QUANTICOL

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

quantiCOL

http://www.quanticol.eu

D5.3

The QUANTICOL software tool suite for modelling smart cities (Final)

Revision: 1.0; March 30, 2017

Author(s): Vincenzo Ciancia (ISTI-CNR), Diego Latella (ISTI-CNR), Mieke Massink (ISTI-CNR), Laura Nenzi (IMT), Mirco Tribastone (IMT), and Andrea Vandin (IMT)

Due date of deliverable: Month 48 (March 2017)
Actual submission date: March 30, 2017
Nature: R. Dissemination level: PU

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

Executive Summary

This deliverable is an account of the QUANTICOL software tool suite, with application to the smart city scenarios studied in the project. While Deliverable D4.3 focuses on the CARMA language and its implementation with the CARMA Eclipse plug-in, in this deliverable we present software tools developed to support further techniques such as model checking, model order reduction, and reachability analysis. Their integration with the CARMA Eclipse plug-in is discussed in Deliverable D4.3. Instead, here we provide detailed account of such tools as stand-alone products. We present *FlyFast*, a probabilistic model checker for mean-field models; *jSSTL*, for the analysis of Signal Spatio-Temporal Logic; *ERODE*, for the reduction of systems of ordinary differential equations; *UTOPIC*, for the reachability analysis of nonlinear dynamical systems; and *topochecker*, a spatio-temporal model checker.

While an application of *FlyFast* to smart cities is presented in D4.3, in this deliverable we discuss an application of *jSSTL* and *ERODE* to the verification and reduction, respectively, of a model of a bike-sharing system. Instead, bus transportation systems are studied with *topochecker*, to detect problems in vehicle location data and clumping in frequent services.

Contents

1 Introduction

Whereas Deliverable D4.3 focuses describes the CARMA Eclipse plug-in, this deliverable reports on the QUANTICOL Software Tool Suite. This represents a collection of other tools developed during the project in order to make our scientific results applicable to a range of case studies of collective adaptive systems, and beyond. In accordance with the Description of Work, this deliverable is concerned with presenting the tools and showing how they can be applied for the analysis of the smart city case studies that have been considered throughout the entire duration of the project. Specifically, we present:

- *FlyFast*, a first-of-its-kind, *on-the-fly mean-field* probabilistic model checker for bounded PCTL (Probabilistic Computation Tree Logic) properties of a *selected individual* in the context of systems that consist of a *large number* of independent, *interacting objects*. The underlying on-the-fly mean field model checking algorithm was developed and proven correct in [37, 38] (in the context of WP3). *FlyFast* is provided within the jSAM (java StochAstic Model Checker) framework which is an open source Eclipse plug-in[1] integrating a set of tools for stochastic analysis of concurrent and distributed systems specified using process algebras.

- *jSSTL*, a Java tool for the specification and the verification of Signal Spatio-Temporal Logic (SSTL) properties [43, 44]. It consists of a library (the *jSSTL* API) and a front-end developed as an Eclipse plug-in. The plug-in provides a user friendly interface to the tool, whereas the library can be used to integrate *jSSTL* within other applications and tools, as it has been done in [4].

- *ERODE*, a tool for the evaluation and reduction of ordinary differential equations implemented as an Eclipse plug-in [16]. It supports the results of [12, 15, 13, 14] (in the context of WP3).

- *UTOPIC*, which supports an under-approximation technique for the reachability analysis of nonlinear systems of ordinary differential equations (ODEs). It implements an algorithm based on control-theoretic principles of optimal control, originally developed in [10] (in the context of WP1). *UTOPIC* itself is implemented as an extension of *ERODE*.

- *topochecker*, a spatio-temporal model checker based on closure spaces and Kripke frames. Currently it checks a spatial extension of Computation Tree Logic named STLCS (Spatio-Temporal Logic for Closure Spaces). The spatial fragment was originally presented in [20], and then extended in [21]; the spatio-temporal logic was introduced in [19]. The underlying theory has been developed in the context of WP3.

This deliverable is closely related to Deliverable D4.3, which focusses on the CARMA language and its implementation in the CARMA Eclipse plug-in. The integration possibilities for some of the tools (e.g. *jSSTL* and *FlyFast*) are explained in more detail in Deliverable D4.3. Here we describe how the tools operate as stand-alone products.

2 *FlyFast*

The on-the-fly mean-field model-checker *FlyFast* has been briefly introduced in Deliverable 5.2, whereas an outline of its underlying theoretical framework can be found in Deliverable 3.1. In this section, we illustrate the various analysis options offered by the tool in more detail [39]. We do this using the gossip shuffle protocol, as presented in [2, 3], as a running example. We analyse the replication and coverage for a generic data item d (called "d-item" in the following) in a *fully connected network* in which the nodes execute the shuffling protocol. We consider the discrete time variant of this protocol with a maximal delay between two subsequent gossips of a node denoted by G_{max}. Following the mean field approximation technique, the behaviour of an individual node is based on its local state and the current occupancy measure vector. Figure 1 shows the states

[1] http://quanticol.github.io/jSAM/

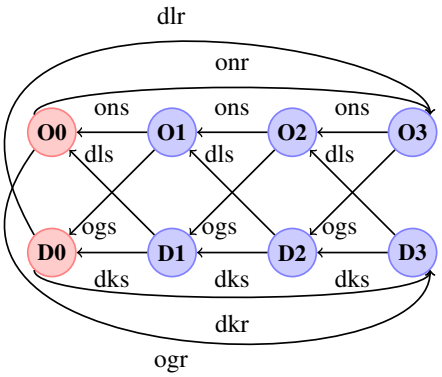


Figure 1: Push-pull gossip model of individual gossip node with rounds of length 3 (i.e. $g_{max} = 3$). Active states are red, passive ones blue.

const N = 2500	**const** P_01_10 = $\frac{s}{c} * \frac{n-c}{n-s}$	**const** P_11_10 = P_11_01
const n = 500	**const** P_10_01 = P_01_10	**const** P_01_11 = $\frac{s}{c} * \frac{c-s}{c} * \frac{n-c}{n-s}$
const c = 100	**const** P_01_01 = $\frac{c-s}{c}$	**const** P_10_11 = P_01_11
const s = 50	**const** P_10_10 = P_01_01	**const** P_11_11 = $1 - 2 * \frac{s}{c} * \frac{c-s}{c} * \frac{n-c}{n-s}$
	const P_11_01 = $\frac{s}{c} * \frac{c-s}{n-s}$	**const** P_00_00 = 1

Figure 2: Constants of the *FlyFast* Gossip model.

and transitions of a single node where $G_{max} = 3$. The red states, $D0$ and $O0$, denote states in which the gossip node is active, i.e. it can initiate an exchange of local information with a passive node; in $D0$ (resp. $O0$) the node has (resp. does not have) the data element in its local store. The blue states denote states in which the node is passive and can be contacted by an active node. The number in the node-labels denotes the value of the current gossip delay g, ranging from 0 to 3. The D/O convention with respect to having the data element applies also to the passive nodes.

The modelling language of *FlyFast* consists of basic constructs to describe the probabilistic behaviour of an individual object, such as constants, states, action probabilities and transitions. The constants in the gossip model are the total number of nodes N, the number of different data elements in the system n, the size of the cache c and the number of data elements exchanged between two shuffling nodes s. Their definition is shown in Figure 2. Furthermore, the action probabilities make use of a number of conditional probabilities, expressed in terms of the constants n, c and s. For example, P_01_10 stands for P(01|10) [2, 3] and denotes the conditional probability that after a shuffle the active node loses the data element, whereas the passive node acquires it (the '01' part of P_01_10) given that before the shuffle the active node had the data element and the passive one did not (the '10' part of P_01_10). Action probabilities are defined as shown in Fig 3. The action labels are those of Figure 1. For example, the action dlr ('has **d**, **l**oses it and **r**esets gossip delay') labels the transition from the active state in which the object has the d-element ($D0$) to the passive state without d in which the clock is reset to G_{max}, i.e. $O3$ in this case. The probability of action dlr depends on the occupancy measure via the quantities frc(Xi), with $X \in \{O, D\}$ and $i \in \{0, \ldots, 3\}$, which denote the *fraction* of objects that are in state Xi. The expression $e^{-2*(\text{frc}(O0)+\text{frc}(D0))}$ denotes the probability that no 'collision' occurs in the communication between two nodes, such as two active nodes or two passive nodes that contact each other. Finally, Figure 4 shows the definition of the states and transitions of a single node as in Figure 1, and the non-empty elements of the initial occupancy measure vector mainO0. By default, the first element of the vector is the object selected for *FlyFast* analysis. Elements with initially zero occurrences are omitted. We refer to [2] for further details of the gossip model.

action dlr : // has d, looses it, resets delay
$(\mathrm{frc}(O1)+\mathrm{frc}(O2)+\mathrm{frc}(O3))*\mathrm{P\_01\_10}*e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))}+$
$(\mathrm{frc}(D1)+\mathrm{frc}(D2)+\mathrm{frc}(D3))*\mathrm{P\_01\_11}*e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))}$
action dls : // has d, looses it, one time step passes
$\mathrm{frc}(O0)*\mathrm{P\_10\_01}*e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))}+\ \mathrm{frc}(D0)*\mathrm{P\_10\_11}*e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))}$
action dkr : // has d, keeps it, resets delay
$(\mathrm{frc}(O1)+\mathrm{frc}(O2)+\mathrm{frc}(O3)+\mathrm{frc}(D1)+\mathrm{frc}(D2)+\mathrm{frc}(D3))*(1-e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))})+$
$(\mathrm{frc}(O0)+\mathrm{frc}(D0))+$
$(\mathrm{frc}(O1)+\mathrm{frc}(O2)+\mathrm{frc}(O3))*(\mathrm{P\_10\_10}+\mathrm{P\_11\_10})*e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))}+$
$(\mathrm{frc}(D1)+\mathrm{frc}(D2)+\mathrm{frc}(D3))*(\mathrm{P\_10\_11}+\mathrm{P\_11\_11})*e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))}$
action dks : // has d, keeps it, one time step passes
$(\mathrm{frc}(O0)+\mathrm{frc}(D0))*(1-e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))})+\ (1-(\mathrm{frc}(O0)+\mathrm{frc}(D0)))+$
$\mathrm{frc}(O0)*(\mathrm{P\_01\_01}+\mathrm{P\_11\_01})*e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))}+$
$\mathrm{frc}(D0)*(\mathrm{P\_01\_11}+\mathrm{P\_11\_11})*e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))}$
action ogr : // no d, gets it, resets delay
$(\mathrm{frc}(D1)+\mathrm{frc}(D2)+\mathrm{frc}(D3))*(\mathrm{P\_10\_01}+\mathrm{P\_11\_01})*e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))}$
action ons : // no d, no get, one time step passes
$(\mathrm{frc}(O0)+\mathrm{frc}(D0))*(1-e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))})+\ (1-(\mathrm{frc}(O0)+\mathrm{frc}(D0)))+$
$\mathrm{frc}(O0)*\mathrm{P\_00\_00}*e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))}+\ \mathrm{frc}(D0)*\mathrm{P\_10\_10}*e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))}$
action onr : // no d, no get, resets delay
$(\mathrm{frc}(O1)+\mathrm{frc}(O2)+\mathrm{frc}(O3)+\mathrm{frc}(D1)+\mathrm{frc}(D2)+\mathrm{frc}(D3))*(1-e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))})+$
$(\mathrm{frc}(O0)+\mathrm{frc}(D0))+$
$(\mathrm{frc}(O1)+\mathrm{frc}(O2)+\mathrm{frc}(O3))*\mathrm{P\_00\_00}*e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))}+$
$(\mathrm{frc}(D1)+\mathrm{frc}(D2)+\mathrm{frc}(D3))*\mathrm{P\_01\_01}*e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))}$
action ogs : // no d, gets it, one time step passes
$\mathrm{frc}(D0)*(\mathrm{P\_01\_10}+\mathrm{P\_11\_10})*e^{-2*(\mathrm{frc}(O0)+\mathrm{frc}(D0))}$

Figure 3: Actions and their probability functions in the *FlyFast* Gossip model. Comments in blue.

2.1 Interface Overview

Figure 5 provides a screenshot with an overview of the Eclipse-based user interface for *FlyFast*. Three main panels are shown. The left panel shows the repository structure, with the current population models in directory 'model'. The panel at the top shows part of the specification of the current model. *FlyFast* uses different colours to distinguish language keywords from comments and user definitions. The panel at the bottom shows analysis results in graphical form. Graphs can be easily customised using the radio buttons in the upper-left corner of this panel. These customisations allow for an interactive inspection of details of the graph, such as vertical and horizontal zoom, addition of annotations, insertion of labels for the axes and title of the graph, use of colours, selection of particular trajectories, manual and automatic rescaling for both axes, including the selection of log-scale. Furthermore, the results can be exported in numerical form for customised processing by external tools such as Octave, Matlab or gnuplot.

state D0 {dlr.O3 + dkr.D3} **state** O0 {ogr.D3 + onr.O3}
state D1 {dls.O0 + dks.D0} **state** O1 {ogs.D0 + ons.O0}
state D2 {dls.O1 + dks.D1} **state** O2 {ogs.D1 + ons.O1}
state D3 {dls.O2 + dks.D2} **state** O3 {ogs.D2 + ons.O2}

system mainO0 = <O0[(N/4)-1],O1[N/4], ..., O3[N/4],D0[1]>

Figure 4: States and initial configuration of the *FlyFast* Gossip model.

The bottom panel of Figure 5 shows the evolution of the values of the deterministic approximation of the occupancy measure vector, i.e. the fraction of nodes that are in a particular local state at any time—note that in this gossip model the values of all *D*-states nearly overlap, and the same holds for the *O*-states. This analysis provides a fast way to approximate the true occupancy measure values. In the case of the gossip model the occupancy measure of the *D*-states show the replication of the d-item throughout the network. Computing the true occupancy measure values via full stochastic simulation of a model with N objects may be very costly when N is large. Mean field approximation for fast simulation, used also by *FlyFast*, instead, is independent of the number of objects in the model and provides good approximations as long as N is large enough [40]. However, *FlyFast* also provides full stochastic simulation that may be used to compare results. For example, Figure 6 (top) shows the mean of 10 simulation traces, whereas Figure 6 (bottom) shows a close-up of part of the upper traces concerning *O*-states with their variance indicated by vertical bars.

2.2 Available Analysis Techniques

The various analysis techniques can be selected from a pop-up menu. In that menu also further options for each technique can be provided, such as the length of the trajectories and, in case of full simulation, the number of replications. Figure 7 (right) shows the menu options for fast (i.e. mean field based) probabilistic model checking of population models. In the top-panel the initial state of a system is selected that has been included in the textual specification of the model. It provides both the initial state of the individual component selected for model checking and the initial values of the state counters, from which the initial occupancy measure vector is computed. By default, the first element of the counters vector is the selected individual component. Several different initial states may be defined in the model file indicated by the keyword 'system'. The initial occupancy measure vector and the state of the individual of interest can also be set directly from the menu using the second

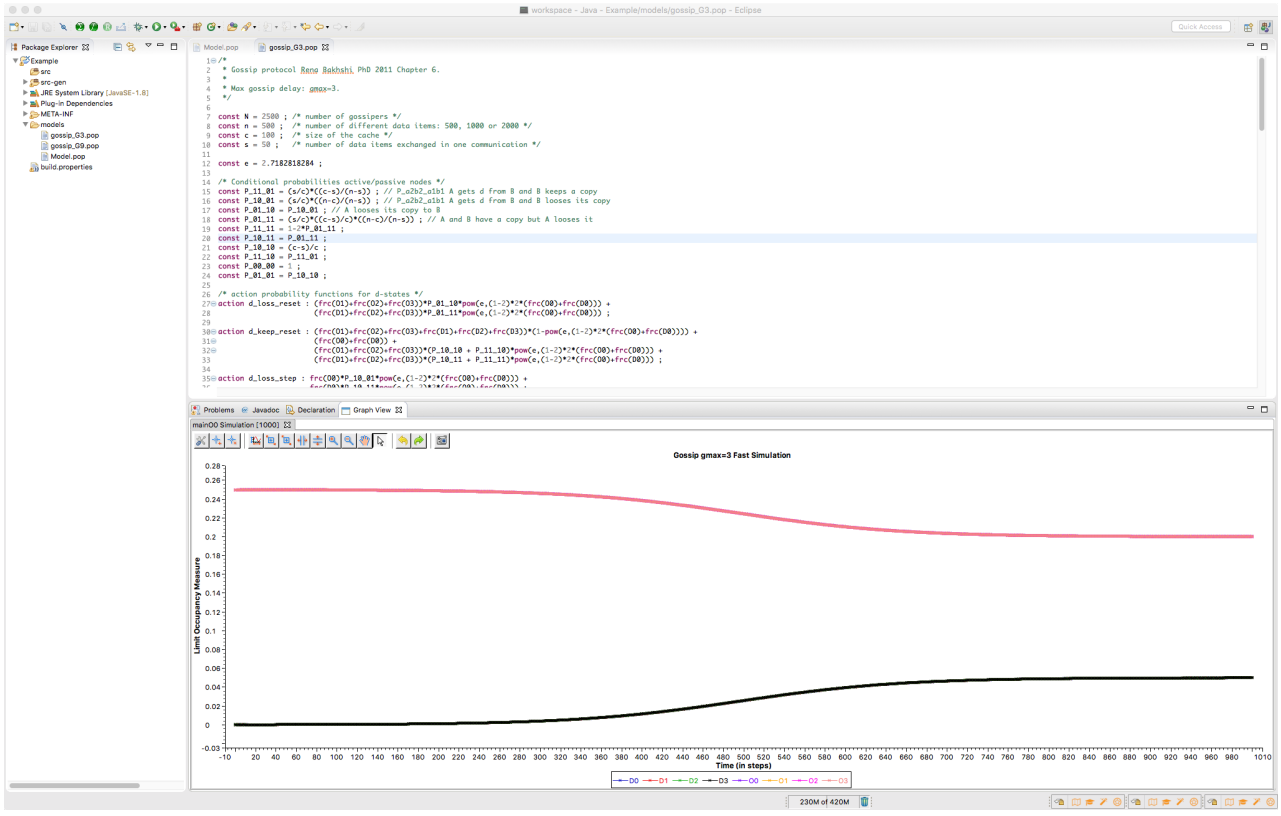


Figure 5: Overview of the *FlyFast* Eclipse user interface and replication of the data element.

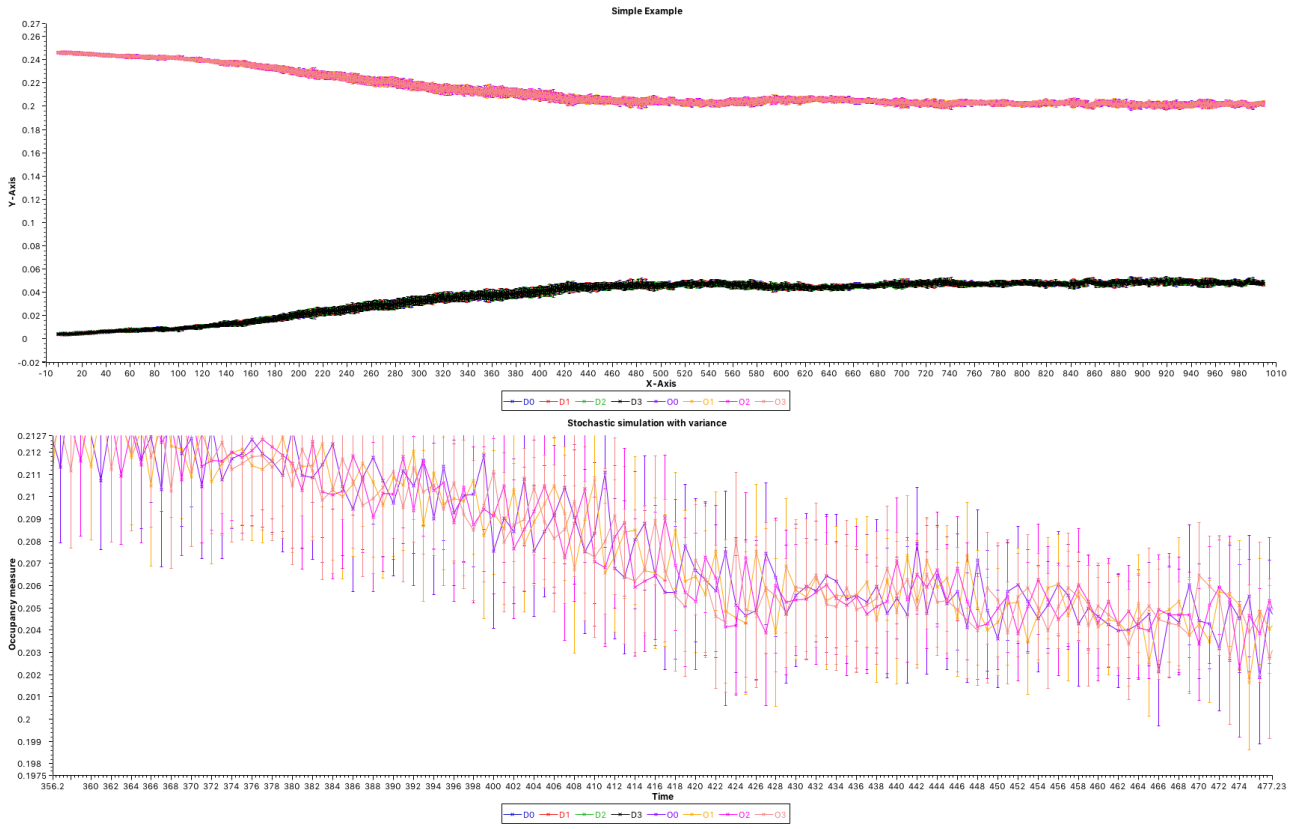


Figure 6: *FlyFast* stochastic simulation (top) and close-up (bottom) of data item replication.

panel.

The third panel (still in Fig. 7 (right)) shows the different fast probabilistic model-checking options. Properties always refer to the selected first object and are evaluated in the context of the large population of which the selected object is a part. The first two options, '*State Formulae*' and '*Path Formulae*', are used to verify a single PCTL formula of the selected kind, providing either a yes/no answer in the former case and a path probability in the latter case. The two right-most options, i.e. '*Time dependent satisfaction*' and '*Path Probabilities*' provide more complex analysis (the latter is shaded in Figure 7 (right) because it is currently being selected and therefore less visible.) In population models the probability of an individual to perform a particular transition may depend on the state (approximated by the occupancy measure in this case) of the rest of the system. This means that the probability value of a path formula may depend on the time at which it is evaluated. The evolution over time of this probability value can be analysed with the option '*Time dependent satisfaction*'. It requires the selection of a path formula, defined in the model file, a starting time and an end time. The result is presented graphically for the selected time interval.

The right-most option '*Path Probabilities*' is the most complex one. It requires the specification of an until path formula that is parametric in the time-bound t and an interval for the time-bound. It computes the relevant probability when t varies in the given interval. A plot is produced in the bottom panel. There is no restriction on the state-formulas which appear in the specified until-formula, except that these state formulas have to be selected from the menu and are of course among those the user defined in the model specification file.

Figure 7 (left) shows the probability, for a gossip model extended in the obvious way to one in which $G_{max} = 9$, that the selected node 'has seen' the data element within time $t \in \{0, \ldots, 3000\}$:

$$\mathsf{isTrue}\ \mathsf{U} \leq t\ \mathsf{hasD} \quad \textit{where} \quad \mathsf{hasD} = \mathsf{inD0} \mid \ldots \mid \mathsf{inD9}$$

Since all nodes have the same probabilistic behaviour, this probability corresponds to the fraction of the network that has seen the data-element within time t (i.e. the coverage and convergence). This parametric analysis

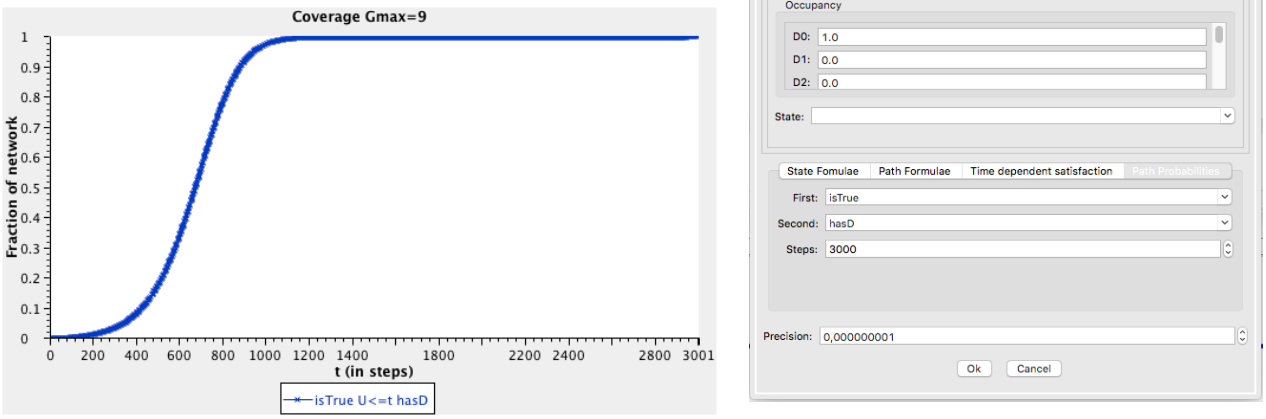


Figure 7: Network coverage for a model with $G_{max} = 9$, $N = 2500$, $n = 500$, $s = 100$ and $c = 50$.

required 16,997 ms. on an iMAC, 2,66GHz ICi5, with 8GB memory (same for *any* population size $N \geq 2500$). The results in [2] show close correspondence to those obtained[2] with *FlyFast* for an initial state defined as system main in Figure 4 but for $G_{max} = 9$.

3 *jSSTL*

In this section, we introduce *jSSTL* [45], a Java tool for the specification of *Signal Spatio-Temporal Logic* (SSTL) properties and their robust monitoring on spatio-temporal trajectories. SSTL is a linear-time temporal logic, recently designed [43, 44] , suitable for describing behaviours of spatio-temporal traces generated from simulations or measured from real systems. It spatially extends *Signal Temporal Logic* (STL [41, 28]) with a number of spatial modalities that permit us to specify metric and topological properties in discrete space. In [43, 44], the authors provide a Boolean and quantitative semantics of SSTL, efficient monitoring algorithms for both semantics and case studies to show the usefulness of the logic.

In order to make the logic usable in practice, an efficient and usable implementation has been developed, *jSSTL*. The tool consists of a *jSSTL* API and a front-end, integrated in Eclipse. The API, available at `https://bitbucket.org/LauraNenzi/jsstl`, provides the backend classes and can be used to integrate *jSSTL* within other applications and tools, as it has been done in [4]. The Eclipse plug-in, available at `http://quanticol.sourceforge.net/?page_id=90`, provides a user friendly interface to the tool. Below, we describe the framework and the use of the plug-in through a running example of a cholera outbreak, a prominent case study of a waterborne disease [7, 42]. We will use it here to introduce the type of models that we can consider, to give some examples of SSTL properties and to show the tool at work.

3.1 Running example

Model. As mentioned above, as a running example we consider a model of Cholera spread among communities that live close to a river. The space is represented as a weighted graph, see Figure 8. The nodes ℓ_1, ..., ℓ_7 represent the communities; the edges describe the connection between the communities through the river. The graph is equipped with a weight function $w : E \to \mathbb{R}$ that returns the cost of each edge; for this model we interpret the cost $w(\ell_i, \ell_j)$ of an edge as the distance between the locations ℓ_i and ℓ_j.

[2]Note that there is no need to extend the model with additional states that represent the fact that a node 'has seen' the data element, as was the case in [2].

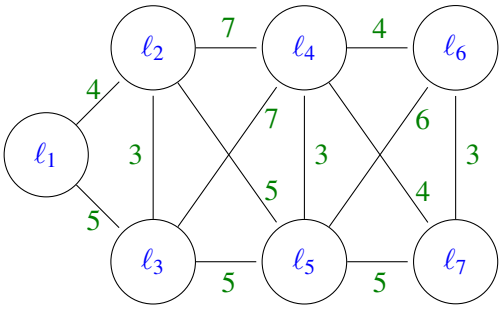


Figure 8: Representation of the space of the epidemic model. The nodes ℓ_1, ..., ℓ_7 represent the different communities; the edges represent the link between the communities through the water basin. The green numbers correspond to the values w of the weight function for each edge.

There are two agent classes: the bacteria and the individuals. The bacteria have only one state (B) but they can be transported to different nodes via the river. An individual can be in three different states: susceptible (S) infected (I) and recovered (R), but cannot change location. Beside the inter-node movement of the bacteria, the dynamics of individuals is specified by transitions encoding birth, death, and the epidemic behaviour.

The model of this system has state variables $\vec{X} = (X_S, X_I, X_R, X_B)$, counting the number of susceptible, infected, and recovered individuals, and the concentration of the bacteria in each location, respectively. The dynamics is specified by a list of transitions, each corresponding to an event and changes the system according to the transition rule (e.g,. $I \to R$ means an infected individual becomes recovered). Each transition has a speed governed by a rate function, that can depend on the concentration of the bacteria and individuals in each state.

For such a description, we can derive both a deterministic model (a set of ODEs) or a stochastic one (a patch CTMCs), that can respectively be integrated with an ODE solver or simulated with standard algorithms such as SSA or Gibson-Bruck. The solution of the integration/simulation is a spatio-temporal trace $\vec{x}(t,\ell) = (x_S(t,\ell), x_I(t,\ell), x_R(t,\ell) x_B(t,\ell))$ that describes the concentration of susceptible, recovered, infected individuals and bacteria, at each time, in each location. Such spatio-temporal trajectories are the input of our jSSTL tool. This means that SSTL specifies properties directly over the spatio-temporal traces of the system (with continuous time and discrete space). An interesting consideration about this is that we do not necessarily need a mathematical model but just observable traces of some process, including real observations. Hence, the tool can be used also for the analysis of black box systems for which we do not have a model but only information about their behaviour.

SSTL properties. SSTL integrates the temporal modalities of STL with two spatial operators: the *somewhere* operator, $\diamond_{[d_1,d_2]}$, and the *bounded surround* operator $\mathscr{S}_{[d_1,d_2]}$, where $[d_1,d_2]$ is a closed real interval with $d_1 < d_2$. There is a third derivable operator: the *everywhere* operator $\boxdot_{[d_1,d_2]}\, \varphi := \neg \diamond_{[d_1,d_2]} \neg \varphi$.

Let us consider the epidemic model described above, with an infection that starts at location ℓ_1. Suppose we wish to describe the diffusion of the epidemic among the communities. In particular, we wish to check whether the infection has propagated a certain distance from ℓ_1 after a given time. This behaviour can be captured by the SSTL formula:

$$\phi_1 = F_{[0,T_{inf}]} \diamond_{[d_1,d_2]} (X_I > c_{inf}), \tag{1}$$

verifying it in location ℓ_1. The precise meaning of the formula is:

> *Eventually, in less than T_{inf} unit time, the number of infected individuals becomes more than c_{inf} in a location ℓ with $d(\ell_1,\ell) \in [d_1,d_2]$, i.e., at a distance from location ℓ_1 equal or greater than d_1 and equal or less than d_2.*

Let us see also a more complicated property:

$$\phi_2 = \boxdot_{[0,d_{max}]} (\psi_1 \longrightarrow \psi_2), \tag{2}$$

where:

$$\psi_1 = F_{[0,T_{start}]}(\diamond_{[0,d_{near}]}(X_I > c_{inf}))$$
$$\psi_2 = (F_{[T_{start},T_{start}+DT]}\diamond_{[d_{far},d_{max}]}(X_I > c_{inf}))$$

The property states that a large infection (with at least c_{inf} individuals) happening at some time $t \in [0, T_{start}]$ and localised within distance d_{near} from a given reference point, will spread further away, at a location at distance between d_{far} and d_{max}, at some time $t' \in [T_{start}, T_{start} + DT]$. Furthermore, this is true for every reference point (say at distance at most d_{max} from ℓ_1).

Verification. Similarly to STL, SSTL has two semantics, the classical *boolean* semantics and a *quantitative* semantics. The boolean semantics returns true or false depending on whether the trace satisfies the SSTL property, i.e. $(\vec{x},t,\ell) \models \varphi$ is true if and only if the trace $\vec{x}(t,\ell)$ satisfies φ. The whole trace satisfies a property in location ℓ iff it satisfies the property at time zero, i.e. $(\vec{x},\ell) \models \varphi \Leftrightarrow (\vec{x},0,\ell) \models \varphi$. The trace is verified in each point in space, as we assume no privileged direction or location. The quantitative semantics, instead, returns a real value $\rho(\varphi,\vec{x},t,\ell)$ that quantifies the level of satisfaction of the formula by the trajectory $\vec{x}$ at time t in location ℓ. The sign of $\rho(\varphi,\vec{x},t,\ell)$ is related to the truth of the formula: if $\rho(\varphi,\vec{x},t,\ell) > 0$, then $(\vec{x},t,\ell) \models \varphi$, and similarly if $\rho(\varphi,\vec{x},t,\ell) < 0$, then $(\vec{x},t,\ell) \not\models \varphi$. In accordance with the boolean semantics, the quantitative value of the whole trace in location ℓ is given by its value at time zero, i.e. $\rho(\vec{x},\ell) = \rho(\vec{x},0,\ell)$. Given a spatio-temporal trace $\vec{x}(t,\ell)$ and a property φ, the logic has specific algorithms to compute the boolean and the quantitative satisfaction. For details about monitoring algorithms we refer the reader to [43, 44].

3.2 Interface

An Eclipse plug-in provides the user interface to specify and verify SSTL properties of spatio-temporal trajectories generated from the simulation of a system or from real observations. The user can specify the properties, describe a model of the space (i.e., its graph structure), import the trajectories, and then verify if such trajectories satisfy the specified properties. The instructions to download the plug-in and to create a jSSTL Project are available at `http://quanticol.sourceforge.net/`. Figure 9 shows a screenshot. It provides an *editor*, containing the script with the SSTL properties that we want to analyse in our scenario (on the left) and a *view* to visualise the space model, the data and the results of the analyses (on the right).

In the jSSTL editor, it is possible to define a script that contains the list of properties to be analysed. The editor is based on Xtext[3], a framework for development of programming languages and domain-specific languages. The syntax of the script of our language is reported in Figure 10. Besides the list of formulas, each script contains the list of the variables considered in the model, a set of constants, and a list of parameters that may occur in the formula. The parameters can be declared to take values in an interval. When the monitoring procedure is performed, the user can select a specific value for each parameter in the corresponding interval. Standard expressions can be used to define both constant and parameter intervals. In the script, each formula is associated with a name, that is used to select the specific property during the monitoring procedure.

On the left of Figure 9 is the file containing the script of properties (1) and (2), described in the previous subsection. First, we have the declaration of the value of each parameter (const cinf= 150, ...), then the declaration of the variables (variable I,..), and finally the description of each property, e.g., property (1) is defined as: formula phi1 = F[0,5]<<>>[12,15] { I>cinf }.

The jSSTL view provides three different panels: to visualise the spatial models (*Model* panel), to summarise the relevant data declared in a script (*Script* panel), and to plot the system's trajectories and the Boolean and quantitative satisfaction signals (*Signal* panel). Clicking on the icons on the top right of the view can perform the following actions: open a graph model, create a grid model, open an SSTL script, open signal data (the trajectory), execute the monitor, display the signal, and clear the data. The *Model* panel can be used to see a graph-based representation of the spatial model. The spatial model can be imported as a `.tra` file or, in the case of a regular grid, can be directly created, selecting the number of rows and columns. In Figure 11, we can see,

[3] `https://eclipse.org/Xtext/`

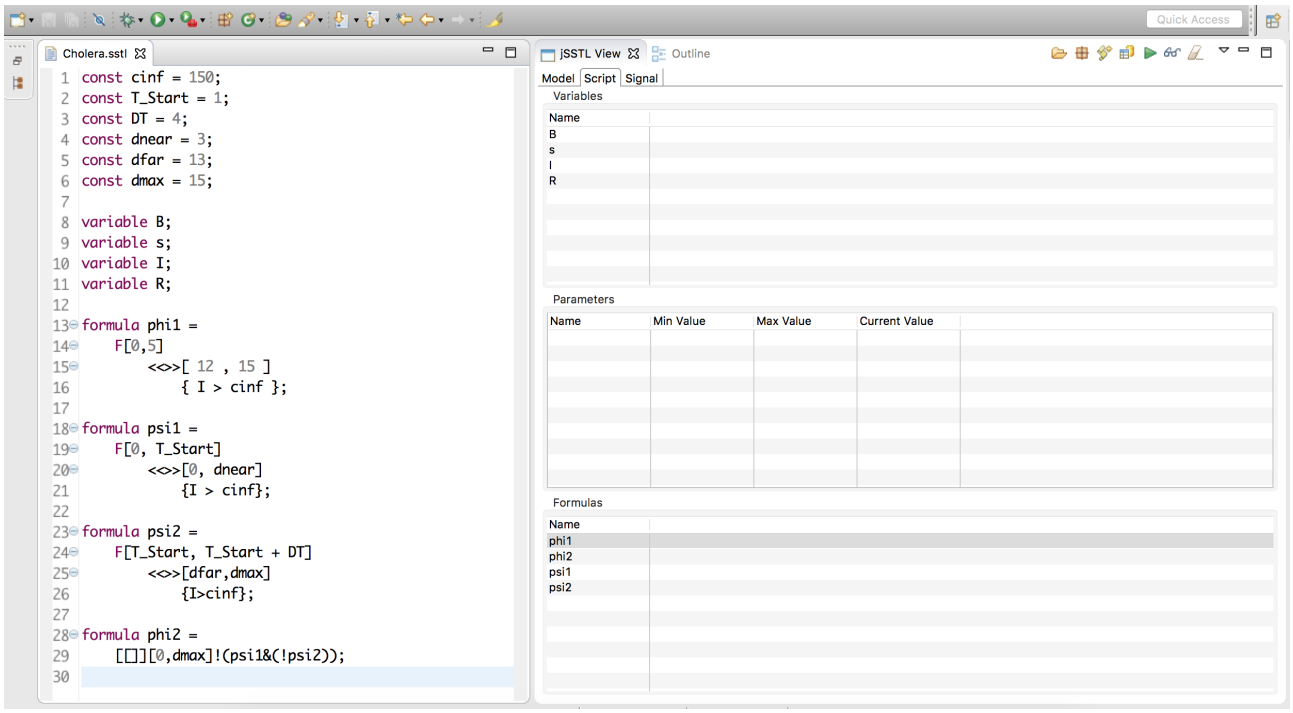


Figure 9: The jSSTL Eclipse plug-in.

in the model panel, the representation of the space for the Cholera model that corresponds to the graph reported in Figure 8. The *Script* panel outlines the information about our scenario. We can see here the list of variables, parameters and formula names. Currently, CSV and tabular based ASCII files are supported for both input and output of signals.

The traces have to be imported with a single file for each variable and location, e.g., for the infected individual I we will have the files values_i_I.dat, with $i \in \{1,..,7\}$. Having selected the space model, the trajectory and the property, we can compute the related Boolean and quantitative satisfaction signals. To do that, we have to select from the Script panel which property we want to verify, and to chose the specific values for the parameters for which we considered a range. In the *Signal* panel, the spatio-temporal trajectories and the Boolean and quantitative signals can be visualised. We can choose which variables and which locations to plot. In Figure 12, we can see the trace x_I, the variation in time of the number of infected individuals in each location. We can observe that, at time 0, we have infected individuals only in ℓ_1. The temporal trajectories of each location are indicated adding $@i$ to each variable, e.g., $x_I(t,\ell_1)$ is indicated in the plot as I@1.

Figure 13 shows the Boolean satisfaction of property psi1 (in each location). Instead, in Figure 14, we can see the quantitative satisfaction of phi1 and psi1 for location ℓ_1 only. We can see from the plot that the trajectories satisfy both properties, but that property psi1 is more robust than property phi1. The different length of the signals is due to the different temporal intervals in the two formulas. Indeed, in the monitoring procedure, if the signal has duration T, and a formula has temporal horizon h (i.e. its truth depends on what happens in h units of time in the future), then the duration of the Boolean/ satisfaction signal will be $T-h$. We recall that the satisfaction of a formula for the whole trace is determined at time zero and for a specific location; hence, we can say that the $\vec{x}$ satisfies ϕ_1 in location ℓ_1 iff phi1@1 at time 0 is greater than 0.

Both the plots of the trajectories of Boolean and quantitative signals can be easily exported. Furthermore, the tool also permits one to import a set of trajectories, instead of a single one, to compute the Boolean and quantitative satisfaction signals of such trajectories for a specific property and then to evaluate their mean (the average robustness) and their variance. This can be very useful when we want to analyse stochastic systems.

Here the model has been kept simple for the sake of clarity. However, the plug-in may work with much more complex models such as the one in [44].

We stress here a few features of the tool: it can handle traces coming from different sources, hence it can be

script	::=	element*
element	::=	variableDec | constDec | parameterDec | formulaDec
variableDec	::=	**variable** *name*
constDec	::=	**const** *name* = expr
parameterDec	::=	**parameter** *name* **in** interval
interval	::=	[expr, expr]
expr	::=	baseExpr | expr + expr | expr × expr | ⋯
baseExpr	::=	int | float | literalExpr
formulaDec	::=	**formula** *name* = formula
formula	::=	formula**&**formula | formula|formula
	|	formula**U**interval formula | **G**interval formula
	|	**F**interval formula | formula**S**interval formula
	|	<<>> interval formula | [[]]interval formula
	|	!formula | relExpr
relExpr	::=	expr<expr | expr≤expr | expr>expr | expr≥expr
	|	expr==expr | ! =expr

Figure 10: jSSTL formula syntax.

used to monitor simulations but also real systems, the parametrisation of formulas enables us to integrate the tool to perform parameter synthesis and estimation, see for instance [4], the possibility to directly check a set of trajectories can simplify the statistical analysis of stochastic systems, it has a quantitative semantics that can be used to evaluate the robustness of the satisfaction and to drive the behaviour of the system, see for instance [5].

4 *ERODE/UTOPIC*

ERODE is a recently developed software tool [16], which supports two complementary equivalence relations over ODE variables [15]: *forward differential equivalence*, yielding a self-consistent aggregate system where each ODE gives the cumulative dynamics of the sum of the original variables in the respective equivalence class; and *backward differential equivalence* identifies variables that have identical solutions whenever starting from the same initial conditions. As backend *ERODE* uses the well-known Z3 SMT solver [26], which computes the largest equivalence that refines a given initial partition of ODE variables. In the special case of ODEs with polynomial derivatives of degree at most two (covering affine systems and elementary chemical reaction networks [12]), it implements a more efficient partition-refinement algorithm in the style of Paige and Tarjan [14]. The theory has been developed in the context of WP3 and is reported in D3.3 as well as in the earlier D3.2.

ERODE is a mature tool that distinguishes itself from the prototypes accompanying [12, 14, 15] (and discussed in Deliverable D5.2) in that:

(i) It is not a command-line prototype but a mature tool with a modern integrated development environment;

(ii) It collects all the techniques of our framework for ODE reduction in a unified coherent environment;

(iii) It offers a language, and an editor, to express the entire class of ODEs supported by the reduction techniques, while the prototypes could reduce only chemical reactions networks (CRNs);

(iv) It implements an ODE workflow consisting of numerical solution and graphical visualisation of results;

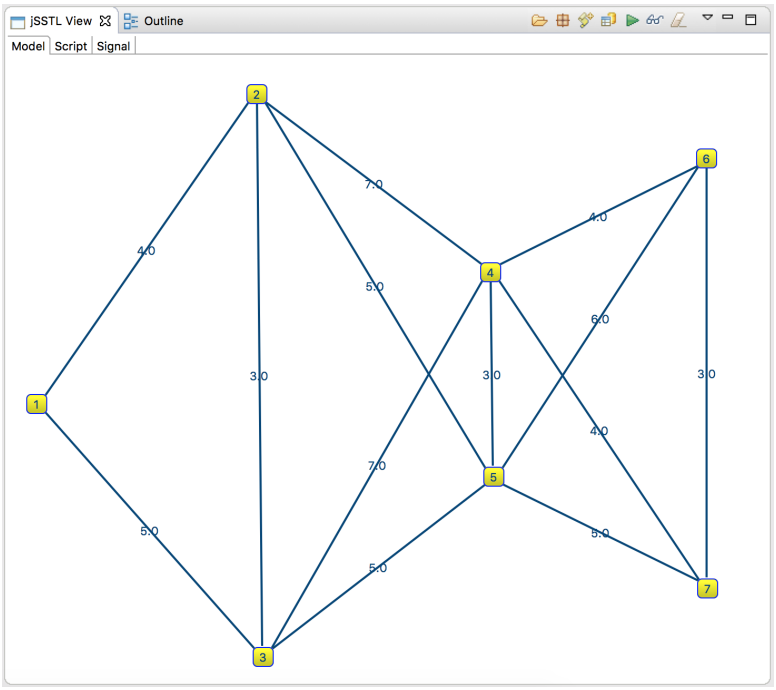


Figure 11: Visualisation of space in the jSSTL Eclipse plug-in.

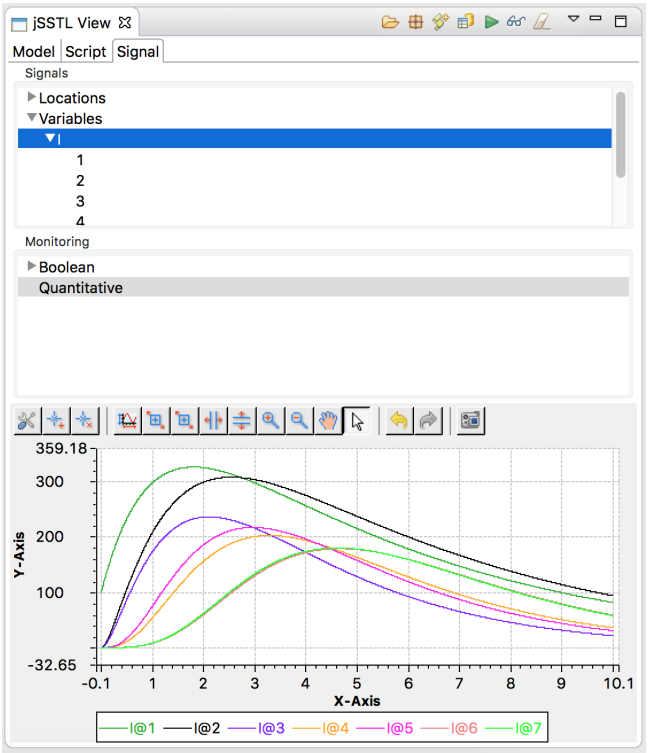


Figure 12: Plot of x_I, number of infected individual in each location.

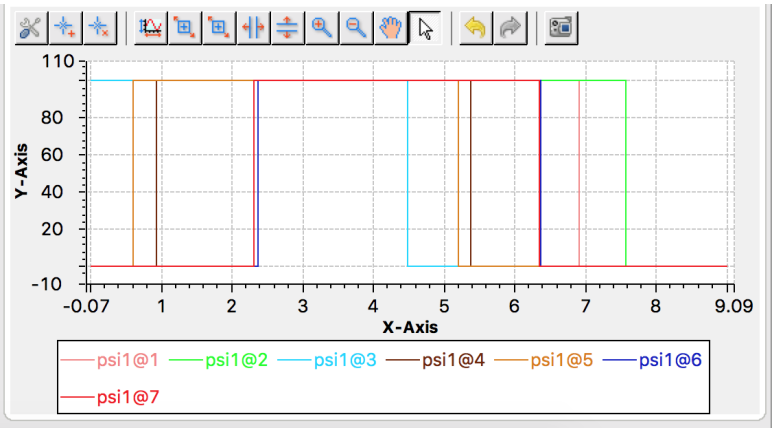


Figure 13: Plot of the Boolean semantics of the properties psi1.

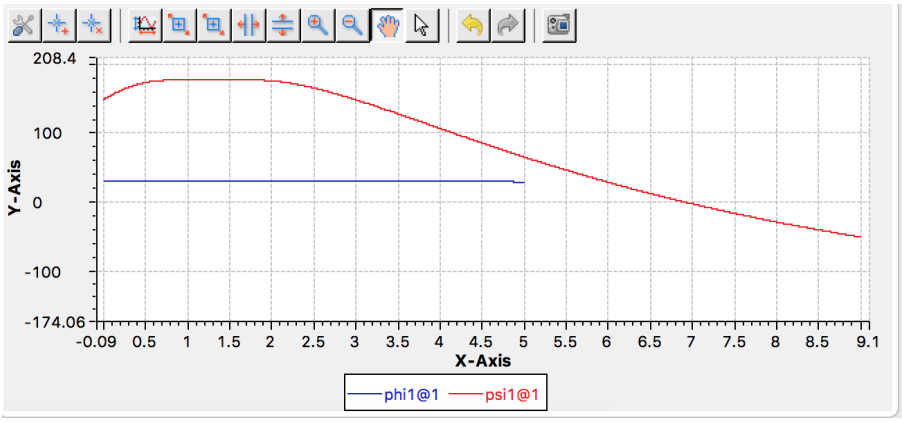


Figure 14: Plot of the quantitative semantics of the properties phi1 and psi1 for location 1.

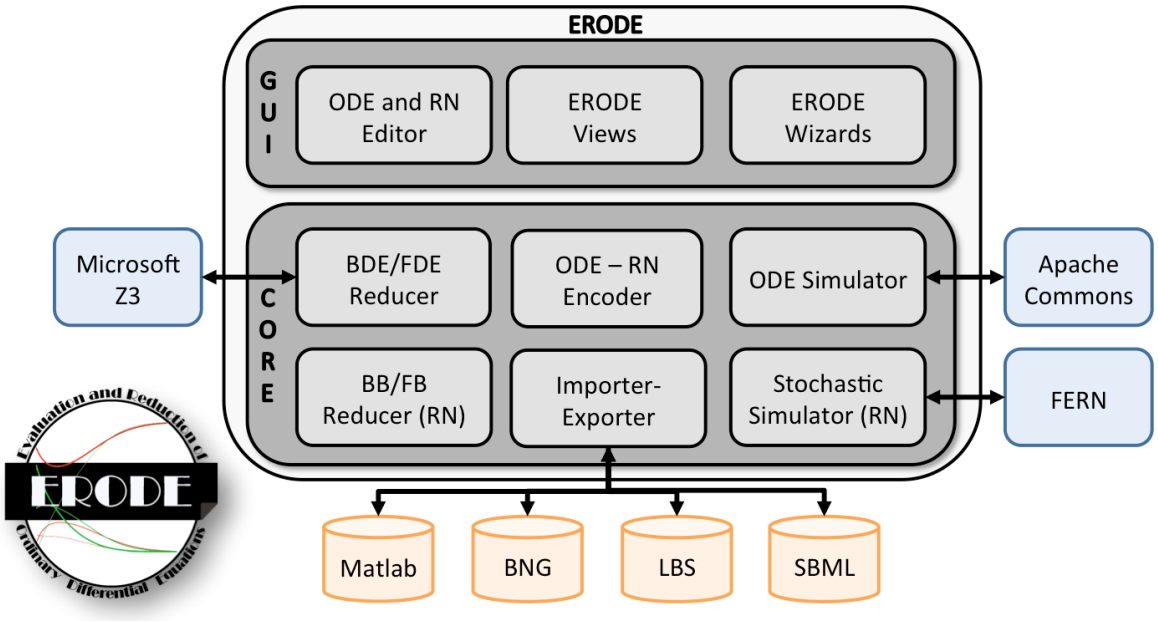


Figure 15: *ERODE*'s Architecture.

(v) It offers import/export facilities for other formats like biochemical models for the well-known tools BioNetGen [8] and Microsoft GEC [31], or ODEs defined in MATLAB.

More recently, it has been extended with *UTOPIC*, a command to obtain an under-approximation, at a given time point, of the reachable set of a linear combination of ODE variables under time-varying uncertainties. In particular, *UTOPIC* exploits properties (i), (iii), and (v) in that its input language is an ERODE model, which is automatically translated into a Matlab script which can be invoked to perform the required analysis.

4.1 Architecture

ERODE is an application based on the Eclipse framework for Windows, Mac OS and Linux. It does not require any installation process, and is available at `http://sysma.imtlucca.it/tools/erode`, together with a manual and sample models. Figure 15 provides a pictorial representation of the architecture of *ERODE*. It is organized in the presentation layer, with the graphical user interface, and the core layer. The main components of the GUI layer are depicted in the screenshot of *ERODE* in Figure 16, including a fully-featured text editor based on the *XTEXT* framework which supports syntax highlighting, content assist, error detection and fix suggestions (top-middle of Figure 16). This layer also offers a number of views, including a *project explorer* to navigate between different *ERODE* files (top-left of Figure 16); an *outline* to navigate the parts of the currently open *ERODE* file (bottom-left of Figure 16); a *plot view* to display ODE solutions (top-right of Figure 16); and a *console view* to display diagnostic information like warnings and model reduction statistics (bottom-right of Figure 16). Finally, the GUI layer offers a number of wizards for: (i) updating *ERODE* to the latest distribution; (ii) creating new *ERODE* files and projects; and (iii) importing models provided in third-party languages.

The core layer implements the minimisation algorithms and related data structures. A wrapper to Z3 via Java bindings is included. The core layer also provides support for numerical ODE solvers, using the Apache Commons Maths library [1]. When the input is a CRN (i.e. an RN with only positive rates) it can also be interpreted as a CTMC. Using the *FERN* library [29], *ERODE* features CTMC simulation.

4.2 Illustrating Example

Let us consider an idealised biochemical interaction between molecules A and B; A can be in two states, u (unphosphorylated) and p (phosphorylated) and can bind/unbind with B. This results in a network with five *species*, denoted by A_u, A_p, B, A_uB, and A_pB.

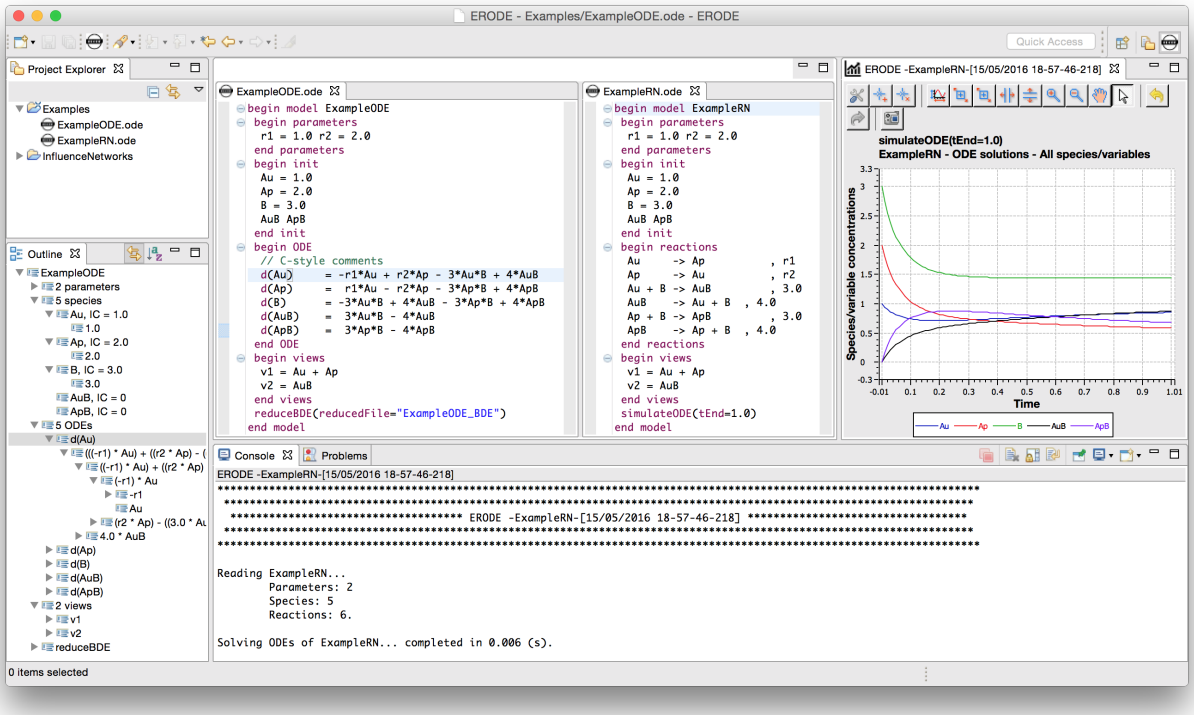


Figure 16: A screenshot of *ERODE*.

The dynamics of the system is described in Figure 17 (a) through a CRN with six reactions, where r_1, r_2, r_3 and r_4 are the kinetic constants. By applying the well-known law of mass action, each species is associated with one ODE variable which models the evolution of its concentration as a function of time, with reactions that fire at a speed proportional to their rate times the concentrations of their reagents. For example, $A_u + B \xrightarrow{r_3} A_uB$ fires at speed $r_3[A_u][B]$, where $[\cdot]$ denotes the current concentration of a species. Consequently, this term appears with negative sign in the ODEs of its *reagents* (A_u and B), and with positive sign in the ODE of its *product*, A_uB. The resulting ODEs for our sample system are shown in Figure 17 (b), where the 'dot' operator denotes the (time) derivative. The model is completed by an *initial condition* which assigns the initial concentration $[X](0)$ to each species X in the network.

Differential equivalences. It can be shown that $\{\{[A_u],[A_p]\},\{[B]\},\{[A_uB],[A_pB]\}\}$ is a forward differential equivalence (FDE) for our running example. Indeed, exploiting basic properties one can write self-consistent ODEs for the sums of species in each equivalence class:

$$\begin{aligned} [\dot{A_u}] + [\dot{A_p}] &= -r_3\big([A_u]+[A_p]\big)[B] + r_4\big([A_uB]+[A_pB]\big), \\ [\dot{B}] &= -r_3\big([A_u]+[A_p]\big)[B] + r_4\big([A_uB]+[A_pB]\big), \\ [\dot{A_uB}] + [\dot{A_pB}] &= \ \ r_3\big([A_u]+[A_p]\big)[B] - r_4\big([A_uB]+[A_pB]\big). \end{aligned} \tag{3}$$

By the change of variables $[A] = [A_u] + [A_p]$ and $[AB] = [A_uB] + [A_pB]$, we get:

$$[\dot{A}] = -r_3[A][B] + r_4[AB], \quad [\dot{B}] = -r_3[A][B] + r_4[AB], \quad [\dot{AB}] = r_3[A][B] - r_4[AB]$$

This *quotient* ODE model essentially disregards the phosphorylation status of the A molecule. Setting the initial condition $[A](0) = [A_u](0) + [A_p](0)$ and $[AB](0) = [A_uB](0) + [A_pB](0)$ yields that the solution satisfies $[A](t) = [A_u](t) + [A_p](t)$ and $[AB](t) = [A_uB](t) + [A_pB](t)$ at all time points t.

Backward differential equivalence (BDE) equates variables that have the same solutions at all time points, if initialised equally. It can be shown that $\{\{[A_u],[A_p]\},\{[B]\},\{[A_uB],[A_pB]\}\}$ is also a BDE if $r_1 = r_2$. In this

$$
\begin{array}{rl}
A_u \xrightarrow{r_1} A_p & \dot{[A_u]} = -r_1[A_u] + r_2[A_p] - r_3[A_u][B] + r_4[A_uB] \\
A_p \xrightarrow{r_2} A_u & \dot{[A_p]} = r_1[A_u] - r_2[A_p] - r_3[A_p][B] + r_4[A_pB] \\
A_u + B \xrightarrow{r_3} A_uB & \dot{[B]} = -r_3[A_u][B] + r_4[A_uB] - r_3[A_p][B] + r_4[A_pB] \\
A_uB \xrightarrow{r_4} A_u + B & \dot{[A_uB]} = r_3[A_u][B] - r_4[A_uB] \\
A_p + B \xrightarrow{r_3} A_pB & \dot{[A_pB]} = r_3[A_p][B] - r_4[A_pB] \\
A_pB \xrightarrow{r_4} A_p + B &
\end{array}
$$

(a) (b)

Figure 17: CRN model (a) and underlying ODEs (b) of an idealised biochemical interaction.

case, we obtain a quotient ODE by keeping only one variable (and equation) per equivalence class, say $[A_u]$, $[B]$ and $[A_uB]$, and rewriting every occurrence of $[A_p]$ and $[A_pB]$ as $[A_u]$ and $[A_uB]$, respectively:

$$
\begin{aligned}
\dot{[A_u]} &= -2r_1[A_u] - r_3[A_u][B] + r_4[A_uB] \\
\dot{[B]} &= -2r_3[A_u][B] + 2r_4[A_uB] \\
\dot{[A_uB]} &= r_3[A_u][B] - r_4[A_uB]
\end{aligned}
$$

Both FDE and BDE yield a reduced model that can be exactly related to the original one. BDE is lossless, because every variable in the same equivalence class has the same solution, but it is subject to the constraint that variables in the same block be initialised equally. Instead, with FDE one cannot recover the individual solution of an original variable in general, but no constraint is imposed on the initial conditions.

Symbolic minimisation algorithms. In [15], establishing that a given partition is a differential equivalence amounts to checking the equality of the functions representing their derivatives. This is encoded in (quantifier-free) first-order logic formulas over the nonlinear theory of the reals. The problem is decidable for a large class of ODEs (and Z3 implements a decision procedure [35]). Such a class is identified by the IDOL language of [15], covering polynomials of any degree, rational expressions, minima and maxima. This captures affine systems, CRNs with mass-action or Hill kinetics [51], and the deterministic *fluid* semantics of process algebra [32, 49].

A partition of variables is a BDE if any assignment with equal values in any equivalence class has equal derivatives within each equivalence class. Thus, $\{\{[A_u],[A_p]\},\{[B],[A_uB],[A_pB]\}\}$ is a BDE if and only if the following formula is *valid* (i.e. true for all assignments to the real variables $[A_u]$, $[A_p]$, $[B]$, $[A_uB]$, and $[A_pB]$):

$$
[A_u] = [A_p] \wedge [B] = [A_uB] = [A_pB] \implies f_{[A_u]} = f_{[A_p]} \wedge f_{[B]} = f_{[A_uB]} = f_{[A_pB]} \tag{4}
$$

where $f_{[\cdot]}$ stands for the derivative assigned to the corresponding species in Figure 17 (b). As usual, the SMT solver will check the satisfiability of its negation.

To automatically find differential equivalences of an ODE model, the SMT checks are embedded in a partition-refinement algorithm that computes the largest differential equivalence which refines a given input partition of variables. In particular, a current partition is refined at each step using the *witness* returned by the SMT solver, i.e. a variable assignment that falsifies the hypothesis that the current partition is a differential equivalence. The algorithm terminates when no witness is found, guaranteeing that the current partition is a differential equivalence. Let us fix the rates $r_1 = r_2 = 1$, $r_3 = 3$ and $r_4 = 4$. Then, $\{\{[A_u],[A_p]\},\{[B],[A_uB],[A_pB]\}\}$ is not a BDE for our running example. Indeed, the assignment $\{[A_u] = 1, [A_p] = 1, [B] = 2, [A_uB] = 2, [A_pB] = 2\}$ is a witness for the negation of Equation 4, since we get $f_{[A_u]} = 2$, $f_{[A_p]} = 2$, $f_{[B]} = 4$, $f_{[A_uB]} = -2$ and $f_{[A_pB]} = -2$ under this assignment. This information is used to refine the current partition by splitting its blocks into sub-blocks that have the same computation of derivative, obtaining $\{\{[A_u],[A_p]\},\{[B]\},\{[A_uB],[A_pB]\}\}$. No witness can be generated for this partition, ensuring that it is a BDE.

ERODE specifications of the running example in Figure 17.

```
begin model ExampleODE
 begin parameters
  r1 = 1.0
  r2 = 2.0
 end parameters
 begin init
  Au = 1.0 Ap = 2.0 B = 3.0
  AuB = 0 ApB = 0
 end init
 begin partition
  {Au,Ap}, {AuB}, {B,ApB}
 end partition
 begin ODE
  // C-style comments
  d(Au)   = -r1*Au + r2*Ap - 3*Au*B + 4*AuB
  d(Ap)   =  r1*Au - r2*Ap - 3*Ap*B + 4*ApB
  d(B)    = -3*Au*B + 4*AuB - 3*Ap*B + 4*ApB
  d(AuB)  =  3*Au*B - 4*AuB
  d(ApB)  =  3*Ap*B - 4*ApB
 end ODE
 begin views
  v1 = Au + Ap
  v2 = AuB
 end views
 reduceBDE(reducedFile="ExampleODE_BDE.ode")
end model
```

Listing 1: Direct ODE specification.

```
begin model ExampleRN
 begin parameters
  r1 = 1.0
  r2 = 2.0
 end parameters
 begin init
  Au = 1.0 Ap = 2.0 B = 3.0
  AuB ApB
 end init
 begin partition
   {Au,Ap}, {AuB}
 end partition
 begin reactions
  Au     -> Ap      , r1
  Ap     -> Au      , r2
  Au + B -> AuB     , 3.0
  AuB    -> Au + B  , 4.0
  Ap + B -> ApB     , 3.0
  ApB    -> Ap + B  , 4.0
 end reactions
 begin views
  v1 = Au + Ap
  v2 = AuB
 end views
 simulateODE(tEnd=1.0)
end model
```

Listing 2: Reaction network.

The FDE case is more involved, as discussed in [15]. Considering our running example, we have that $\{\{[A_u],[A_p]\},\{[B],[A_uB],[A_pB]\}\}$ is an FDE if and only if

$$(f_{[A_u]} + f_{[A_p]} = \hat{f}_{[A_u]} + \hat{f}_{[A_p]}) \wedge (f_{[B]} + f_{[A_uB]} + f_{[A_pB]} = \hat{f}_{[B]} + \hat{f}_{[A_uB]} + \hat{f}_{[A_pB]}) \tag{5}$$

is *valid*, where each $\hat{f}_{[\cdot]}$ is obtained from the corresponding derivative $f_{[\cdot]}$ by replacing each variable with the sum of the variables in its block divided by the size of the block. For example, each occurrence of the term $r_4[A_uB]$ is replaced by $r_4\frac{[B]+[A_uB]+[A_uB]}{3}$. It can be shown that the partition is not an FDE, because a witness falsifying Equation 5 can be found by the SMT solver. However, differently from the BDE case, Equation 5 does not compare single derivatives, but sums of derivatives, hence it cannot be used to decide how to refine the partition. For this, a "binary" characterisation of FDE performs SMT checks on each pair of species in the same block of a partition to decide if they have to be split into different sub-blocks.

We remark that the algorithms allow the preservation of user-defined observables. For instance, a variable of interest can be put in an initial singleton block when reducing with FDE. Similarly, in order to meet the constraints on BDE, one can build an initial partition *consistent* with the initial conditions of the original model (that is, two variables are in the same initial block if their initial conditions are the same).

Syntax-driven minimisation. A reaction network (RN) differs from an elementary CRN in that the kinetic constants may be negative. This gives rise to an ODE system with derivatives that are multivariate polynomials of degree at most two [14]. Forward and backward bisimulation are equivalence relations over variables/species in the Larsen-Skou style of probabilistic bisimulation [36]. They are defined in terms of quantities computed by inspecting the set of reactions [36]. In [14], this bisimulation style enabled the adaptation of Paige and Tarjan's coarsest refinement problem [46] to compute the largest one. This is done by generalising algorithms for Markov chain lumping [27, 50], obtaining algorithms with $\mathcal{O}(m \cdot n \cdot \log n)$ and $\mathcal{O}(m \cdot n)$ time and space complexity, respectively, with m being the number of monomials appearing in the underlying ODE system, and n the number of ODE variables.

4.3 Language

The two alternative specification formats of our running example from Figure 17 can be expressed in *ERODE* in Listings 1 and 2. There are six components of an *ERODE* specification:

(i) *Parameter specification* to set values of initial conditions, kinetic rates, or views (block `parameters`).

(ii) *Variable declaration* and corresponding initial conditions (block `init`).

(iii) *Initial partition of variables* to be used as the initial partition of the partition-refinement algorithms, as described later (block `partition`). The user is required to specify only the partition blocks of interest, while all variables not mentioned explicitly are assigned to an implicit additional block.

(iv) *ODE system* either in plain format (block `ode`) or as an RN (block `reactions`).

(v) *Views*, i.e., the quantities to be tracked by the numerical solver. For instance, in Listings 1 and 2 the intent is to collect the total concentration of the `A`-molecules, regardless of their phosphorylation state (view `v1`), and the concentration of the species `AuB` (view `v2`).

(vi) *Commands* for ODE numerical solution, reduction, *UTOPIC*, and exporting into other formats.

Reduction commands. All ODE reduction commands share the common signature

```
reduce<kind>(prePartition=<NO|IC|USER>, reducedFile=<name>)
```

where `kind` can be `FDE`, `BDE`, `FB`, or `BB`. The ODE input format affects which reduction options are available. For an ODE system defined directly, only FDE and BDE are enabled. Polynomial-time minimisations up to forward and backward bisimulation (FB and BB, respectively) are additionally available for RNs representing polynomial ODE systems of degree at most two [14]. This is imposed by having reagents multisets of size at most two in each reaction and restricting to mass-action type rate expressions.

The optional parameter `prePartition` defines the initial partition for the minimisation algorithm. The maximal aggregation is obtained with the `NO` option (which is the default). If it is set to `IC`, the initial partition is built according to the constraints given by the initial conditions: variables are in the same initial block whenever their initial conditions are equal. If the option is set to `USER`, then the partition specified in the `partition` block will be used.

If `reducedFile` is present, then a reduced model will be generated according to the computed partition following the model-to-model transformation from [12] (for FB and BB) and [15] (for FDE and BDE). This will have the same format as the input, and will contain one variable for each equivalence class. The name of the variable is given by the first variable name in that block, according to a lexicographical order.

Considering our running example, no reduction is found executing `reduceFDE` on Listing 1, with the pre-partitioning option set to `USER`. Instead, when it is set to `NO` we find the FDE $\{\{A_u,A_p\},\{B\},\{A_uB,A_pB\}\}$ discussed in Section 4.2, implying that it is the maximal one of the model. The output file for the case without pre-partitioning is provided in Listing 3, which also shows that the association between the original ODE variables and those in the reduced model is maintained by annotating the output file with comments alongside the new variables.[4] This information can be useful for visually inspecting the reduced model in order to gain insights into the physical interpretation of the reduction [12]. Finally, we note that each reduced species has initial concentration equal to the sum of those in the corresponding block.

In Section 4.2 we have shown that the partition $\{\{A_u,A_p\},\{B\},\{A_uB,A_pB\}\}$ is also a BDE provided that $r_1 = r_2$. However, this reduction is not found if running `reduceBDE` with pre-partitioning set to `IC`, as it violates the initial conditions for `Au` and `Ap`. Instead, if the pre-partitioning is disabled, then the above partition is the coarsest refinement, but the user is warned about the inconsistency with the initial conditions. The BDE reduction without pre-partitioning for `r1=r2=1.0` is given in Listing 4. The initial condition for the ODE of each representative is equal to that of the corresponding original variable.

[4] Here output files have been typographically adjusted to improve presentation.

ERODE reductions of the running example in Figure 17.

```
begin model ExampleODE_FDE
 begin parameters
  r1 = 1.0
  r2 = 2.0
 end parameters
 begin init
  Au  = 1.0 + 2.0
  B   = 3.0
  AuB
 end init
 begin ODE
  d(Au)   = - 3*Au*B + 4*AuB
  d(B)    = - 3*Au*B + 4*AuB
  d(AuB)  =   3*Au*B - 4*AuB
 end ODE
 //Comments of the species
 //Au:  Block {Au, Ap}
 //B:   Block {B}
 //AuB: Block {AuB, ApB}
end model
```

Listing 3: FDE reduction.

```
begin model ExampleODE_BDE
 begin parameters
  r1 = 1.0
  r2 = 1.0
 end parameters
 begin init
  Au  = 1.0
  B   = 3.0
  AuB
 end init
 begin ODE
  d(Au)   = - 3*Au*B + 4*AuB
  d(B)    = - 6*Au*B + 8*AuB
  d(AuB)  =   3*Au*B - 4*AuB
 end ODE
 //Comments of the species
 //Au:  Block {Au, Ap}
 //B:   Block {B}
 //AuB: Block {AuB, ApB}
end model
```

Listing 4: BDE reduction.

```
begin model ExampleRN_BB
 begin parameters
  r1 = 1.0   r2 = 1.0
 end parameters
 begin init
  Au = 1.0  B = 3.0  AuB  SINK
 end init
 begin reactions
  Au -> 2*Au , r2
  Au -> SINK , r1
  Au + B -> Au , 3.0
  Au + B -> AuB , 3.0
  AuB -> Au + B , 4.0
  AuB -> B + AuB , 4.0
 end reactions
 //Comments of the species
 //Au:  Block {Au, Ap}
 //B:   Block {B}
 //AuB: Block {AuB, ApB}
end model
```

Listing 5: BB reduction.

The model of Listing 2 is not reduced by FB, independently on the pre-partitioning choice. This is consistent with the fact that FB is only a sufficient condition for FDE (although it is effective on many meaningful models from the literature, as discussed in [14]). The result of the BB reduction is instead provided in Listing 5. As for BDE, we considered the case `r1=1.0` and `r2=1.0` without pre-partitioning. It can be shown that the underlying ODEs of the reduced model correspond to those of Listing 4, as expected. (The place-holder species `SINK` is created to rule out reactions that have no products.)

UTOPIC. *UTOPIC* (Under-approximation Through Optimal Control) implements an algorithm from [10], where the basic idea is to interpret uncertain parameters of an ODE system as controls and to minimise and maximise a given linear combination of state variables using Pontryagin's principle, a well-established result in optimal control theory. The interval spanned by the so-obtained extrema can be shown to be an under-approximation of the reachable set of the linear combination. Moreover, thanks to the fact that Pontryagin's principle is a necessary condition for optimality, the aforementioned interval can be expected to be tight enough to cover the actual reachable set in many cases.

ERODE acts as a front-end that generates a MATLAB script implementing the algorithm. In our running example we have two parameters: `r1` and `r2`, to which we arbitrarily assigned values 1 and 2, respectively. With the following `utopic` command we can study the impact on the dynamics of a relaxation of such assumptions, considering `r1` and `r2` as uncertain time-varying parameters in $[0.9, 1.1]$ and $[1.9, 2.1]$, respectively:

```
utopic(fileOut="bu.m", tEnd=1, paramsToPerturb={r1 in [0.9,1.1],r2 in [1.9,2.1]},
       coefficients={AuB:1}, kMax=400, epsilon=1e-3)
```

Coherently with the view `v2` from Listings 1–2, here the modeller is interested in studying the concentration of species *AuB*. This is indicated by the option `coefficients` used in the command. The command takes the further additional settings: the output MATLAB file; the time horizon (here this is set to 1.0, as done in Listing 2); the maximum number of iterations (`kMax`); and the relative tolerance ε. Instead, the initial conditions are taken from the ODEs specification (within the `init` block).

By running the generated script, *UTOPIC* returns the interval $[0.81867, 0.89381]$ as an under-approximation of the concentration of *AuB* at time $T = 1.0$. This is consistent with the `simulateODE` command from Listings 2, which computes value 0.878 for the view `v2` at time time $T = 1.0$ when `r1` and `r2` have their original constant values.

5 *topochecker*

topochecker is a spatio-temporal model checker based on closure spaces and Kripke frames. It checks a spatial extension of CTL (*Computation Tree Logic*) named STLCS (*spatio-temporal logic of closure spaces*). The spatial fragment was presented in [20, 21], whereas the spatio-temporal logic was introduced in [19].

topochecker has been used in the QUANTICOL project in the smart buses [18], and bike sharing [22] scenarios. Integration with other tools in order to extend it to *statistical model checking* has been done, in the bike sharing context, in [23]. Recently (see the position paper [6]), *topochecker* has been extended in collaboration with *Azienda Ospedaliera Universitaria Senese* (the university hospital of Siena). The tool is currently in use in case studies aimed at segmentation of medical images via spatial model checking. The extended version of *topochecker* is available in the experimental branch of the source code repository of the tool. Multi-dimensional medical images are loaded as spatial models. The tool is capable of distance-based reasoning in linear time, using the techinque of *distance transforms*, borrowed from the field of Computational Imaging. Furthermore, a logical operator for *statistical texture analysis* has been implemented. The expressive power of the obtained logical language is high. In first experiments, using about 30 lines of specification, medical physicists in Siena have been able to identify glioblastoma and the associated oedema with confidence and execution times in line with the state of the art in the field of automated segmentation of the tumour. A clinical study is currently in progress.

5.1 Architecture

The tool, available as a stand-alone application[5], and distributed under an open source license, is a *global model checker* using a *dynamic programming* algorithm to verify STLCS formulas on finite models. Models are composed of a temporal part, which is a Kripke structure, and a spatial part, which is a finite, *quasi-discrete* closure space (see [21]). The Kripke structure and the spatial structure are given in the `dot` graph description language[6], and valuations of atomic propositions are provided by a *comma-separated-values* file associating to each point in space-time a list of propositions. The tool enriches basic STLCS by allowing users to specify that some atomic properties have an associated floating-point value; therefore, atomic propositions can be basic comparisons between the name of an atomic property and a floating point value. The tool permits parametric macro abbreviations.

The model checker is written in the programming language OCaml[7], and carefully optimised. Native arrays and native memory management are used (through the OCaml library `bigarray`); the algorithm uses a table of $k \times s \times f$ memory words, where k is the number of temporal states, s is the number of spatial locations, and f is the number of subformulas of a formula. One pass is executed over this table, filling it with a truth value for every cell, running in $O(k \times s \times f)$ steps. The model checker uses *memoization* of results for each subformula. The cache is stored on-disk, so that checking again the same formulas over the same models is done in negligible time, leveraging incremental design of complex formulas through multiple execution sessions.

5.2 Usage

A *topochecker* session consists of a text file, usually identified by the extension `.topochecker`, containing instructions for the model checker, and in particular: a *model declaration*; an optional list of *macro declarations*; a list of commands, instructing the model checker on what formulas to verify and where to save output files. The tool is simply invoked on the command line as follows:

```
topochecker filename.topochecker
```

and produces as output a set of files representing the spatial snapshot of the valuation of formulas at temporal states that are specified in the session file.

[5] Websites: `http://topochecker.isti.cnr.it`, `https://github.com/vincenzoml/topochecker`

[6] `http://www.graphviz.org/Documentation.php`.

[7] See `http://www.ocaml.org`.

Model declarations. A model declaration specifies the kind of model to be loaded. The stable branch of the tool currently has loaders for graphs in the *graphviz* format, whereas the experimental branch also loads images (including multi-dimensional medical images). We shall discuss the stable branch in the following. Model declaration (one per *topochecker* session) is written as

```
Kripke "kripke.dot" Space "space.dot" Eval "eval.csv";
```

where `kripke.dot`, `space.dot` and `eval.csv` are arbitrary file names. The file `kripke.dot` is a graphviz file defining a directed graph, which is the Kripke frame describing the temporal evolution of the system. Node identifiers must be numbers starting at 0 and with no gaps. The file `space.dot` is another graph, obeying the same conventions on node identifiers, which is the *quasi-discrete closure space* used as a spatial part of the model. Both graphs can be weighted; to achieve this, one adds the property `weight=n` to each edge, where `n` is a floating point constant (with default value 1.0). In the experimental branch of the tool, also implementing distance-based operators, such weights are used to compute shortest-path distances (Euclidean distances are also implemented using node positioning, but this has been mostly tested with medical images rather than graphs). The file `eval.csv` contains the valuation of atomic propositions. Proposition symbols follow standard conventions for identifiers in programming languages; for each state in the Kripke structure, and each point in space, any proposition may be assigned a truth value (if a truth value is not assigned, the value `false` is assumed). Furthermore, propositions may also assume quantitative values; the syntax of the logic is extended with basic constraints on such quantitative values. Concretely, `eval.csv` is a *comma-separated values* file with three or more columns. Each row takes the form

```
state,point,prop1,prop2,...,propN
```

In each row, `state` is a node identifier present in `kripke.dot`, `point` is a node identifier present in `space.dot`, and `prop1,...,propN` are identifiers (at least one must be present for each row). The meaning is that each of `prop1,...,propN` is true at the specified state and point of the space. Alternatively, each of `prop1,... propN` may take the form `prop=num`, associating a numeric (possibly floating point) quantity to each proposition letter. Actually, the first form is a shorthand for `prop=1`. The tool represents truth values as floating point constants, where 1 is the value true, and 0 is the value false. Note that each pair of state and point identifier can be repeated many times, if needed; different atomic propositions will be accepted in different rows.

Macro declarations and commands. Macro declarations are defined as lists of statements in the following form: `Let ide = FORMULA;` or `Let ide (arg1,...,argN) = FORMULA;`; macros are simply replaced in formulas; recursion is not permitted. Model checking is invoked by the `Check` statement as follows:

```
Check "COLOUR" FORMULA;
```

where COLOUR is an integer constant (either decimal, or hexadecimal, as in `0xRRGGBB`), that denotes an RGB color. `FORMULA` is a STLCS formula (which can use macros as explained above). The effect of such command is to colour the points satisfying `FORMULA`, using `COLOUR`, in the *current* output file. Output files are specified by statements in the form:

```
Output "PREFIX" state1,...,stateN;
```

where `PREFIX` will be used as a *prefix* for saving a spatial snapshot at each state; `state1,...,stateN` is an optional list of state identifiers (defined in `kripke.dot`), which, if present, restricts output to only the specified states. If such state identifiers are not specified, one file for *each* state in `kripke.dot` is saved. Once an output prefix has been defined, subsequent `Check` statements will create several files, named `PREFIX-N.dot`, where `N` is replaced by a state identifier (one per requested state, or per existing state, as explained above). More than one `Output` statement may be included per session; this permits one to save results for several different formulas, in the same model checking session (therefore, reusing the global model checking cache for each output prefix).

Syntax of formulas. The concrete syntax of the tool is described by the following grammar.

$$
\begin{array}{rcll}
\Phi & ::= & \texttt{TT} & \text{true} \\
 & | & \texttt{FF} & \text{false} \\
 & | & [\mathrm{p}] & \text{proposition letter} \\
 & | & [\mathrm{p} \bowtie \mathrm{n}] & \text{atomic constraint} \\
 & | & \texttt{!}\Phi & \text{not} \\
 & | & \Phi_1 \,\&\, \Phi_2 & \text{and} \\
 & | & \Phi_1 \,|\, \Phi_2 & \text{or} \\
 & | & \texttt{N}\Phi & \text{closure} \\
 & | & \texttt{I}\Phi & \text{interior} \\
 & | & \Phi_1 \,\texttt{S}\, \Phi_2 & \text{surrounded} \\
 & | & \texttt{A}\Psi & \text{for all paths} \\
 & | & \texttt{E}\Psi & \text{exists path} \\
 & & & \\
\Psi & ::= & \texttt{X}\Phi & \text{next} \\
 & | & \texttt{G}\Phi & \text{globally} \\
 & | & \Phi_1 \texttt{U} \Phi_2 & \text{until}
\end{array}
$$

where $\bowtie$ is a comparison operator drawn from $=, \leq, \geq, <, >$. Elements of the language are: the usual Boolean truth values and connectives; proposition letters; constraints on the quantitative value of propositions; the spatial operators *near* and *surrounded*; the temporal operators *for all paths* and *exists path* from CTL; the well-known path operators of CTL. Examples of use on the smart bus scenario are presented in Section 6.2.

6 Applications to Smart Cities

In this section we describe how the QUANTICOL software tool suite can be put to work for the analysis of CAS using the project's case studies. Specifically, Section 6.1 presents an analysis of bike-sharing systems that involves spatio-temporal model checking with *jSSTL* and model reduction with *ERODE*. Instead, Section 6.2 discusses applications of *topochecker* to the analysis of data from a bus transportation system. The application of *FlyFast* to smart-city scenarios as well as further analysis with *jSSTL* are illustrated in D4.3 in the context of the tools' integration with the CARMA Eclipse plug-in.

6.1 Bike-sharing Analysis

In this section, we present an analysis of the London Santander Cycles Hire scheme, a bike sharing system, modelled as a Population Continuous Time Markov Chain (PCTMC) with time-dependent rates. We use jSSTL to study a number of spatio-temporal properties of the system and to explore its robustness for a set of formulas parameters. Then, in Section 6.1.3 we show how *ERODE* can be used to identify symmetries within a time-homogeneous variant of the model.

6.1.1 Model

The Bike-Sharing System (BSS) is composed of a number of bike stations, distributed over a geographic area, for example a city. Each station has a fixed number of bike slots. The users can pick up a bike, use it for a while, and then return it to another station of the area. Following [30], we model the BSS as a PCTMC with time-dependent rates. The model, given the bike availability in a station at time t, predicts the probability distribution of the number of available bikes in that station at time $t+h$ with $h \in [0,40]$ minutes. The parameters of the model have been set using the historic journey data and bike availability data from January 2015 to March 2015

from the London Santander Cycles Hire scheme. In detail, the model contains the following transitions:

$$\begin{aligned}
&B_i \rightarrow S_i && \text{at rate } out_i(t), && \forall i \in (1,N)\\
&S_i \rightarrow B_i && \text{at rate } in_i(t), && \forall j \in (1,N)\\
&B_i \rightarrow S_i + T_j^i && \text{at rate } out_j^i(t), && \forall i,j \in (1,N)\\
&S_i + T_j^i \rightarrow B_i && \text{at rate } in_j^i(\#T_j^i), && \forall i,j \in (1,N)
\end{aligned}$$

where B_i (respectively S_i) represents the bike agent (respectively the slot agent) in the i^{th} station, $out_i(t)$ (respectively $in_i(t)$) is the bike pickup (respectively return) rate in station i at time t, T_j^i is the agent of a bike picked up in station i that will be returned in station j, $out_j^i(t) = out_i(t) * p_j^i(t)$, where $p_j^i(t)$ is the probability that a journey will end at station j given that it started from station i at time t, in_j^i is the return rate of a bike picked up in station i that will be returned in station j, $\#T_j^i$ denotes the population of an agent type T_j^i and N is the number of stations. Overall, the model consists of 733 bike stations and 57713 agents.

6.1.2 Verification of spatio-temporal logics using jSSTL

We simulated the model using Simhya [9], a Java tool for the simulation of stochastic and hybrid systems. In particular we exploit its Gibson-Bruck algorithm. The time-dependent rates change 2 times, all at the same time unit: the first interval is 14 minutes, the second interval is 20 minutes, the third interval is 6 minutes; we simulate the model for 40 minutes. Then, we consider the trajectories only of the bike (B) and slot (S) agents, in each station. Our spatio-temporal trace is then $X_B(t,\ell) = (X_B(t,\ell), X_S(t,\ell))$, i.e., the number of bikes and free slots at each time, in each station. The space is represented by a fully connected weighted graph, where the nodes are the stations and the edges describe the connections between stations. The weight function $w : E \rightarrow \mathbb{R}$ returns the distance between stations, where $E = L \times L$ is the set of edges and L is the set of stations.

One of the main problems of these systems is the availability of bikes or free slots in a specific station at a given time. Two important questions related to this issue are:

- If I do not find a bike/free slot, how long should I wait before another user returns a bike/picks one up?
- If I do not find a bike/free slot, is there another station at the distance less than a certain value where I can find a bike/free slot?

These behaviours can be captured with the SSTL properties described below.

$$\phi_1 = \mathscr{G}_{[0,T_{end}]}\{\diamond_{[0,d]}(B > 0) \wedge \diamond_{[0,d]}(S > 0)\} \tag{6}$$

A station ℓ satisfies ϕ_1 if and only if it is always true that, between 0 and T_{end} minutes, there exists a station at a distance less than or equal to d, where there is at least one bike and a station at a distance less than or equal to d where there is at least one free slot.

In the analysis, we explore parameter $d \in [0,1]$ to see how the satisfaction of the property in each location changes with respect to the walking distance to another station. In Figure 18, we plot the approximate probability satisfaction p_{ϕ_1} for 1000 runs for all the stations, for $d = 0$, (b) $d = 0.2$, (c) $d = 0.3$ and (d) $d = 0.5$ (expressed in kilometres). We can see that for (a) $d = 0$ many stations have a high probability to be full or empty (red points). Already with $d = 0.2$ km, i.e. walking 200 metres from the station with no bikes or slots, we greatly increase the probability of ϕ_1 and that, at $d = 0.5$, p_{ϕ_1} is larger than 0.5 for all the stations.

The behaviour "if I wait t minutes, I will always find a bike and a free slot" can instead be captured by the following property:

$$\phi_2 = \mathscr{G}_{[0,T_{end}]}\{\mathscr{F}_{[0,t]}((B > 0) \wedge (S > 0))\} \tag{7}$$

A station ℓ satisfies ϕ_2 if and only if it is always true, for each time $h \in [0,T_{end}]$, that eventually, in a time between h and $h+t$, there will be a bike and a free slot.

In the analysis, we explore parameter $t \in [0,10]$ minutes to see how the satisfaction of the property in each location changes with respect to the waiting time for a bike or a free slot. In Figure 19, we plot the approximate

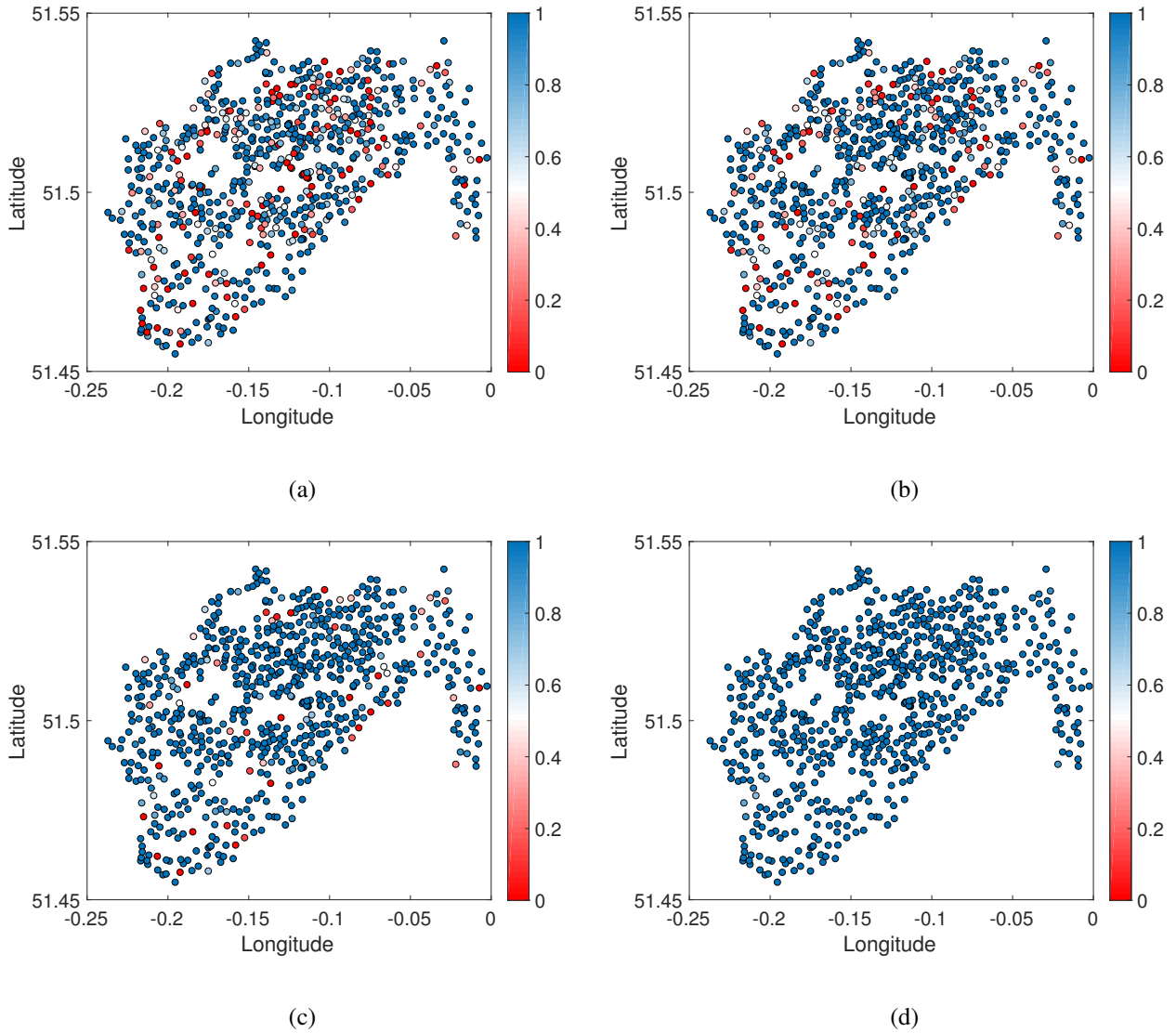
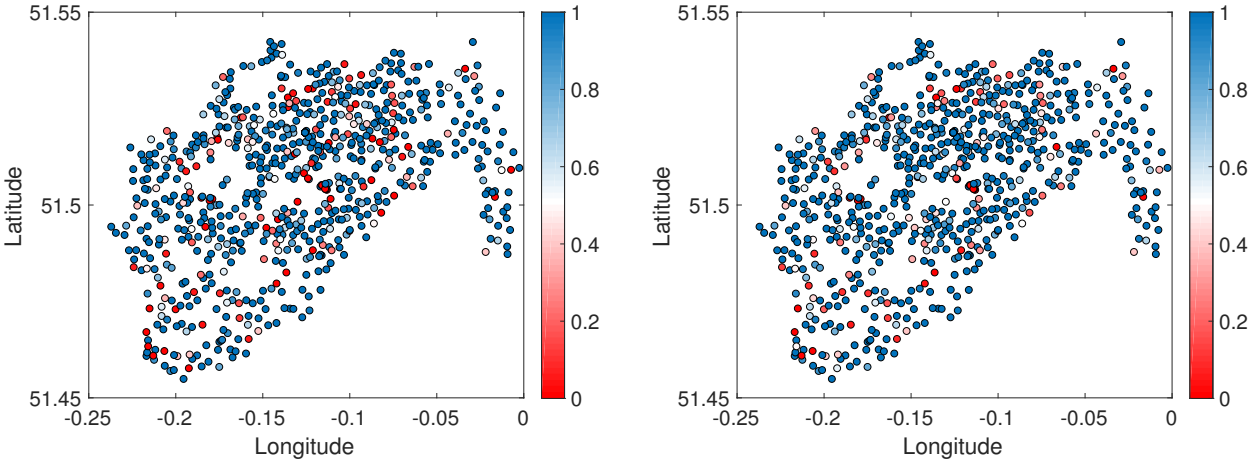


Figure 18: Approximate probability satisfaction degree of formula ϕ_1 for 1000 runs for each BSS station for (a) $d = 0$, (b) $d = 0.2$, (c) $d = 0.3$ and (d) $d = 0.5$. The value of the degree is given by the colour legend.

probability satisfaction p_{ϕ_1} for 1000 runs for all the stations, for (a) $t = 0$ and (b) $t = 10$. In this case, we can see that waiting a certain time does not greatly increase the probability to always find a bike or a free slot; even after 10 minutes we can see in Figure 19 (b) that there are still many stations with satisfaction probability close to zero. This means that there are stations that remain for more than 10 minutes always full or empty. Considering that the average human walking speed is about 5.0 kilometres per hour (km/h), we can conclude that if a user does not find a bike or a free slot in a station, she has usually more probability to find the bike/free slot in another stations than waiting in the same station.

6.1.3 Model reduction using ERODE

In this section we use ERODE to identify symmetries among the dynamics between different bike stations, potentially leading to: (i) more insights into the nature of the studied model; and (ii) a more compact representation.



(a) (b)

Figure 19: Approximate probability satisfaction degree of formula ϕ_2 for 1000 runs for each BSS station for (a) $t = 0$ and (b) $t = 10$. The value of the degree is given by the color legend.

The model. As discussed in Section 4, *ERODE* is designed for time-homogeneous systems. For this reason, we consider a variant of the model from Section 6.1.1 where we fix the parameters of the first time interval. In other words, we consider a model of the form:

$$\begin{array}{lll} B_i \to S_i & \text{at rate } out_i(0), & \forall i \in (1,N) \\ S_i \to B_i & \text{at rate } in_i(0), & \forall j \in (1,N) \\ B_i \to S_i + T_j^i & \text{at rate } out_j^i(0), & \forall i,j \in (1,N) \\ S_i + T_j^i \to B_i & \text{at rate } in_j^i(\#T_j^i), & \forall i,j \in (1,N) \end{array}$$

We remark that this is not a real limitation, as it would just suffice to apply our techniques to the three distinct models obtained in the three considered time intervals. Furthermore, we consider an ODE interpretation of the model as a standard first-order approximation of the Markov population process (e.g., [11]). For example, the reaction

$$B_i \to S_i \text{ at rate } out_i(0)$$

fires at speed $out_i(0)$. This term appears in the ODE of B_i (with negative sign) as well as in the ODE of S_i. Therefore, the obtained ODE system consists of 57,713 ODE variables.

Exact reduction. We applied BDE to the obtained ODE system. Since this captures ODE solutions that are equal at all time points, it required us to pre-partition the variables according to their initial conditions, taken from the model analysed in Section 6.1.1. This results in 49 blocks of variables. The coarsest BDE refining such initial partition has 32,037 blocks, corresponding to a reduction of about 45% in the number of ODE variables.

By studying the obtained BDE partition it turns out that this is only an "artificial" reduction due to the assumption of time-homogeneity. Indeed, a number of rates were set to 0 in the first time interval where the model is assumed to be homogeneous. (The rates have positive values only in the later intervals.) This led to many variables without dynamics: in particular we obtained a singleton block per variable with non-zero dynamics and a large block of 25,749 variables without dynamics. Interestingly, part of the ODEs for variables T_j^i agents have no dynamics, while all variables corresponding to bikes (B_i) and slots (S_i) agents have dynamics. This means that a number of movements of bikes between pairs of stations are not taking place during the first time interval.

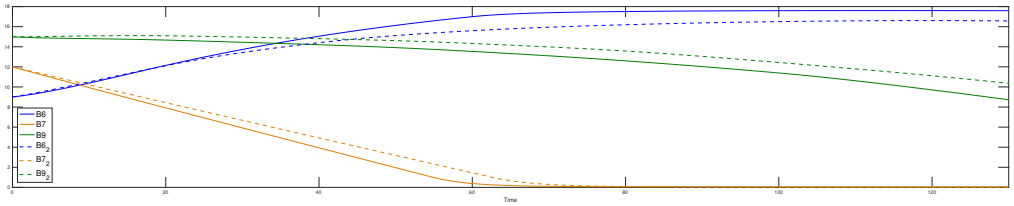
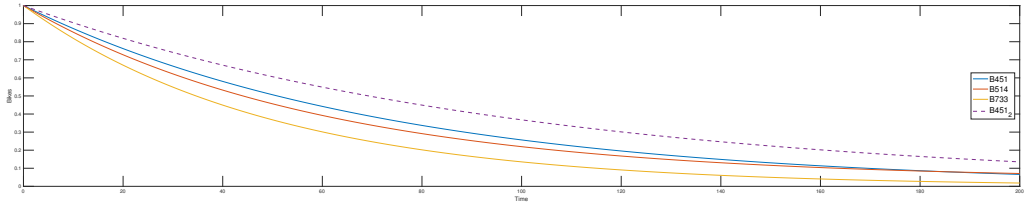
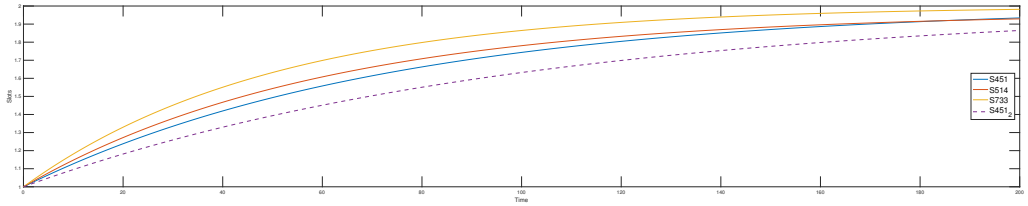


Figure 20: Bikes in three stations of the original model (solid lines) and in the perturbed one (dashed lines).



(a) Bikes in stations 451, 514, and 733.



(b) Free slots in stations 451, 514, and 733.

Figure 21: Trajectories relative to three bikes stations of the original model (solid lines), together with those of the corresponding BDE block in the reduced truncated model (dashed lines).

Approximate reduction. We use this model to motivate ongoing work on approximate notions of reductions, which can still be experimented with using the theory and technology developed so far. One well known problem of exact model reductions is the sensitivity to the rate parameters. For instance, in our model we have

$$out_{451}(0) = 0.00759924 \qquad out_{514}(0) = 0.00759848$$

A possible solution would be to develop approximate variants where the equivalence conditions are considered up to a given threshold on the parameters. This would be backed, at least for asymptotically small perturbations, by the well-known Gronwall inequality which guarantees nearby solutions. Then, it might be interesting to investigate if it is possible to recast an approximate reduction technique as an exact one applied to a slightly perturbed, more symmetric, model. An immediate way of obtaining such perturbation consists in considering a *truncated* version of the model, obtained by rounding the rates to a given significant decimal. Here we round all parameters up to two decimals, obtaining, e.g.:

$$out_{451}(0) = 0.01 \qquad out_{514}(0) = 0.01$$

Clearly, such truncation has consequences for the system dynamics. However we observed that it preserved the general trends. This is confirmed by Figure 20, which compares the evolution in time of the number of bikes in three representative stations (namely, B_6, B_7, and B_9) in the original model and in the perturbed one.

The perturbed model now allows for interesting BDE aggregations. The largest BDE has 14,835 blocks only (thus leading to a reduced ODE system with 14,835 variables). Instead, less aggressive perturbations (e.g., rounding up to the third decimal) did not lead to any additional reduction with respect to the original model. The following are two examples of BDE blocks obtained when rounding up to the second decimal:

$$\{B_{451}, B_{514}, B_{733}\} \qquad \{S_{451}, S_{451}, S_{733}\}$$

These two BDE blocks tell us that the three stations 451, 514 and 733 always have same number of bikes (and hence of free slots) in the perturbed model. Instead, this was not the case in the original model. However, Figure 21 shows that the introduced error is limited in this case. Furthermore, it shows that the model is still physically meaningful, in the sense that the sum of bikes and free slots in each station remained unchanged across all time points (and is equal to 1).

This discussion has shown that the approach is promising. Indeed, by resorting to just simple rounding operations, BDE has been able to identify variables with similar, but not identical, dynamics. This paves the way to more sophisticated approaches, e.g., aiming to obtain the smallest perturbation that enables a significant reduction, which are the subject of ongoing investigations.

6.2 Bus transportation

We now proceed with an application to bus transportation systems. We show an example of use of *topochecker* identifying diverted bus positions based on GPS data and the detection of bus clumping or platooning in frequent bus services. Bus clumping, and, more in general, issues related to headway and slack time of buses, are the subject of an active research field; see for instance [34, 33, 25, 24, 47] and references therein. Promising self-regulating strategies have been proposed such as [34, 33, 25], by letting buses wait at pre-defined control points in order to maintain a constant headway between subsequent buses. Such strategies have shown to be resilient to several kinds of perturbations in the system such as buses breaking, change in traffic conditions and temporary re-routing of buses, as long as these perturbations are not too severe. Such strategies depend critically on information on the position of buses at specific times and on the correctness of GPS data.

6.2.1 Identifying Diverted Bus Positions

In [18, 17], *topochecker* has been successfully applied to detect problems in the automatic vehicle location (AVL) data, which is the input to other systems that provide information to passengers and system operators such as bus arrival predictions. Such data may contain errors originating in a problem with the hardware of the measurement device or also indicate operational problems experienced by bus drivers that encountered unexpected road works or accidents and have to deviate from their planned route. The GPS data of the position of the buses can be automatically projected onto a portion of the digital map (such as for example provided by OpenStreetMap[8]) of the area where the bus is operating, after which spatial model checking can be applied on an image of the map to identify those bus positions that may indicate a problem. An example of buses apparently being diverted or in an off-road position is shown in Figure 23. The following formulas specify a diverted bus and a bus that is off-road. First, different types of streets are characterised by their colour on the map and defined as atomic propositions. For example, small streets are white, the main street is pink, etc. Also a bus is specified by a particular colour. Then a diverted bus can be specified by a bus surrounded by the colour of the `smallStreet`, meaning that the bus is on a small street. A bus that is off-road is not on a street, and thus not surrounded by any colour that is associated to streets, be they main streets or small streets in this particular case. An excerpt from the *topochecker* specification is shown in Figure 22.

Being able to detect and identify errors in AVL data, associating them with the right category of error in an automatic way, is of great use in obtaining better prediction algorithms. More complicated examples are those related to operational errors. For example one can detect buses of the same route that have overtaken each other or that are approaching the same bus stop within a too short interval of time.

6.2.2 Bus Clumping in Frequent Bus Services

The next example concerns the analysis of some specific problems in modern public transport systems such as the so-called “frequent” bus services operating in many densely populated cities [18, 17]. Frequent bus services are public bus services without a published timetable but with regular and frequent buses operating along pre-established routes. In such services a particular phenomenon may occur that is commonly known

[8] https://www.openstreetmap.org.

```
Let smallStreet = [red = 255] & [green = 255] & [blue = 255];
Let mainStreet = ... ;

Let bus = [red = 0] & [blue = 255];

Let street = smallStreet | mainStreet;

Let divertedBus = bus S smallStreet;

Let busOutOfStreet = bus & (! (bus S street));

Check "0xFF0000" divertedBus;     // Colours in red the points
                                  // satisfying divertedBus
Check "0xFF00FF" busOutOfStreet; // Colours in magenta the points
                                  // satisfying busOutOfStreet
```

Figure 22: Spatial formulas to detect diverted buses

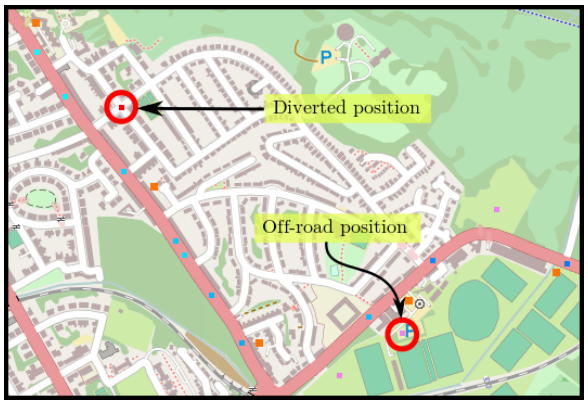


Figure 23: Diverted positions (neither off road, nor on a main street) are automatically highlighted in red by the spatial model checking procedure, off-road positions are highlighted in violet.

Figure 24: Because of delays caused by boarding passengers the headway between buses is successively eroded over time until the buses are essentially 'clumped' together. The successive time frames run from top to bottom.

as *bus bunching* or *bus clumping*. Bus clumping occurs where one bus catches up with – or at least comes too close to – the bus which is in front of it. In the absence of a published timetable for frequent services the important performance metric to consider is not timetable adherence but headway, a measure of the separation between subsequent buses. This separation can be defined both in terms of distance between buses on the same route or in terms of the time between two buses on the same route passing by the same bus stop. It is possible to identify these two notions of clumping using STLCS on a time series of street map images on which the bus positions are projected. The problem is illustrated in Figure 24 showing three successive "satellite view" images of the position of three different buses on the same route projected on a portion of a map of a city. The buses are shown as three small red dots on the main road. The distance between the buses is getting smaller in each successive image, resulting in the three buses being lined up in the final image as indicated by the red arrows (Figure 24 left-bottom).

In the sequel, we focus on the spatio-temporal characterisation of clumping. Consider a single bus route, served by k buses. At each instant of time, the state of the system is completely described by a tuple of k GPS positions; therefore, a system trace is a finite sequence of such tuples. We can distinguish two different variants of clumping that differ in a subtle way. One in which two consecutive buses serving the same route are *spatially* close to each other, and one in which they pass by the same bus stop within a too short amount of time. Here we formalise the latter variant considering three buses on the same route. The input code of the model checking session is shown in Figure 25. Formulas `bus1`, `bus2`, `bus3`, and `busStop` serve the purpose of identifying bus positions on a digital map of the city. In this example, colours are used to distinguish the different buses serving the same route, so that each bus has a specific colour. Similarly, formula `busStop`

```
Let bus1 = [red = 155] & [green = 0] & [blue = 0];
Let bus2 = [red = 188] & [green = 0] & [blue = 0];
Let bus3 = [red = 221] & [green = 0] & [blue = 0];
Let bus = bus1 | bus2 | bus3;
Let busStop = [red = 55] & [green = 55] & [blue = 255];

Let close(x) = N^7 x;

Let busAtStop(x) = busStop & close(x);

Let busAfterBus1 = busAtStop(bus1) &
     EX busAtStop(bus2 | bus3);
Let busAfterBus2 = busAtStop(bus2) &
     EX busAtStop(bus1 | bus3);
Let busAfterBus3 = busAtStop(bus3) &
     EX busAtStop(bus1 | bus2);

Let timeConglomerate = (busAfterBus1 | busAfterBus2 | busAfterBus3);

Check "0xFF0000" timeConglomerate; // Colours in red the points
                                   // satisfying timeConglomerate
```

Figure 25: Spatio-temporal formulas for time conglomerates

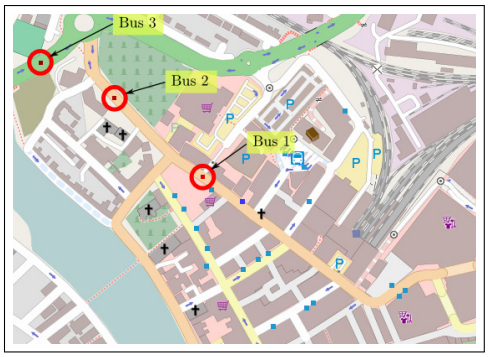
identifies the position of a bus stop. The formula `timeConglomerate`, that we explain below, is true at points of a *bus stop* whenever clumping is happening (formation of a conglomerate of buses) at that particular stop.

A *spatio-temporal* conglomerate happens when two buses serving the same route pass by the same stop within a short amount of time. This case is subtler than the spatial one, as it does not necessarily imply that the headway between two buses becomes too small. This event is described by the formula `timeConglomerate`, which features a combination of spatial operators (used to detect that a bus is close to a stop) and temporal operators (used to identify the spatio-temporal conglomerate). For instance, consider the formula `busAfterBus1`. This formula is true on points that are: i) part of a bus stop, and close to `bus1`, because `busAtStop` must be true for `bus1`; ii) such that, in the next snapshot[9], these will be part of a bus stop, and close to either `bus2` or `bus3`. Note that the use of spatial and temporal connectives in the same formula permits one to refer to the colour of points at a specific time, and at subsequent time instants.

Figure 26 is obtained from the spatio-temporal model checker, starting from the positions of three buses serving the same route. Figures 26a-26e are obtained by mapping bus coordinates over a base map. Buses are represented by small squares of different shades of red on the roads. To make them more visible they are also highlighted by the red circles in Figure 26b. The small dark blue square is a bus stop (see Figure 26c). Figure 26f shows the output of the model checker when checking the formula `EF timeConglomerate` in the initial state shown in Figure 26a. Indeed, Figure 26f is the same as Figure 26a, except for the colour of the bus stop, whose points are now turned green by the model checker, indicating that clumping happens at that stop, at some point in the future.

[9]More than one time step can be required. This can be achieved by repeated nesting of the `EX` operator. We did not do so for the sake of clarity in Figure 26.

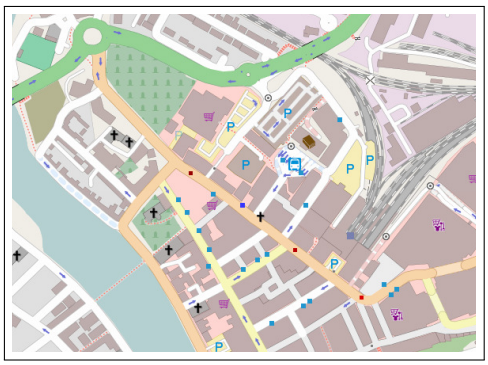
(a) Initial state



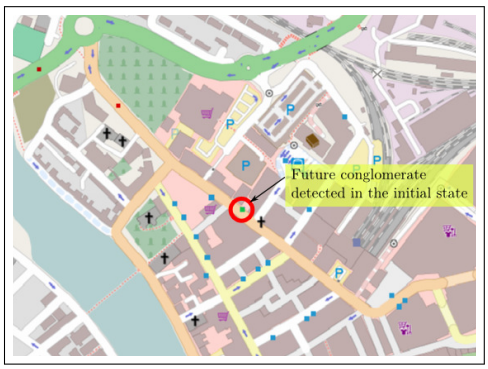
(b) Second state

(c) Bus 1 passes by the stop

(d) Bus 2 passes by the stop



(e) Final state



(f) Result from the model-checker. Points of the initial state that will be involved in a future conglomerate are coloured in green and indicated by the red circle

Figure 26: Spatio-temporal conglomerate.

7 Conclusion

Summary. We presented the QUANTICOL software tool suite, a collection of mature tools for the analysis of collective adaptive systems. Taken together, our software supports many notable phases of a typical modelling workflow, such as specification (by means of domain-specific languages), model checking, and model reduction. Each tool has been developed having in mind important principles of software engineering such as robustness, usability, re-usability, and scalability. This has led, for instance, to ease of interoperability, as witnessed by the integration efforts with the CARMA Eclipse plug-in which are documented in D4.3.

In the wider perspective of the whole project, we believe that this deliverable has demonstrated a successful synergy between the theories developed in the other work packages and their practical repercussions in terms of software implementation and applications to realistic models. The combination of theory and practice produces a virtuous self-reinforcing circle whereby the theory can impact on real systems, which in turn solicit further theoretical results to tackle problems of increasing complexity.

Deviations from plans. Overall, the work carried out in WP5 has been in line with what had been planned in the Description of Work. This deliverable is no exception since its intent was to showcase a CAS toolkit and apply it to our case studies. However, we did register a more pronounced interest in case studies concerning bus transportation systems and bike sharing systems than smart grids. This is consistent with a similar situation observed in D1.4 and can be attributed to the same fact, namely one principal investigator (Nicolas Gast) moving to INRIA from EPFL, where most of the expertise on smart grids resided. On the other hand, it is fair to say that the amount of research on buses and bike sharing exceeded our expectations.

Future work. Although our tools — and the theory behind them — have been motivated by the smart city scenarios of the QUANTICOL project, their applicability has been demonstrated to go beyond: *FlyFast* has analyzed network protocols for gossiping, *jSTTL* can be used for studying epidemiological systems, *ERODE* finds applications in systems biology, and *topochecker* is being experimented with for medical imaging. We believe that such a diversity of application areas adds significant value to our contributions and serves as a motivation to pursue new avenues of research in the future.

FlyFast will be used for further case studies, among which a continuation of the work on gossip models, and will remain publicly available and supported as part of the jSAM framework. Recent work on a front-end language for predicate-based communication will be further consolidated (see Deliverables 3.3 and 4.3).

jSTTL is still under development and new features will be integrated. For instance, we are working on new kinds of formats and sources for input and output of data, and on a better integration with Matlab. We are also developing better monitoring algorithms and extending the logic with new operators, implementing them in the tool. In particular, we are developing an online monitoring procedure where trajectories are collected from external data sources, and a semantics to incorporate uncertainty in measurements.

We plan to continue supporting *ERODE* for the foreseeable future, with the implementation of techniques for approximate model reduction (discussed in Deliverable 5.3), the consolidation of prototypes supporting recent research papers [13, 48], and the extension to other classes of dynamical systems such as differential-algebraic equations, which is the subject of current research.

The spatio-temporal model checker topochecker is currently being extended with further operators concerning collective properties on sets of points in space. Furthermore, the work on medical imaging using topochecker [6] has been extended and is currently under review for publication in an international scientific journal (available also as Quanticol Technical Report TR-QC-02-2017). Also a clinical trial is in progress at the University Hospital of Siena and an investigation is being carried on to explore how the research results can be further exploited after the end of the project.

References

[1] Apache Commons Mathematics Library, `http://commons.apache.org/proper/commons-math/`.

[2] R. Bakhshi. *Gossiping Models – Formal Analysis of Epidemic Protocols*. PhD thesis, Vrije Universiteit Amsterdam, January 2011.

[3] R. Bakhshi, L. Cloth, W. Fokkink, and B. R. Haverkort. Mean-field framework for performance evaluation of push-pull gossip protocols. *Perform. Eval.*, 68(2):157–179, 2011.

[4] E. Bartocci, L. Bortolussi, D. Milios, L. Nenzi, and G. Sanguinetti. Studying Emergent Behaviours in Morphogenesis Using Signal Spatio-Temporal Logic. In *Hybrid Systems Biology*, number 9271 in Lecture Notes in Computer Science, pages 156–172. Springer International Publishing, Sept. 2015. DOI: 10.1007/978-3-319-26916-0_9.

[5] E. Bartocci, L. Bortolussi, L. Nenzi, and G. Sanguinetti. System design of stochastic models using robustness of temporal properties. *Theoretical Computer Science*, 587:3–25, July 2015.

[6] G. Belmonte, V. Ciancia, D. Latella, and M. Massink. From collective adaptive systems to human centric computation and back: Spatial model checking for medical imaging. In M. H. ter Beek and M. Loreti, editors, *Proceedings of the Workshop on FORmal methods for the quantitative Evaluation of Collective Adaptive SysTems, FORECAST@STAF 2016, Vienna, Austria, 8 July 2016.*, volume 217 of *EPTCS*, pages 81–92, 2016.

[7] E. Bertuzzo, S. Azaele, A. Maritan, M. Gatto, I. Rodriguez-Iturbe, and A. Rinaldo. On the space-time evolution of a cholera epidemic. *Water Resources Research*, 44(1):W01424, 2008.

[8] M. L. Blinov, J. R. Faeder, B. Goldstein, and W. S. Hlavacek. Bionetgen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, 2004.

[9] L. Bortolussi, V. Galpin, and J. Hillston. Hybrid performance modelling of opportunistic networks. In *Proceedings 10th Workshop on Quantitative Aspects of Programming Languages and Systems, QAPL 2012, Tallinn, Estonia, 31 March and 1 April 2012.*, pages 106–121, 2012.

[10] L. Bortolussi and N. Gast. Mean field approximation of uncertain stochastic models. In *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016*, pages 287–298, 2016.

[11] L. Bortolussi, J. Hillston, D. Latella, and M. Massink. Continuous approximation of collective system behaviour: A tutorial. *Performance Evaluation*, 70(5):317–349, 2013.

[12] L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin. Forward and backward bisimulations for chemical reaction networks. In *CONCUR*, 2015.

[13] L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin. Comparing chemical reaction networks: A categorical and algorithmic perspective. In *LICS*, 2016.

[14] L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin. Efficient syntax-driven lumping of differential equations. In *TACAS*, volume 9636, pages 93–111, 2016.

[15] L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin. Symbolic computation of differential equivalences. In *POPL*, 2016.

[16] L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin. ERODE: A tool for the evaluation and reduction of ordinary differential equations. In *TACAS*. Springer, 2017. To appear.

[17] V. Ciancia, S. Gilmore, G. Grilletti, D. Latella, M. Loreti, and M. Massink. On Spatio-temporal model-checking of vehicular movement in public transport systems - Preliminary Version. Quanticol Technical Report TR-QC-02-2016, QUANTICOL, 2016.

[18] V. Ciancia, S. Gilmore, D. Latella, M. Loreti, and M. Massink. Data verification for collective adaptive systems: Spatial model-checking of vehicle location data. In *SASO Workshops*, pages 32–37. IEEE Computer Society, 2014.

[19] V. Ciancia, G. Grilletti, D. Latella, M. Loreti, and M. Massink. An experimental spatio-temporal model checker. In *Software Engineering and Formal Methods - SEFM 2015 Collocated Workshops: ATSE, HOFM, MoKMaSD, and VERY*SCART, York, UK, September 7-8, 2015, Revised Selected Papers*, pages 297–311, 2015.

[20] V. Ciancia, D. Latella, M. Loreti, and M. Massink. Specifying and verifying properties of space. In *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, volume 8705 of *Lecture Notes in Computer Science*, pages 222–235. Springer, 2014.

[21] V. Ciancia, D. Latella, M. Loreti, and M. Massink. Model Checking Spatial Logics for Closure Spaces. *Logical Methods in Computer Science*, Volume 12, Issue 4, Oct. 2016.

[22] V. Ciancia, D. Latella, M. Massink, and R. Pakauskas. Exploring spatio-temporal properties of bike-sharing systems. In *2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASO Workshops*, pages 74–79. IEEE Computer Society, 2015.

[23] V. Ciancia, D. Latella, M. Massink, R. Paskauskas, and A. Vandin. A tool-chain for statistical spatio-temporal model checking of bike sharing systems. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I*, volume 9952 of *Lecture Notes in Computer Science*, pages 657–673, 2016.

[24] C. F. Daganzo. A headway-based approach to eliminate bus bunching: Systematic analysis and comparisons. *Transportation Research Part B: Methodological*, 43(10):913 – 921, 2009.

[25] C. F. Daganzo and J. Pilachowski. Reducing bunching with bus-to-bus cooperation. *Transportation Research Part B: Methodological*, 45(1):267 – 277, 2011.

[26] L. De Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.

[27] S. Derisavi, H. Hermanns, and W. H. Sanders. Optimal state-space lumping in Markov chains. *Inf. Process. Lett.*, 87(6):309–315, 2003.

[28] A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *Proc. of FORMATS 2010, the 8th International Conference on Formal Modeling and Analysis of Timed Systems, Klosterneuburg, Austria, September 8–10*, volume 6246, pages 92–106, 2010.

[29] F. Erhard, C. C. Friedel, and R. Zimmer. FERN - a Java framework for stochastic simulation and evaluation of reaction networks. *BMC Bioinformatics*, 9(1):356, 2008.

[30] C. Feng, J. Hillston, and D. Reijsbergen. Moment-based probabilistic prediction of bike availability for bike-sharing systems. pages 139–155. SPRINGER INT PUBLISHING AG, 2016.

[31] Microsoft GEC, `http://research.microsoft.com/en-us/projects/gec/`.

[32] R. A. Hayden and J. T. Bradley. A fluid analysis framework for a Markovian process algebra. *Theor. Comput. Sci.*, 411(22-24):2260–2297, 2010.

[33] J. J. B. III, R. J. Clark, D. W. Williamson, D. D. Eisenstein, and L. K. Platzman. Building a self-organizing urban bus route. In *Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2012, Lyon, France, September 10-14, 2012*, pages 66–70. IEEE Computer Society, 2012.

[34] J. J. B. III and D. D. Eisenstein. A self-coordinating bus route to resist bus bunching. *Transportation Research Part B: Methodological*, 46(4):481 – 491, 2012.

[35] D. Jovanovic and L. M. de Moura. Solving non-linear arithmetic. In *IJCAR*, pages 339–354, 2012.

[36] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[37] 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 - 8th International Symposium, TGC 2013*, volume 8358 of *LNCS*, pages 297–314. Springer, 2013.

[38] D. Latella, M. Loreti, and M. Massink. On-the-fly PCTL fast mean-field approximated model-checking for self-organising coordination. *Sci. Comput. Program.*, 110:23–50, 2015.

[39] D. Latella, M. Loreti, and M. Massink. FlyFast: A Mean Field Model Checker. In A. Legay and T. Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS. Springer, 2017. ISSN 0302-9743, (To appear) .

[40] J. Le Boudec, D. D. McDonald, and J. Mundinger. A generic mean field convergence result for systems of interacting objects. In *Fourth International Conference on the Quantitative Evaluaiton of Systems (QEST 2007), 17-19 September 2007, Edinburgh, Scotland, UK*, pages 3–18. IEEE Computer Society, 2007.

[41] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Proc. of Joint International Conferences on Formal Modeling and Analysis of Timed Systmes, FORMATS 2004, and Formal Techniques in Real-Time and Fault -Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24*, volume 3253 of *Lecture Notes in Computer Science*, pages 152–166, 2004.

[42] L. Mari, E. Bertuzzo, L. Righetto, R. Casagrandi, M. Gatto, I. Rodriguez-Iturbe, and A. Rinaldo. Modelling cholera epidemics: the role of waterways, human mobility and sanitation. *Journal of The Royal Society Interface*, 9(67):376–388, Feb. 2012.

[43] L. Nenzi and L. Bortolussi. Specifying and monitoring properties of stochastic spatio-temporal systems in signal temporal logic. In *8th International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS 2014, Bratislava, Slovakia, December 9-11, 2014*, 2014.

[44] L. Nenzi, L. Bortolussi, V. Ciancia, M. Loreti, and M. Massink. Qualitative and Quantitative Monitoring of Spatio-Temporal Properties. In E. Bartocci and R. Majumdar, editors, *Runtime Verification*, number 9333 in Lecture Notes in Computer Science, pages 21–37. Springer International Publishing, 2015.

[45] L. Nenzi, L. Bortolussi, and M. Loreti. jSSTL - a tool to monitor spatio-temporal properties. In *VALUETOOLS*, 2016.

[46] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[47] M. Ruan and J. Lin. An investigation of bus headway regularity and service performance in Chicago bus transit system. In *Transport Chicago, Annual Conference, 14.*, 2009.

[48] S. Tognazzi, M. Tribastone, M. Tschaikowski, and A. Vandin. EGAC: A genetic algorithm to compare chemical reaction networks. In *GECCO*, 2017. To appear.

[49] M. Tribastone, S. Gilmore, and J. Hillston. Scalable differential analysis of process algebra models. *IEEE Trans. Software Eng.*, 38(1):205–219, 2012.

[50] A. Valmari and G. Franceschinis. Simple $o(m \log n)$ time Markov chain lumping. In *TACAS*, 2010.

[51] E. O. Voit. Biochemical systems theory: A review. *ISRN Biomathematics*, 2013:53, 2013.