The FlyFast Mean Field Model Checker[⋆] – Quick User Manual

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

[1] Consiglio Nazionale delle Ricerche - Istituto di Scienza e Tecnologie dell'Informazione 'A. Faedo', CNR, Italy
[2] Università di Firenze, Italy
{diego.latella,mieke.massink}@isti.cnr.it michele.loreti@unifi.it

Abstract. This document provides a brief user manual for FlyFast, the on-the-fly *mean field* model checker for the verification of time-dependent probabilistic properties of individual objects in the context of large populations. FlyFast is provided within the Java StochAstic Model checker (jSAM) framework which offers several other formal analysis techniques among which stochastic simulation and fast simulation.

Keywords: Mean field model checking; Collective Adaptive Systems; Discrete Time Markov Chains; Self-organisation; Gossip protocols.

1 Introduction

FlyFast is an on-the-fly mean field probabilistic model checker. Its purpose is the automatic verification of time-bounded PCTL (Probabilistic Computation Tree Logic) properties of a *selected individual* in the context of systems that consist of a *large number* of (similar, but) independent, *interacting objects*. Typical examples of such systems are large scale Collective Adaptive Systems (CAS) and distributed algorithms for sharing data in a distributed network, such as gossip protocols. Following the mean field approach proposed in [4], an on-the-fly mean field model checking algorithm was developed and proven correct in [1, 3]. This model checking algorithm has been implemented in FlyFast.

Models that can be analysed by FlyFast are clock-synchronous DTMC-based population models in which each object performs a probabilistic step in each discrete time unit, moving between its local states and possibly returning to the same state. Objects interact in an indirect way, via the global state of the system. In fact, the evolution of the global system is specified by the *local* transition probabilities of an object. The latter are the same for each object in the population (i.e. one abstracts from their identity) and may depend on the occupancy measure vector[3].

When the number of objects is large (at least several hundreds) the overall behaviour, in terms of its occupancy measure vector, can be approximated by the deterministic solution of a difference equation, which is called the 'mean field' [4]. This iterative approach to obtain the occupancy measure vector has shown to combine very well with an on-the-fly probabilistic model checking approach [3]. The model checking algorithm is parametric w.r.t. the semantics interpretation of the model specification language. Both the instantiation on the exact, probabilistic semantics and that on the mean field semantics are available in FlyFast. In the sequel we will mainly focus on the mean field model checking option. The model checking algorithm consists of two phases, namely an expansion phase and a computation phase. Both phases are linear in the number of states and transitions of the expansion of the initial state of the selected object and occupancy measure vector [3] for the time bounded fragment of PCTL. FlyFast has been applied on a.o. bike sharing [3], client-server systems and computer worm epidemic models [2].

⋆ Research partially funded by the EU project QUANTICOL (nr. 600708).

[3] More specifically, the occupancy measure vector consists of a number of elements equal to the number of local states of an object, providing, for each state, the fraction of objects in the total population that are currently in that state.

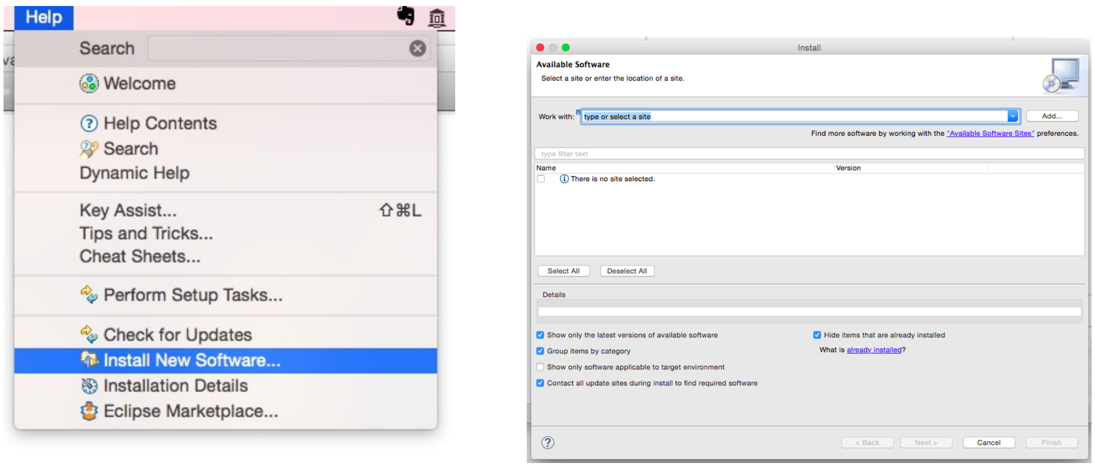
FlyFast is provided within the jSAM (java StochAstic Model Checker) framework which is an open source Eclipse plugin[4] integrating a set of tools for stochastic analysis of concurrent and distributed systems specified using process algebras.

2 Installation Instructions

Prerequisites. FlyFast is provided within the Java Stochastic Model Checker (jSAM) which in turn is provided as an Eclipse Plugin. To install the plugin you need to install Eclipse for DSL[5] (Domain Specific Language based on Xtext) and Java 1.8.

Installation. Once Eclipse is installed, perform the following steps to install FlyFast.

1) Open your copy of Eclipse for DSL, and select Help >> Install New Software, as shown in Fig. 1 (left).



Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
☐ There is no site selected.	

☑ Show only the latest versions of available software

☑ Group items by category

☐ Show only software applicable to target environment

☑ Contact all update sites during install to find required software

☑ Hide items that are already installed

What is already installed?

Fig. 1. Installing new software in Eclipse for DSL.

2) Click Add to open the “add site” dialogue (see Fig. 1 (right) button in the top-right of the figure).

3) Create a jSAM repository and set the location to (see Fig. 2):

```
http://dl.bintray.com/michele-loreti/jSAM/updates/1.1/
```

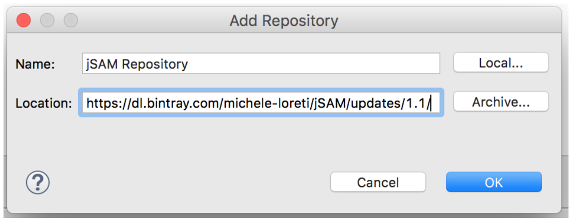
4) Select the packages of interest from the category jSAM *Plugin* and click on the button *Next* (see Fig. 3 (left)).

5) Confirm the list of plugins to install by clicking *Next* (see Fig. 3 (right)). After pressing *Next* all the required features are downloaded and you will be asked to accept the *License* (EPL1 in this case).

6) Accept the *License* by ticking the corresponding option and click *Next* (se Fig. 4 (left)).

7) The installation process continues and when it reaches the end confirm the installation by selecting “OK” (see Fig. 4 (right)).

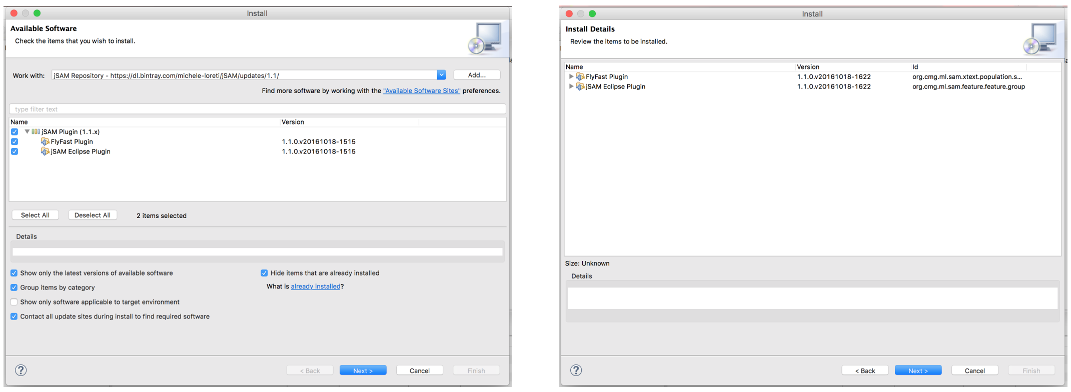
[4] http://quanticol.github.io/jSAM/
[5] www.eclipse.org/home/index.php


Add Repository

Name: jSAM Repository

Location: https://dl.bintray.com/michele-loreti/jSAM/updates/1.1/

Fig. 2. Select new repository


Install

Available Software

Check the items that you wish to install.

Work with: jSAM Repository - https://dl.bintray.com/michele-loreti/jSAM/updates/1.1/

Find more software by working with the "Available Software Sites" preferences.

Name	Version
✓ jSAM Plugin (1.1.x)	
✓ FlyFast Plugin	1.1.0.v20161018-1515
✓ jSAM Eclipse Plugin	1.1.0.v20161018-1515

2 items selected

Details

✓ Show only the latest versions of available software

✓ Group items by category

Show only software applicable to target environment

✓ Contact all update sites during install to find required software

✓ Hide items that are already installed

What is already installed?

Install

Install Details

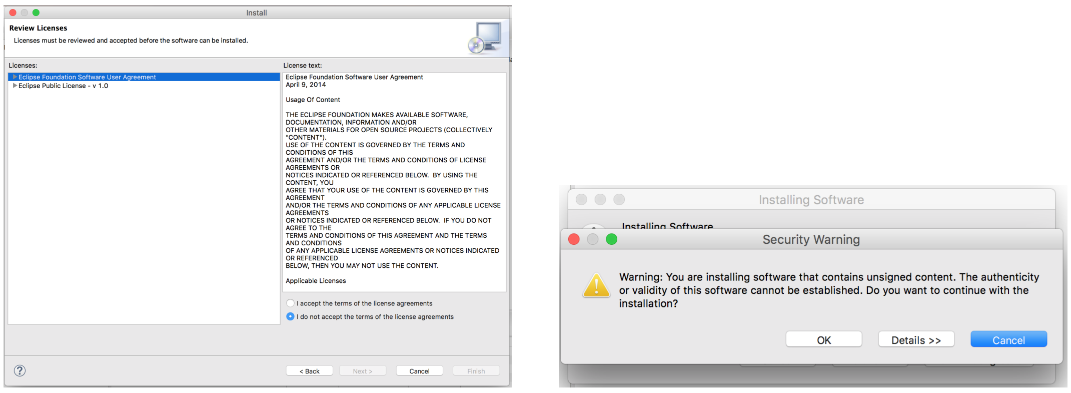
Review the items to be installed.

Name	Version	Id
FlyFast Plugin	1.1.0.v20161018-1622	org.cmg.ml.sam.xtext.population.s...
jSAM Eclipse Plugin	1.1.0.v20161018-1622	org.cmg.ml.sam.feature.feature.group

Size: Unknown

Details

Fig. 3. Select the packages FlyFast Plugin and jSAM Eclipse Plugin (left) and confirm (right)


Install

Review Licenses

Licenses must be reviewed and accepted before the software can be installed.

Licenses:

Eclipse Foundation Software User Agreement

Eclipse Public License - v 1.0

License text:

Eclipse Foundation Software User Agreement
April 9, 2014

Usage Of Content

THE ECLIPSE FOUNDATION MAKES AVAILABLE SOFTWARE, DOCUMENTATION, INFORMATION AND/OR OTHER MATERIALS FOR OPEN SOURCE PROJECTS (COLLECTIVELY "CONTENT"). USE OF THE CONTENT IS GOVERNED BY THE TERMS AND CONDITIONS OF THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. BY USING THE CONTENT, YOU AGREE THAT YOUR USE OF THE CONTENT IS GOVERNED BY THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF ANY APPLICABLE LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. IF YOU DO NOT AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT AND THE TERMS AND CONDITIONS OF ANY APPLICABLE LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW, THEN YOU MAY NOT USE THE CONTENT.

Applicable Licenses

I accept the terms of the license agreements [✓ I do not accept the terms of the license agreements]

Installing Software

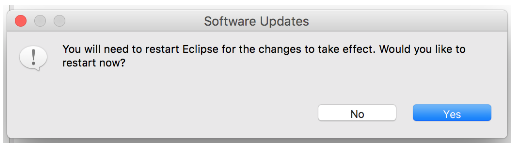
Installing Software

Security Warning

Warning: You are installing software that contains unsigned content. The authenticity or validity of this software cannot be established. Do you want to continue with the installation?

Fig. 4. Accept license and confirm installation

8) Restart Eclipse to enable the new features clicking “Yes” (see Fig. 5).


Software Updates

You will need to restart Eclipse for the changes to take effect. Would you like to restart now?

Fig. 5. Restart Eclipse

3 The FlyFast Language

FlyFast supports a simple specification language for DTMC synchronous population models. It is a state-based language in which the probabilistic behaviour of individual components of one (or more) populations is specified following an approach inspired by Le Boudec, et al. [4]. In this approach the probabilities of the transitions of the individual components may depend on the occupancy measure vector.

The modelling language of FlyFast consists of basic constructs to describe the probabilistic behaviour of an individual object, in particular constants, expressions, states, action probabilities, transitions and initial system configurations. In this section we report the full syntax of the FlyFast modelling language in EBNF form. Below, we will use < symb > to denote a non terminal symbol and we let < name > to denote a valid identifier. Moreover, we let < int > and < real > denote an integer and a real token, respectively.

FlyFast Model

A FlyFast model consists of a number of elements:

```
< model > = < elem >*
```

where `elem` is defined as:

```
< elem > =
   | < const > //Constant definition
   | < actfun > //Action function definition
   | < stateconst > //State constant definition
   | < config > //Configuration definition
   | < proplabel > //Property label definition
   | < stateformula > //State formula definition
   | < pathformula > //Path formula definition
```

We present the syntax for each FlyFast model element shortly, but we first introduce expressions.

Expressions. Integer and real expressions can be combined by using the standard arithmetic operators $+,-,*,/$.

```
< exp > =
     < int >
   | < real >
   | 'true'
   | 'false'
   | < name >
   | < exp > '<' < exp > // smaller than
   | < exp > '<=' < exp > // smaller equal than
   | < exp > '>=' < exp > // greater equal than
   | < exp > '>' < exp > // greater than
   | < exp > '+' < exp > // sum
   | < exp > '-' < exp > // minus
   | < exp > '*' < exp > // multiplcation
   | < exp > '/' < exp > // division
   | '(' < exp > ')'
   | 'acos' '(' < exp > ')' // arc cosine
   | 'asin' '(' < exp > ')' // arc sine
   | 'atan' '(' < exp > ')' // arc tangent
   | 'ceil' '(' < exp > ')' // ceiling
   | 'cos' '(' < exp > ')' // cosine
   | 'floor' '(' < exp > ')' // floor
   | 'frc' '(' < name > ')' // population fraction
   | 'ln' '(' < exp > ')' // natural logarithm
| 'max' '(' <exp> ',' <exp> ')' // maximum
| 'min' '(' <exp> ',' <exp> ')' // minimum
| 'pow' '(' <exp> ',' <exp> ')' // power
| 'sin' '(' <exp> ')' // sine
| 'tan' '(' <exp> ')' // tangent
```

Constant definition

```
<const> = 'const' <name> '=' <exp> ';'
```

where <name> is the constant name while <exp> is the expression defining the constant value. The expression is required to evaluate to a number (i.e. an integer or a real).

Action functions. Action functions are composed of an action name and a probability function and can be declared as follows:

```
<actfun> = 'action' <name> '=' <exp> ';'
```

where <name> is the name of the action and <exp> is a probability expression defining the probability function. We recall here that the presence of `frc`(<name>) subexpressions make <exp> define in fact a *function* on the space of occupancy measure vectors (a unit simplex). The guarantee that such a function is continuous [4] and yields values in [0,1] is left to the user.

States constants and Transitions. States and transitions of an individual component can be defined as:

```
<stateconst> = 'state' <name> '{' <trans> ( '+' <trans> )* '}' ';'
```

where <name> is the name of the state and <trans> is a transition defined as follows:

```
<trans> = <name> '.' <name>
```

where the first <name> is the name of a previously declared action and the second <name> the name of a state constant in the model.

Configuration A configuration of the model defines the initial state of the population model. For each local state of an individual component type it specifies the number (as integer) of individuals that are in that local state initially as follows:

```
<config> = 'system' <name> '=' '<' <pop-elem> ( ',' <pop-elem> )* '>'
    ';'
```

where <name> is the name of the system configuration and <pop-elem> is an element of the population vector defined as follows:

```
<pop-elem> = <name> '[' <exp> ']'
```

where <name> is the name of a previously declared state of an individual object and <exp> an integer number denoting the initial number of objects in the local state with name <name>. By default, the first population element represents the selected object under analysis. Recall that, while the user specifies an absolute number of components, FlyFast initially computes the associated fractions, i.e. the initial occupancy measure vector.

PCTL state formulas. The FlyFast specification language may include one or more property specifications expressed as PCTL state-formulas and path-formulas using the syntax described in the following.

PCTL state-formulas have the following syntax:

```
<stateformula> = 'formula' <name> ':' <pctl-exp> ';'
```

where <name> is the name of the formula and <pctl-exp> is defined as:

```
< pctl − exp >  =
          'true'
        | 'false'
        | '!'  < pctl − exp >
        | < pctl − exp >  '|'  < pctl − exp >
        | < pctl − exp >  '&'  < pctl − exp >
        | 'P'  '{'   < rel − sym >  < pBound >  '}'  '['  < path − exp >  ']'
        | < name >
```

where < rel − sym > is a relation symbol denoted by any of the following relational operators '<' | '<=' | '>=' | '>' and < pBound > is a probability bound denoted by a real number between 0 and 1. Furthermore, < path − exp > is a path formula expression defined below.

PCTL path formulas. PCTL path-formulas have the following syntax:

```
< pathformula >  =  'path'  'formula'  < name >  ':'  < pctl − path − exp >  ';'
```

where < name > is the name of the path formula and < pctl − path − exp > is defined as:

```
< pctl − path − exp >  =
          'X'  < pctl − exp >
        | < rel − exp >  'U'  '<='  < int >  < rel − exp >
        | < name >
```

where < int > is an integer bound on the the number of steps and < rel − exp > is a relational expression or a PCTL state formula:

```
< rel − exp >  =
        | < pctl − exp >
        | < simple − exp >  < rel − sym >  < simple − exp >
```

where < simple − exp > is an expression of the form:

```
      < simple − exp >  =
        | < simple − exp >  '+'  < simple − exp >
        | < simple − exp >  '-'  < simple − exp >
        | < simple − exp >  '*'  < simple − exp >
        | < simple − exp >  '/'  < simple − exp >
        | < pop − exp >
        | < int − exp >
        | < real − exp >
        | '('  < simple − exp >  ')'
        | < exp >
```

where < pop − exp > denotes the occupancy measure value of a state in the model at a particular time step (i.e. the fraction of the population that is in that particular local state) and has the following syntax:

```
      < pop − exp >  =
        | 'frc'   '('  < name >  ')'
```

where < name > is the name of a state constant.

Proposition labels. State constants of the model can be labelled by atomic propositions in the following way:

```
< proplab >  =  'label'  < name >  ':'  '{'  < name >  (  ','  < name >)*  '}'  ';'
```

where the first occurrence of < name > is the name of the label (atomic proposition) and the other < name > are the names of state constants. The set of state constant names are the states in which the atomic proposition holds.

4 Examples

In this section we present a simple running example of a FlyFast model and related model checking results that illustrate the modelling language and analysis.

The input language of FlyFast is a simple language for the definition of discrete time population models. In the simplest case, such models are composed of N identical interacting processes or objects, where N is the size of the population. In more elaborate cases the model may consist of more than one population. The language is inspired by the discrete time population models in the work by Le Boudec et al. [4]. At any point in time, each object can be in any of its finitely many states and the evolution of the system proceeds in a clock-synchronous fashion: at each clock tick each member of the population must either execute one of the probabilistic transitions that are enabled in its current state, or remain in the current state. Formal details of the approach can be found in [3].

4.1 Computer Epidemic Model

We illustrate the creation of a model and the use of FlyFast for mean field model checking taking a simple computer epidemic model that describes a network of computers that can be infected by a worm as a running example. Each node in the network can become infected. Such an infection can come from two sources, namely by the activity of a worm of an infected node (inf_sus) in the network, or by an external source (inf_ext). Once a computer is infected, the worm remains latent for a while, and then activates (*activate*).

When the worm is active, it tries to propagate over the network by sending messages to other nodes. After some time, an infected computer can be patched (*patch*), so that the infection is recovered. New versions of the worm can appear; for this reason, recovered computers can become susceptible to infection again after some time (*loss*).

The model of the computer epidemic infection in FlyFast consists of a population of objects, modelling the nodes in the network. Each object has four local states, S, E, I and R, denoting respectively that the node is susceptible to (internal or external) infection, or it has been exposed to infection, it is infected and actively infecting other nodes, or it is recovered (i.e. the computer has been patched.) Fig. 6 shows a graphical model of such an object including the names of the states and transitions.

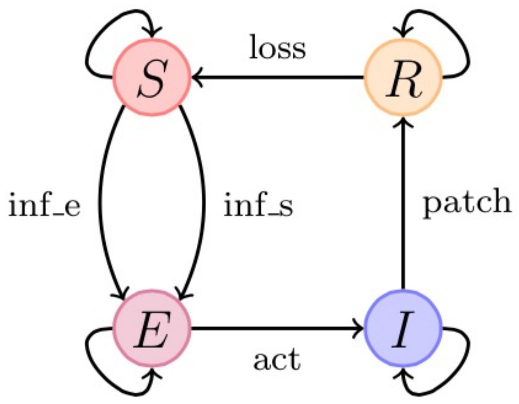

$$\begin{tikzcd}
S \arrow[loop above] \arrow[d, bend right, "inf\_e"'] \arrow[d, bend left, "inf\_s"] & R \arrow[loop above] \arrow[l, "loss"'] \\
E \arrow[loop below] \arrow[r, "act"'] & I \arrow[loop below] \arrow[u, "patch"']
\end{tikzcd}$$

Fig. 6. Model of a node of the network.

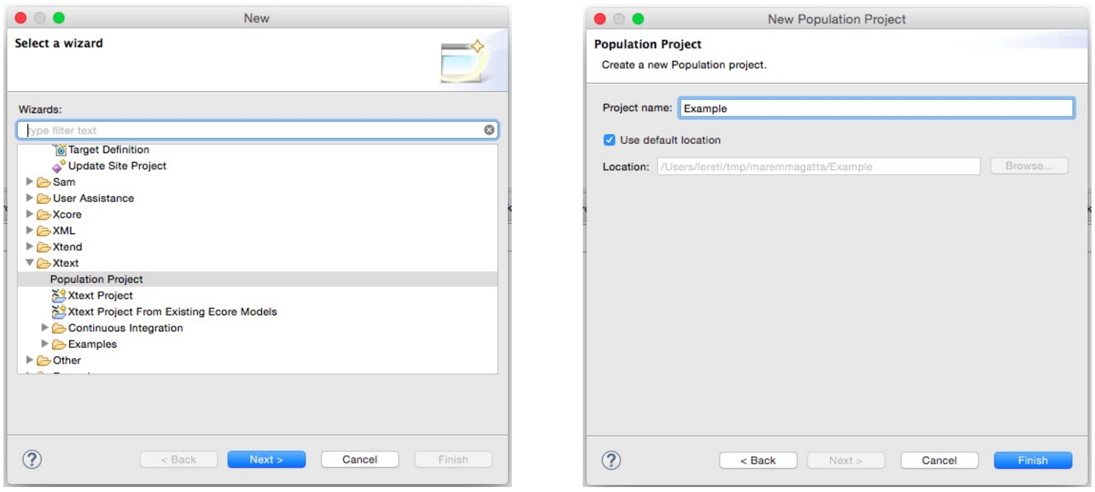
Each transition in Fig. 6 is associated with an action label (except the self-loops). In FlyFast such action labels are associated with action probability functions. These are functions that provide the probability of the transition, which may be a constant, as in Discrete Time Markov Chains (DTMC) or they may depend on the discrete (time) step in which the transition takes place via the (limit) occupancy measure vector. The latter is a vector in which each element of the vector gives the fraction of the population of objects that are in a particular local state. In the case of the epidemic example, the vector consists of four elements. For simplicity of notation in FlyFast the elements of this vector are denoted by the expression $\mathsf{frc}(X)$ where X is the name of a local state. For example, $\mathsf{frc}(I)$ gives the fraction of infected nodes.

A textual version of the same model in the input language of FlyFast follows. We first define the constants of the model, i.e. the population size of one thousand nodes $N = 1000$, and the probability constants of external and internal infection, recovery, activation and loss of resistance. After that, the action probability functions are defined for each action. Probability functions must be continuous functions of the occupancy measure (vector). In these functions the probability constants are used, but in particular it can be observed that the action probability function for internal infection `inf_sus` also depends on the fraction of nodes that are currently infected and activated, namely `frc(I)`. The model specification continues with the definition of states and transitions labelled by actions. Finally, the `system` is defined as a set of N nodes all in initial state `S`. Right after the model some simple formulas are defined. The first one defines the basic formula `isTrue` and the second is a state formula that is true when the selected object for analysis (by default the first of the population defined in the system) is in its local state `I`, i.e. infected and actively infecting other node. The reason for the explicit definition of these simple formulas is that they can now be selected from the user interface for mean field model checking in the composition of more complex formulas. We show an example shortly.

```
const N = 1000;

const alpha_e = 0.1;
const alpha_i = 0.2;
const alpha_r = 0.2;
const alpha_a = 0.4;
const alpha_l = 0.1;

action inf_ext: alpha_e;
action inf_sus: alpha_i * frc(I);
action activate: alpha_a;
action patch: alpha_r;
action loss: alpha_l;

state S {
    inf_ext.E
    + inf_sus.E
}

state E {
    activate.I
}

state I {
    patch.R
}

state R {
    loss.S
}

system mySystem = < S[N] >;

formula isTrue : true ;

formula inI : I ;
```

```
path formula get_infected : true U<=5 inI ;
```

The model can be inserted in FlyFast by starting Eclipse and to create a new *Population project* by going to ***File*** → ***New*** → ***Xtext...*** and selecting ***Population project***. Click *Next* and in the window that appears insert a name for the new project as shown in Fig. 7.



Project name: Example

☑ Use default location

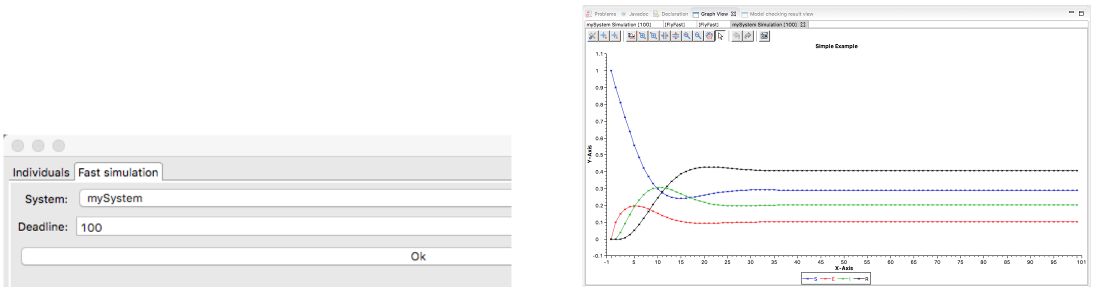
Location: /Users/loreti/tmp/maremmagatta/Example

Fig. 7. Defining a new population project

After that a new project is created that (in this particular case) should be exactly the computer epidemic model (if this is not the case, please correct the model as shown in this document in order to follow the remaining examples). The model is now ready for analysis.

4.2 Analysis with FlyFast

Fast Simulation. The first kind of analysis that can be performed is the transient dynamics of the mean-field approximation of all defined states in the model. This analysis shows how the fraction of the population in each state changes over time. To apply this analysis to the epidemic model, select *SAM* from the main menu, and then *simulation*. From the pop-up menu that will appear, select *fast simulation* and select the system and the end time of the simulation, as shown in Fig. 8 (left). After clicking *ok* the result is generated. These results can be saved numerically by responding 'yes' to the pop-up window when the results are ready. Both when selecting 'yes' and when selecting 'no' the result also appears in the graph window as shown in Fig. 8 (right).

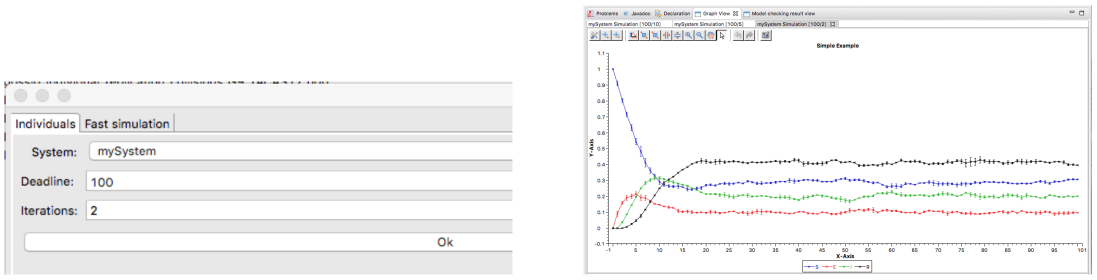


System: mySystem

Deadline: 100

Fig. 8. Fast simulation

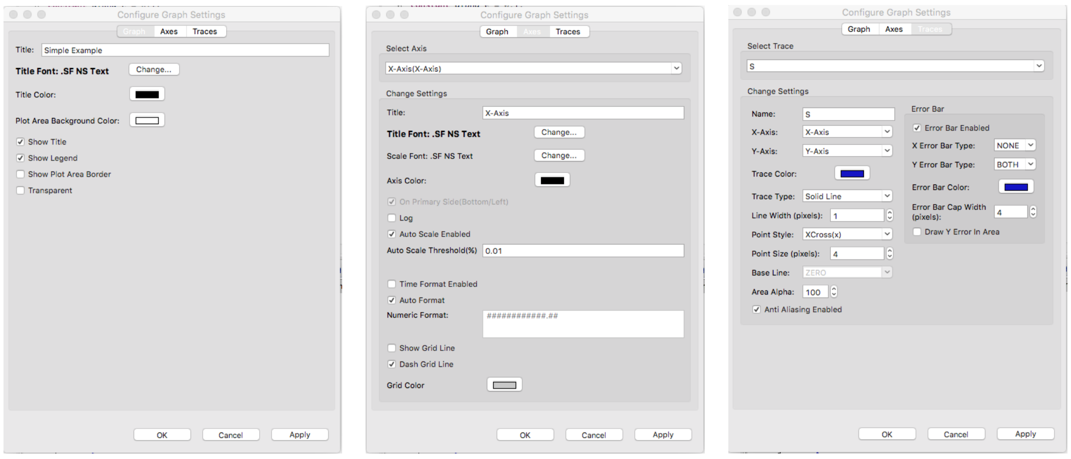
Individual Simulation. Individual simulation can be performed by selecting *individuals* from the simulation menu (see Fig. 9 (left)). This provides a similar pop-up menu as for *fast simulation*, but now one is furthermore asked to specify the desired number of simulation runs. Individual simulation provides a discrete event simulation of the probabilistic population model in which each element of the population is treated as an individual. For large populations this analysis will obviously require much more time than *fast simulation*. It may however provide an empirical method to obtain a first impression of how well the mean field analysis approximates the actual full simulation. Indeed, the traces of individual simulation shown in Fig. 9 (right) show a certain correspondence with those for fast simulation in Fig. 8 (right), despite the average of only two individual simulation traces are considered in this particular case.



System: mySystem
Deadline: 100
Iterations: 2

Fig. 9. Individual simulation, average of two runs showing variance as small vertical bars

The graph settings can be further inspected and configured, for example providing a name and selecting different colours for the curves, by the radio buttons on the upper-left side of the graph in a standard way. For example, clicking the screwdriver-and-pincers button on the far left the graph, the axes and the traces can be configured (see Fig. 10).



Title: Simple Example
Title Font: .SF NS Text
Title Color: ■
Plot Area Background Color: □
✓ Show Title
✓ Show Legend
☐ Show Plot Area Border
☐ Transparent

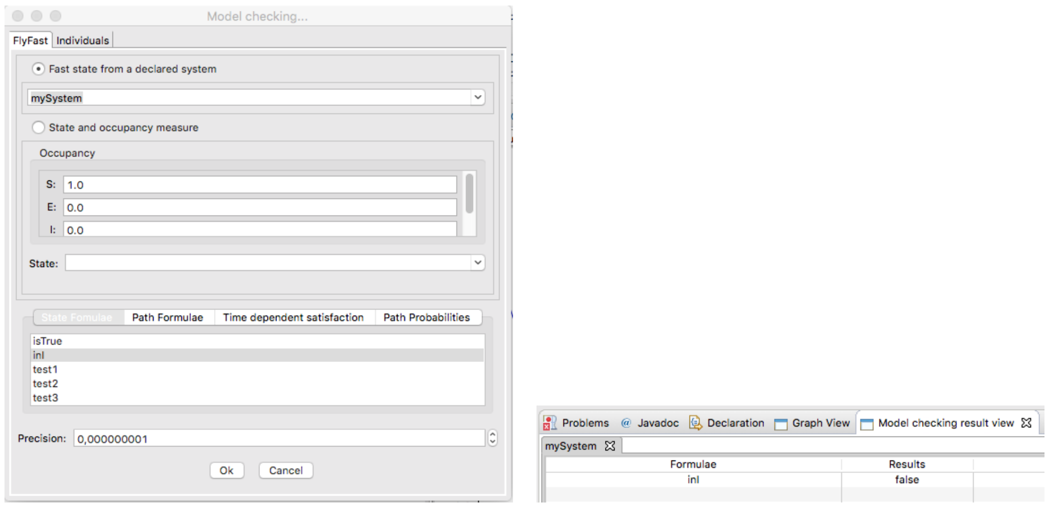
Select Axis: X-Axis(X-Axis)
Change Settings
Title: X-Axis
Title Font: .SF NS Text
Scale Font: .SF NS Text
Axis Color: ■
✓ On Primary Side(Bottom/Left)
☐ Log
✓ Auto Scale Enabled
Auto Scale Threshold(%) 0.01
☐ Time Format Enabled
✓ Auto Format
Numeric Format: ############.##
☐ Show Grid Line
✓ Dash Grid Line
Grid Color

Select Trace: S
Change Settings
Name: S
X-Axis: X-Axis
Y-Axis: Y-Axis
Trace Color: 🟦
Trace Type: Solid Line
Line Width (pixels): 1
Point Style: XCross(x)
Point Size (pixels): 4
Base Line: ZERO
Area Alpha: 100
✓ Anti Aliasing Enabled
Error Bar
✓ Error Bar Enabled
X Error Bar Type: NONE
Y Error Bar Type: BOTH
Error Bar Color: 🟦
Error Bar Cap Width (pixels): 4
☐ Draw Y Error In Area

Fig. 10. Configuration of graph settings

The graph can also be saved as a png file to be used in other documents by clicking on the button showing a foto camera. Other buttons are self-explanatory.

Mean Field Model Checking: State Formula. Selecting *model checking* from the menu under *SAM* in the main menu bar a model checking window pops up. In the top of this window, shown in Fig. 11, one can choose between *flyfast* and *individuals*. The former provides on-the-fly mean field model checking, whereas the latter provides standard on-the-fly probabilistic model checking. In the case of mean field model checking the property is checked for the *first object* specified in the system configuration and evaluated in the context of the overall system. When mean field model checking is selected in the pop-up window, the user is asked to select the system configuration of interest, in this case mySystem and, optionally, the user can set the initial occupancy measure values and the initial state of the object to be verified. In the next panel of the window one of four types of mean field verification can be selected, the first of which is the verification of a *state formula*. Selecting *state formula* one of the state formulas defined in the FlyFast specification of the computer epidemic can be selected. In this case we select `inI`, which is a simple state formula that checks whether the first object is in state `I` (i.e. *infected*). After clicking *ok* the result is shown in the *model checking result view* as shown in Fig. 11 (right), selecting the *model checking result view*. It shows both the formula and the model checking result (false in this case, since the first object is initially in state `S`).



[● Fast state from a declared system]
mySystem
○ State and occupancy measure
Occupancy
S: 1.0
E: 0.0
I: 0.0
State:

isTrue
inI
test1
test2
test3

Precision: 0,000000001

mySystem

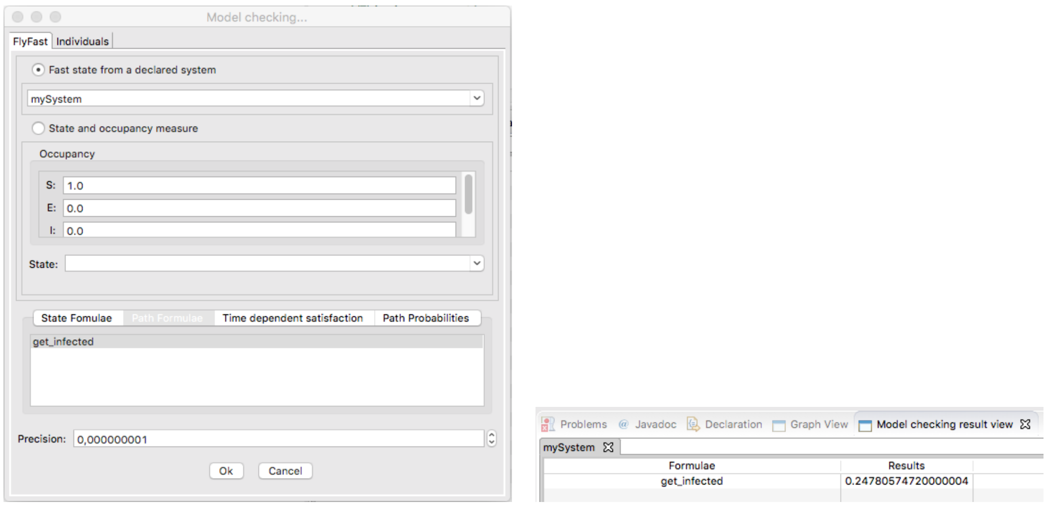
Formulae	Results
inI	false

Fig. 11. Mean field model checking a state formula: window (left) and result (right)

Mean Field Model Checking: Path formula. As is the case for the verification of state formulas, also for the verification of path formulas one needs to select *model checking* from the menu under *SAM* in the main menu bar after which a model checking window pops up. In the top of this window, shown in Fig. 12, one needs to choose *flyfast* and then select the system configuration (and optionally the initial occupancy measure and initial state). After this, the option *path formulas* can be selected for the verification of a path formula. This is done by selecting a path formula from the list, which are the path formulas that have been defined in the FlyFast model of the system. Clicking *ok* runs the model checker on the selected formula and configuration. The result is shown in Fig. 12 (right), selecting the *model checking result view*. It shows both the name of the formula and the model checking result (showing a probability of 0.247 in this case). In fact, the property `get_infected` specifies the probability of the selected individual object, initially in state `S`, to get infected within 5 steps, given the initial state of the rest of the population specified in the system configuration `mySystem`.

Changing the optional values of the initial occupancy measure in the window shown in Fig. 12 (left) using the same formula the path probability can be verified for different initial states of the global system and the individual object of interest.

Mean Field Model Checking: Time Dependent Satisfaction. The third mean field model checking option provides time dependent model checking of the satisfaction of a path formula. As an example we


Model checking...

FlyFast | Individuals

(•) Fast state from a declared system

mySystem

( ) State and occupancy measure

Occupancy

S: 1.0
E: 0.0
I: 0.0

State:

State Fomulae | Path Formulae | Time dependent satisfaction | Path Probabilities

get_infected

Precision: 0,000000001

Ok | Cancel

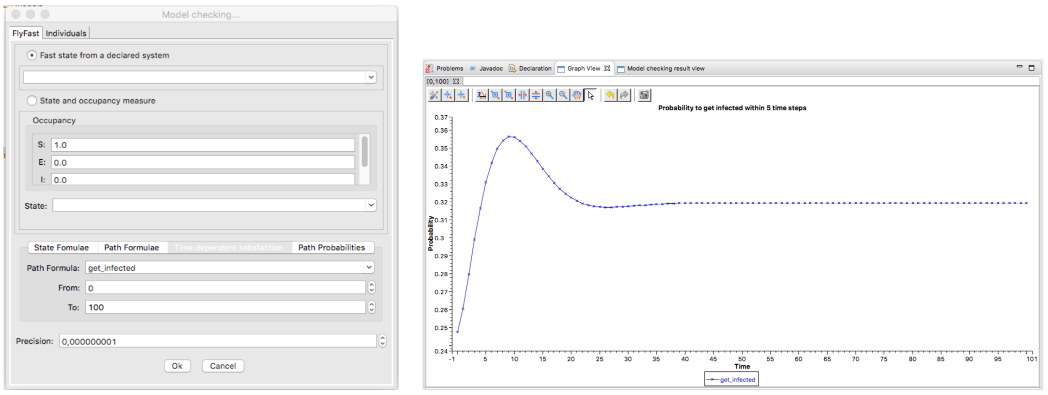
Problems | Javadoc | Declaration | Graph View | Model checking result view

mySystem

Formulae	Results
get_infected	0.24780574720000004

Fig. 12. Mean field model checking a path formula: window (left) and result (right)

take the path formula `get_infected` defined in the FlyFast specification file. Selecting the third model checking option, as shown in Fig. 13 (left), we need to provide the name of a predefined path formula, a start-time and an end-time. The model checking procedure evaluates the path formula in every step of the defined time interval and shows the result in a graph, see Fig. 13 (right). The model checking procedure uses memoization to avoid the repeated re-calculation of earlier results in order to increase efficiency. The graph in Fig. 13 (right) shows the result.


Model checking...

FlyFast | Individuals

(•) Fast state from a declared system

( ) State and occupancy measure

Occupancy

S: 1.0
E: 0.0
I: 0.0

State:

State Fomulae | Path Formulae | Time dependent satisfaction | Path Probabilities

Path Formula: get_infected
From: 0
To: 100

Precision: 0,000000001

Ok | Cancel

Problems
Javadoc
Declaration
Graph View
Model checking result view
[0,100]
Probability to get infected within 5 time steps
Probability
Time
get_infected

Fig. 13. Mean field model checking a path formula at a series of time steps: window (left) and result (right)

Note that the name of the graph and the names of the axes in the graph have been set manually to reflect their meaning as explained previously (see Fig. 10). In this case, the x-axes represents the time steps. The y-axes represents the corresponding probability of the set of paths, satisfied by the path formula at different points in time (and thus with different, time dependent, occupancy measure values reflecting the average state of the overall system at the given time step.) The finalised and configured graph can be exported as a png file by clicking on the button showing the icon of a foto-camera. The result is shown in Fig. 14.

Mean Field Model Checking: Path Probabilities. The fourth and rightmost option provides mean field model checking for a family of path formulas of the form `state_formula_1 U<=t state_formula_2` in-

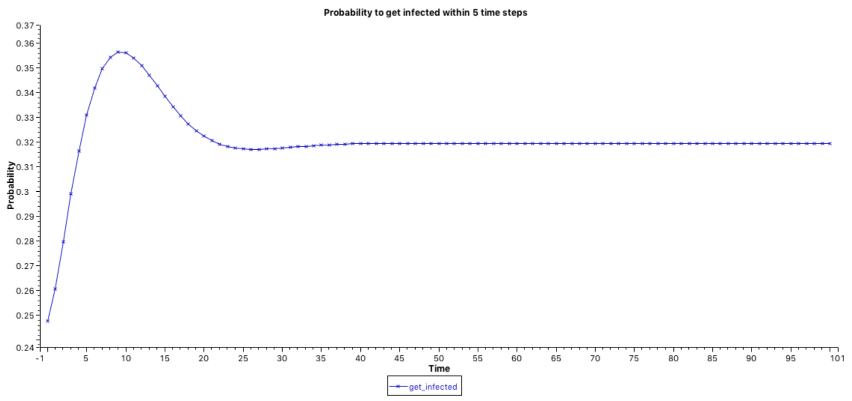
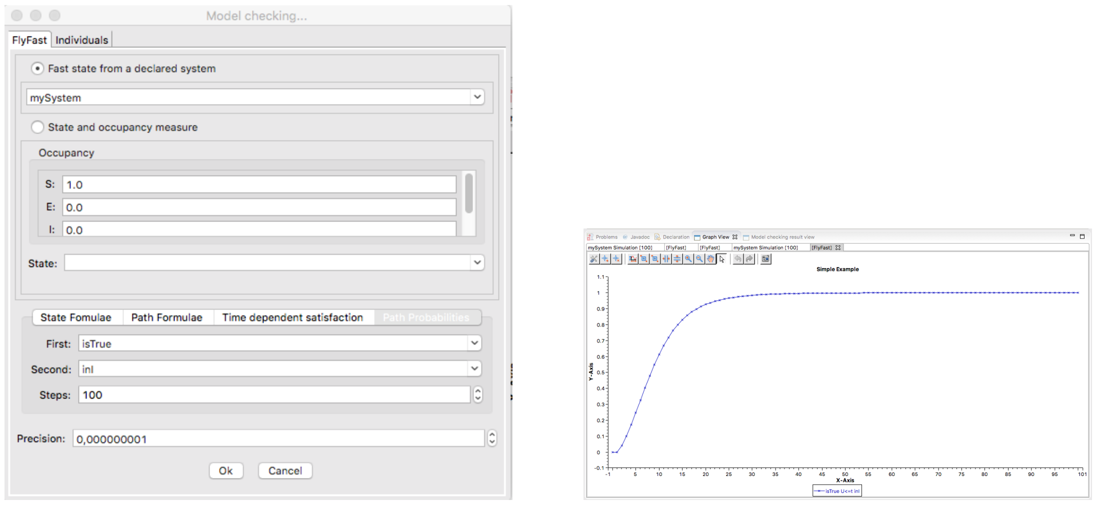


Fig. 14. Exported result graph of Fig. 13 (right) in png format

dexed by the time bound `t`. The first and the second state formula can be selected via the interface window (see Fig. 15 (left)) after *Path Probabilities* has been selected as verification option. Also the range for the value of `t` can be selected as the number of steps to be considered, starting from step zero. In this particular example we use this type of verification to investigate how the probability changes, as a function of the value of bound `t`, that the selected individual object changes its state from susceptible (its initial state) to infected within `t` steps. This is done by selecting `isTrue` as the first state formula and `inI` as the second state formula. The chosen range for `t` is from 0 to 100.

The result is shown in the graph in Fig. 15 (right). In fact, for each value of t mean field model checking is performed to produce the probability.


Model checking...

FlyFast | Individuals

(•) Fast state from a declared system

mySystem

( ) State and occupancy measure

Occupancy

S: 1.0
E: 0.0
I: 0.0

State:

State Fomulae | Path Formulae | Time dependent satisfaction | Path Probabilities

First: isTrue
Second: inI
Steps: 100

Precision: 0,000000001

Ok | Cancel

Fig. 15. Probability the selected individual gets infected within t steps when initially susceptible.

5 Final Remarks

The FlyFast model specification may contain multiple instances of a system configuration and multiple state and path formulas may be defined. Moreover, more complex specifications may contain more than one type of population, see [2] for an example of a client-server model with a population of clients and a population of servers.

References

1. Latella, D., Loreti, M., Massink, M.: On-the-fly fast mean-field model-checking. In: Abadi, M., Lluch-Lafuente, A. (eds.) Trustworthy Global Computing - 8th International Symposium, TGC 2013. LNCS, vol. 8358, pp. 297–314. Springer (2013)
2. Latella, D., Loreti, M., Massink, M.: On-the-fly fluid model checking via discrete time population models. In: Beltrán, M., Knottenbelt, W.J., Bradley, J.T. (eds.) Computer Performance Engineering - 12th European Workshop, EPEW 2015. LNCS, vol. 9272, pp. 193–207. Springer (2015)
3. Latella, D., Loreti, M., Massink, M.: On-the-fly PCTL fast mean-field approximated model-checking for self-organising coordination. Sci. Comput. Program. 110, 23–50 (2015)
4. Le Boudec, J., McDonald, D.D., Mundinger, J.: A generic mean field convergence result for systems of interacting objects. In: Fourth International Conference on the Quantitative Evaluaiton of Systems (QEST 2007). pp. 3–18. IEEE Computer Society (2007)