q} = \mathbf{K}(\mathbf{m})_{z_2,Q}$ for all $Q \in \mathcal{S}_\Delta/R$ and for all $\mathbf{m}$, as one can easily check; clearly, R is also the largest bisimulation relation on $\mathcal{S}_\Delta$. 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 Fig. 9 which corresponds to the FlyFast agent specification $\widehat{\Delta}$ given in Fig. 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 Fig. 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 $H : Q_h \to Q_h$, $H : Q_l \to Q_h$, $L : Q_h \to Q_l$ and $L : Q_h \to Q_l$ where only the location would be modelled whereas information on the infection status would be lost.

7 Conclusions

In this paper we presented PiFF, a language for the predicate-based front-end of the FlyFast on-the-fly tool for approximated mean-field model-checking. A simplified version of the translation proposed

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 $\mathcal{U}^4$.

in [2] has been presented together with an approach for reducing the state-space of individual agents based on probabilistic bisimularity 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.

References

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

[2] V. Ciancia, D. Latella, and M. Massink. On-the-Fly Mean-field Model-checking for Attribute-based Coordination. In A. Lluch Lafuente and J. Proença, editors, *Coordination Models and Languages*, volume 9686 of *LNCS*, pages 67–83. Springer-Verlag, 2016. DOI: 10.1007/978-3-319-39519-7_5, ISSN: 0302-9743, ISBN: 978-3-319-39518-0 (print), 978-3-319-39519-7 (on line).

[3] R. De Nicola, D. Latella, A. Lluch Lafuente, M. Loreti, A. Margheri, M. Massink, A. Morichetta, R. Pugliese, F. Tiezzi, and A. Vandin. The SCEL Language: Design, Implementation, Verification. In M. Wirsing, M. Hölzl, N. Koch, and P. Mayer, editors, *Software Engineering for Collective Autonomic Systems*, volume 8998 of *LNCS*, chapter I.1, pages 3–71. Springer-Verlag, 2015. DOI: 10.1007/978-3-319-16310-9_1, ISBN 978-3-319-16309-3 (print), 978-3-319-16310-9 (online), ISSN 0302-9743.

[4] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing. The International Journal of Formal Methods. Springer-Verlag*, 6(5):512–535, 1994.

[5] D. Huynh and L. Tian. On some equivalence relations for probabilistic processes. *Fundamenta Informaticae*, 17:211–234, 1992.

[6] D. Latella. Cmunicazione basata su proprietà nei sistemi decentralizzati, 1983. [Property-based inter-process communication in decentralized systems] Graduation Thesis. Istituto di Scienze dell'Informazione. Univ. of Pisa, Italy (in italian).

[7] D. Latella, M. Loreti, and M. Massink. On-the-fly Fast Mean-Field Model-Checking. In M. Abadi and A. Lluch Lafuente, editors, *Trustworthy Global Computing, 4th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers*, volume 8358 of *LNCS*, pages 297–314. Springer, 2014. DOI:10.1007/978-3-319-05119-2_17, ISBN 978-3-319-05118-5 (print), 978-3-319-05119-2 (on-line), ISSN 0302-9743 .

[8] D. Latella, M. Loreti, and M. Massink. On-the-fly PCTL fast mean-field approximated model-checking for self-organising coordination. *Science of Computer Programming. Elsevier*, 110:23–50, 2015. DOI: 10.1016/j.scico.2015.06.009; ISSN: 0167-6423.

[9] Jean-Yves Le Boudec, David McDonald, and Jochen Mundinger. A generic mean field convergence result for systems of interacting objects. In *QEST07*, pages 3–18. IEEE Computer Society Press, 2007. ISBN 978-0-7695-2883-0.

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)}

$a_{ij} = b_{ij}$ •