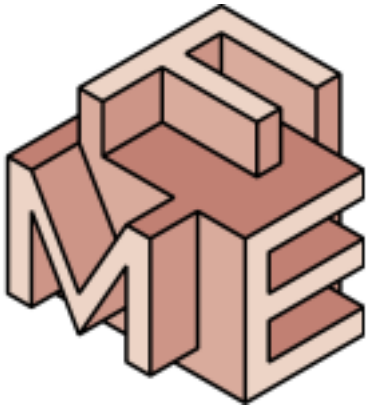
10th Workshop on Quantitative Aspects of Programming Languages

QAPL 2012

Tallinn, Estonia, 31 March–1 April 2012

Pre-Proceedings



10th Workshop on Quantitative Aspects of Programming Languages

QAPL 2012

Tallinn, Estonia, 31 March–1 April 2012

Pre-Proceedings

Institute of Cybernetics at Tallinn University of Technology

Tallinn ◦ 2012

10th Workshop on Quantitative Aspects of Programming Languages
QAPL 2012
Tallinn, Estonia, 31 March–1 April 2012
Pre-Proceedings

Edited by Mieke Massink and Herbert Wiklicky

Institute of Cybernetics at Tallinn University of Technology
Akadeemia tee 21, 12618 Tallinn, Estonia
http://www.ioc.ee/

The final proceedings of QAPL 2012 will appear in *Electron. Proc. in Theor. Comput. Sci.* (EPTCS).

Sponsored by Formal Methods Europe

Preface

This volume contains the preliminary proceedings of the Tenth Workshop on Quantitative Aspects of Programming Languages (QAPL 2012), held in Tallinn, Estonia, on March 31–April 1, 2012. QAPL 2012 is a satellite event of the European Joint Conferences on Theory and Practice of Software (ETAPS 2012).

The central theme of the workshop is that of quantitative aspects of computation. These aspects are related to the use of physical quantities (storage space, time, bandwidth, etc.) as well as mathematical quantities (e.g. probability and measures for reliability, security and trust), and play an important (sometimes essential) role in characterising the behavior and determining the properties of systems. Such quantities are central to the definition of both the model of systems (architecture, language design, semantics) and the methodologies and tools for the analysis and verification of the systems properties. The aim of this workshop is to discuss the explicit use of quantitative information such as time and probabilities either directly in the model or as a tool for the analysis of systems.

In particular, the workshop focuses on:

- the design of probabilistic, real-time and quantum languages, and the definition of semantical models for such languages;
- the discussion of methodologies for the analysis of probabilistic and timing properties (e.g. security, safety, schedulability) and of other quantifiable properties such as reliability (for hardware components), trustworthiness (in information security) and resource usage (e.g. worst-case memory/stack/cache requirements);
- the probabilistic analysis of systems which do not explicitly incorporate quantitative aspects (e.g. performance, reliability and risk analysis);
- applications to safety-critical systems, communication protocols, control systems, asynchronous hardware, and to any other domain involving quantitative issues.

The history of QAPL starts in 2001, when its first edition was held in Florence, Italy, as a satellite event of the ACM Principles, Logics, and Implementations of high-level programming languages, PLI 2001. The second edition, QAPL 2004, was held in Barcelona, Spain, as a satellite event of ETAPS 2004. Since then, QAPL has become a yearly appointment with ETAPS. In the following years, QAPL was held in Edinburgh, Scotland (QAPL 2005), Vienna, Austria (QAPL 2006), Braga, Portugal (QAPL 2007), Budapest, Hungary (QAPL 2008), York, UK (QAPL 2009), Paphos, Cyprus (QAPL 2010) and Saarbrücken, Germany (QAPL 2011). The proceedings of the workshops upto and including 2009 are published as volumes in Electronic Notes in Theoretical Computer Science (ENTCS). The editions of 2010 and 2011 are published as volume 28 and volume 57, respectively, of Electronic Proceedings in Theoretical Computer Science (EPTCS).

Three special issues of the journal of Theoretical Computer Science are dedicated to the QAPL 2004, QAPL 2006 and QAPL 2010 events, and are published in Volume 346(1), Volume 382(1) and Volume 413(1) respectively. A special issue of the journal of Theoretical Computer Science dedicated to QAPL 2011 and QAPL 2012 is planned.

Preliminary Report. Final version to appear in:
QAPL 2012

The Program Committee of QAPL 2012 was composed by:

Alessandro Aldini	*University of Urbino, Italy*
Christel Baier	*Technical University of Dresden, Germany*
Marco Bernardo	*University of Urbino, Italy*
Nathalie Bertrand	*INRIA Rennes Bretagne Atlantique, France*
Luca Bortolussi	*University of Trieste, Italy*
Jeremy Bradley	*Imperial College London, UK*
Tomás Brážd il	*Masaryk University, Czech Republic*
Antonio Cerone	*United Nations University - IIST, Macao*
Kostas Chatzikokolakis	*Eindhoven University of Technology, the Netherlands*
Josée Desharnais	*University of Laval, Canada*
Alessandra Di Pierro	*University of Verona, Italy*
Mieke Massink (Co-chair)	*Italian National Research Council - ISTI, Pisa, Italy*
Paulo Mateus	*Technical University of Lisbon, Portugal*
Annabelle McIver	*Macquarie University, Australia*
Gethin Norman	*University of Glasgow, UK*
David Parker	*Oxford University, UK*
Anne Remke	*University of Twente, the Netherlands*
Jeremy Sproston	*University of Turin, Italy*
Herbert Wiklicky (Co-chair)	*Imperial College London, UK*

The programme committee selected 8 regular papers and 4 presentation-only papers. All regular papers were reviewed by at least three reviewers. They are included in this volume. Authors of regular papers will be asked to submit a revised version for the post-proceedings to appear in the Electronic Proceedings in Theoretical Computer Science (EPTCS). The workshop programme includes three keynote presentations: Kim G. Larsen (Aalborg University, Denmark); Boris Köpf (IMDEA Software Institute, Madrid, Spain) and Jeremy Bradley (Imperial College London, U.K.). We would like to thank the QAPL steering committee for its support, Formal Methods Europe (FME) for their financial support for the speakers, and furthermore all the authors, the invited speakers, the programme committee and the external referees for their valuable contributions.

February 2012

Mieke Massink and Herbert Wiklicky
Program Co-chairs

Workshop Programme

Saturday, March 31

9:25 - 9:30 *Opening*

9:30 - 10:30
Invited speaker: Kim G. Larsen (Aalborg University, Denmark)
Statistical Model Checking for Priced Timed Automata

10:30 - 11:00 *Coffee*

Session: Model Checking

11:00 - 11:30
Sergio Giro
Efficient computation of exact solutions for quantitative model checking
11:30 - 12:00
Elise Cormie-Bowins and Franck Van Breugel
Measuring Progress of Probabilistic LTL Model Checking
12:00 - 12:30
Francesco Belardinelli, Pavel Gonzalez and Alessio Lomuscio
Automated Verification of Quantum Protocols using MCMAS

12:30 - 14:00 *Lunch*

Session: Fluid Flow and Stochastic Modelling

14:00 - 15:00
Invited speaker: Jeremy Bradley (Imperial College London, U.K.)
Mean field and fluid approaches to Markov chain analysis
15:00 - 15:30
Anna Kolesnichenko, Anne Remke, Pieter-Tjerk De Boer and Boudewijn Haverkort
A logic for model-checking of mean-field models (Presentation only)

15:30 - 16:00 *Coffee*

16:00 - 16:30
Mark Timmer, Joost-Pieter Katoen, Jaco Van De Pol and Mariëlle I. A. Stoelinga
Efficient Modelling and Generation of Markov Automata (Presentation only)
16:30 - 17:00
Luca Bortolussi and Jane Hillston
Towards Fluid Model Checking (Presentation only)

Sunday, April 1

9:30 - 10:30
Invited speaker: Boris Köpf (IMDEA Software Institute, Madrid, Spain)
Quantifying Side-Channels in RSA and AES

10:30 - 11:00 *Coffee*

Session: Security, Information Flow and Privacy

11:00 - 11:30
Ivan Gazeau, Dale Miller and Catuscia Palamidessi
A non-local method for robustness analysis of floating point programs
11:30 - 12:00
Hirotoshi Yasuoka and Tachio Terauchi
Quantitative Information Flow as Safety and Liveness Hyperproperties
12:00 - 12:30
Catuscia Palamidessi and Marco Stronati
Differential privacy for relational algebra: improving the sensitivity bounds via constraint systems

12:30 - 14:00 *Lunch*

Session: Hybrid and Time

14:00 - 14:30
Luca Bortolussi, Vashti Galpin and Jane Hillston
Hybrid performance modelling of opportunistic networks
14:30 - 15:00
Marco Bernardo
Weak Markovian Bisimulation Congruences and Exact CTMC-Level Aggregations for Concurrent Processes
15:00 - 15:30
Henri Hansen and Mark Timmer
Why Confluence Reduction is Better than Partial Order Reduction in Probabilistic and Non-Probabilistic Branching Time (Presentation only)

15:30 - 16:00 *Coffee*

16:00 - 16:30 *Closing*

UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata *

Peter Bulychev Alexandre David Kim Guldstrand Larsen
Marius Mikučionis Danny Bøgsted Poulsen
Department of Computer Science
Aalborg University, Denmark
{pbulychev,adavid,kgl,marius,dannybp}@cs.aau.dk

Axel Legay
INRIA Rennes, France
Department of Computer Science
Aalborg University, Denmark
alegay@irisa.fr

Zheng Wang
Shanghai Key Laboratory of Trustworthy Computing
Software Engineering Institute
East China Normal University, China

This paper offers a survey of UPPAAL-SMC, a the major extension of the real-time verification tool UPPAAL. UPPAAL-SMC allows for the efficient analysis of performance properties of networks of priced timed automata under a natural stochastic semantics. In particular, UPPAAL-SMC relies on a series of extensions of the statistical model checking approach generalized to handle real-time systems and estimate undecidable problems. UPPAAL-SMC comes together with a friendly user interface that allows a user to specify complex problems in an efficient manner as well as to get feedback in the form of probability distributions and compare probabilities to analyze performance aspects of systems. The focus of the survey is on the evolution of the tool – including modeling and specification formalisms as well as techniques applied – together with applications of the tool to case studies.

1 Introduction

Quantitative properties of stochastic systems are usually specified in logics that allow one to compare the measure of executions satisfying certain temporal properties with thresholds. The model checking problem for stochastic systems with respect to such logics is typically solved by a numerical approach [3, 13] that iteratively computes (or approximates) the exact measure of paths satisfying relevant sub-formulas; the algorithms themselves depend on the class of systems being analyzed as well as the logic used for specifying the properties.

Another approach to solve the model checking problem is to *simulate* the system for finitely many runs, and use *hypothesis testing* to infer whether the samples provide a *statistical* evidence for the satisfaction or violation of the specification [39]. The crux of this approach is that since sample runs of a stochastic system are drawn according to the distribution defined by the system, they can be used to get estimates of the probability measure on executions. Those techniques, also called *Statistical Model Checking techniques* (SMC) [25, 35, 39, 34], can be seen as a trade-off between testing and formal verification. In fact, SMC is very similar to Monte Carlo used in industry, but it relies on a formal model of the system. The core idea of SMC is to monitor a number of simulations of a system whose behaviors depend on a stochastic semantic. Then, one uses the results of statistics (e.g. sequential hypothesis testing or Monte Carlo) together with the simulations to get an overall estimate of the probability that the system

*The paper is supported by VKR Centre of Excellence – MT-LAB and the IDEA4CPS center established on a grant from Danish National Research Foundation

Preliminary Report. Final version to appear in:
QAPL 2012

will behave in some manner. While the idea resembles the one of classical Monte Carlo simulation, it is based on a formal semantic of systems that allows us to reason on very complex behavioral properties of systems (hence the terminology). This includes classical reachability property such as "can I reach such a state?", but also non trivial properties such as "can I reach this state x times in less than y units of time?". Of course, in contrast with an exhaustive approach, such a simulation-based solution does not guarantee a result with 100% confidence. However, it is possible to bound the probability of making an error. Simulation-based methods are known to be far less memory and time intensive than exhaustive ones, and are sometimes the only option [40, 26].

Statistical model checking is now widely accepted in various research areas such as software engineering, in particular for industrial applications [5, 32, 17], or even for solving problems originating from systems biology [16, 28]. There are several reasons for this success. First, SMC is very simple to understand, implement, and use. Second, it does not require extra modeling or specification effort, but simply an operational model of the system, that can be simulated and checked against state-based properties. Third, it allows us to verify properties [14, 15, 5] that cannot be expressed in classical temporal logics. Finally, SMC allows to approximate undecidable problems. This latter observation is crucial. Indeed most of emerging problems such as energy consumption are undecidable [23, 8] and can hence only be estimated. SMC has been applied to a wide range of problems that goes from embedded systems[14] and systems biology [14, 15] to more industrial applications [5].

In a series of recent works [21, 12, 20], we have investigated the problem of Statistical Model Checking for networks of Priced Timed Automata (PTA). PTAs are timed automata, whose clocks can evolve with different rates, while[1] being used with no restrictions in guards and invariants. In [20], we have proposed a natural stochastic semantic for such automata, which allows to perform statistical model checking. Our work has latter been implemented in UPPAAL-SMC, that is a stochastic and statistical model checking extension of UPPAAL. UPPAAL-SMC relies on a series of extensions of the statistical model checking approach generalized to handle real-time systems and estimate undecidable problems. UPPAAL-SMC comes together with a friendly user interface that allows a user to specify complex problems in an efficient manner as well as to get feedback in the form of probability distributions and compare probabilities to Analise performance aspects of systems.

The objective of this paper is to offer a survey of UPPAAL-SMC. This includes modeling and specification formalism as well as techniques applied – together with applications of the tool to case studies.

Structure of the paper In Section 2, we introduce the formalism of networks of Priced timed automata. Section 3 overviews some existing statistical model checking algorithm, while Sections 4 and 5 introduce the GUI and give some details on the engine of UPPAAL-SMC. Finally, Section 6 presents a series of applications for the tool-set and Section 7 concludes the paper.

2 Modeling Formalism

The new engine of UPPAAL-SMC [21] supports the analysis of Priced Timed Automata (PTAs) that are timed automata whose clocks can evolve with different rates in different locations. In fact, the expressive power (up to timed bisimilarity) of NPTA equals that of general linear hybrid automata (LHA) [1], rendering most problems – including that of reachability – undecidable. We also assume PTAs are input-enabled, deterministic (with a probability measure defined on the sets of successors), and non-zeno.

[1] in contrast to the usual restriction of priced timed automata [6, 2]

PTAs communicate via broadcast channels and shared variables to generate Networks of Price Timed Automata (NPTA).

Fig. 1 provides an NPTA with three components A, B, and T as specified using the UPPAAL GUI. One can easily see that the composite system $(A|B|T)$ has the transition sequence:

$$\big((A_0,B_o,T_0),[x=0,y=0,C=0]\big) \xrightarrow{1} \xrightarrow{a!}$$
$$\big((A_1,B_0,T_1),[x=1,y=1,C=4]\big) \xrightarrow{1} \xrightarrow{b!}$$
$$\big((A_1,B_1,T_2),[x=2,y=2,C=6]\big),$$

demonstrating that the final location T_3 of T is reachable. In fact, location T_3 is reachable within cost 0 to 6 and within total time 0 and 2 in $(A|B|T)$ depending on when (and in which order) A and B choose to perform the output actions $a!$ and $b!$. Assuming that the choice of these time-delays is governed by probability distributions, a measure on sets of runs of NPTAs is induced, according to which quantitative properties such as *"the probability of T_3 being reached within a total cost-bound of 4.3"* become well-defined.

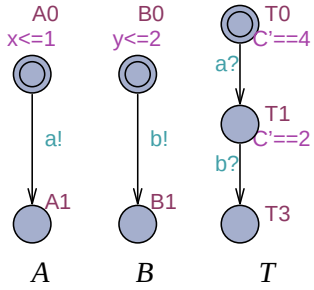


Figure 1: An NPTA, $(A|B|T)$.

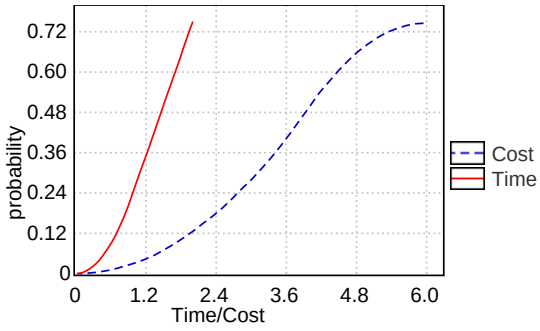


Figure 2: Cumulative probabilities for `time` and `Cost`-bounded reachability of T_3.

In our early works [20], we provide a natural stochastic semantics, where PTA components associate probability distributions to both the time-delays spend in a given state as well as to the transition between states. In UPPAAL-SMC uniform distributions are applied for bounded delays and exponential distributions for the case where a component can remain indefinitely in a state. In a network of PTAs the components repeatedly race against each other, i.e. they independently and stochastically decide on their own how much to delay before outputting, with the "winner" being the component that chooses the minimum delay. For instance, in the NPTA of Fig. 1, A wins the initial race over B with probability 0.75.

As observed in [20], though the stochastic semantic of each individual PTA in UPPAAL-SMC is rather simple (but quite realistic), arbitrarily complex stochastic behavior can be obtained by their composition when mixing individual distributions through message passing. The beauty of our model is that these distributions are naturally and automatically defined by the network of PTAs.

The Hammer Game To illustrate the stochastic semantics further consider the network of two priced timed automata in Fig. 3 modeling a competition between the two players Axel and Alex both having to hammer three nails down. As can be seen by the representing `Work`-locations the time (-interval) and rate of energy-consumption required for hammering a nail depends on the player and the nail-number. As expected Axel is initially quite fast and uses a lot of energy but becomes slow towards the last nail, somewhat in contrast to Alex. To make it an interesting competition, there is only *one* hammer illustrated by repeated competitions between the two players in the `Ready`-locations, where the slowest player has to wait in the `Idle`-location until the faster player has finished hammering the next nail. Interestingly, despite the somewhat different strategy applied, the best- and worst-case completion times are identical for Axel and Alex: 59 seconds and 150 seconds. So, there is no difference between the two players and their strategy, or is there?

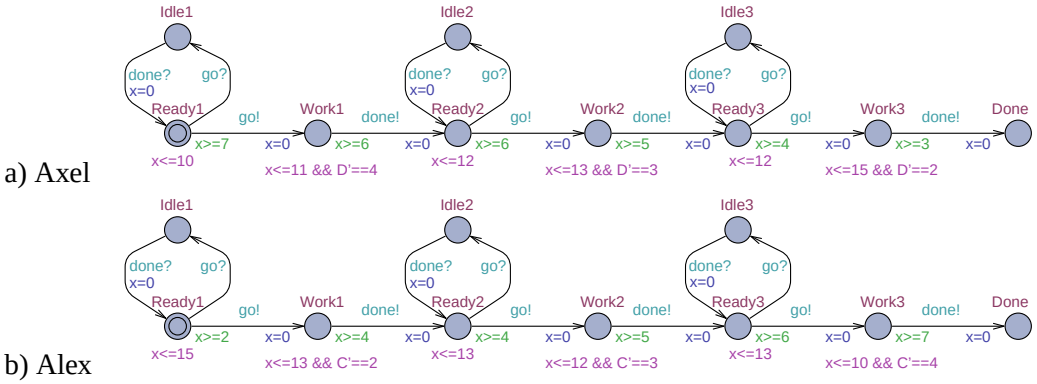
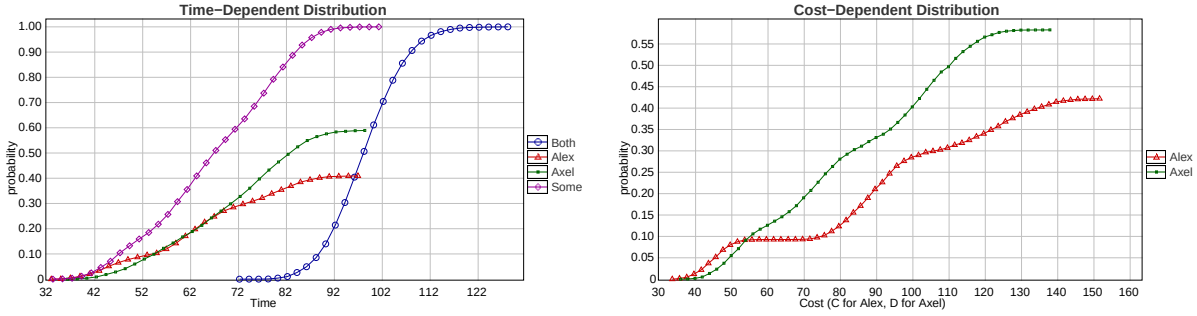


Figure 3: 3-Nail Hammer Game between Axel and Alex.

Assume now that a third person wants to bet on who is the more likely winner – Axel or Alex – given a refined semantics, where the time-delay before performing an output is chosen stochastically (e.g. by drawing from a uniform distribution) and independently by each player (component).

Under such a refined semantics there is a significant difference between the two players (Axel and Alex) in the Hammer Game. In Fig. 4a) the probability distributions for either of the two players winning before a certain time is given. Though it is clear that Axel has a higher probability of winning than Alex (59% versus 41%) given unbounded time, declaring the competition a draw if it has not finished before 50 seconds actually makes Alex the more likely winner. Similarly, Fig. 4b) illustrates the probability of either of the two players winning given an upper bound on energy. With an unlimited amount of energy, clearly Axel is the most likely winner, whereas limiting the consumption of energy to maximum 52 "energy-units" gives Alex an advantage.



a) b)

Figure 4: Time- and Cost-dependent Probability of winning the Hammer Game

Extended Input Language UPPAAL-SMC takes as input NPTAs as described above. Additionally, there is support for other features of the UPPAAL model checker's input language such as integer variables, data structures and user-defined functions, which greatly ease modeling. UPPAAL-SMC allows the user to specify an arbitrary (integer) rate for the clocks on any location. In addition, the automata support branching edges where weights can be added to give a distribution on discrete transitions. It is important to note that rates and weights may be general expressions that depend on the states and not just simple constants.

To illustrate the extended input language, we consider a train-gate example. This example is available

in the distributed version of UPPAAL-SMC. A number of trains are approaching a bridge on which there is only one track. To avoid collisions, a controller stops the trains. It restarts them when possible to make sure that trains will eventually cross the bridge. There are timing constraints for stopping the trains modeling the fact that it is not possible to stop trains instantly. The interesting point w.r.t. SMC is to define the arrival rates of these trains. Figure 5(a) shows the template for a train. The location `Safe` has no invariant and defines the rate of the exponential distribution for delays. Trains delay according to this distribution and then approach and synchronize with `appr[i]!` with the gate controller. Here we define the rational $\frac{1+id}{N^2}$ where *id* is the identifier of the train and *N* the number of trains. Rates are given by expressions that can depend on the current states. Trains with higher *id* arrive faster. Taking transitions from locations with invariants is given by a uniform distribution. This happens in `Appr`, `Cross`, and `Start`, e.g., it takes some time picked uniformly between 3 and 5 time units to cross the bridge. Figure 5(b) shows the gate controller that keeps track of the trains with an internal queue data-structure (not shown here). It uses functions to queue trains (when a train is approaching while the bridge is occupied in `Occ`) or dequeue them when possible (when the bridge is free and some train is queued).

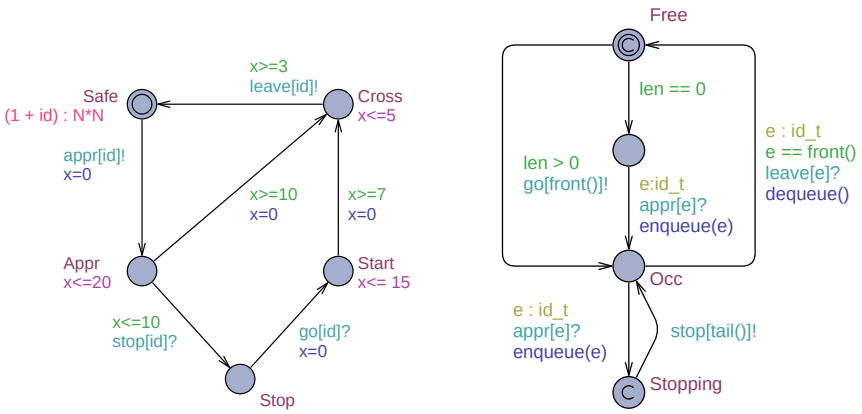


Figure 5: Template of a Train (a) and the Gate Controller (b).

Floating Point Arithmetic For modeling certain systems, e.g., biological systems, integer arithmetic shows its precision limits very quickly. The current engine implements simple arithmetic operations on clocks as floating point variables. This allows various tricks, in particular the tool can compute nontrivial functions using small step integration. For example, Figure 6(a) shows a timed automaton with floating point arithmetic. The clocks `sin_t` and `cos_t` are used to compute $sin(t)$ and $cos(t)$ using simple facts

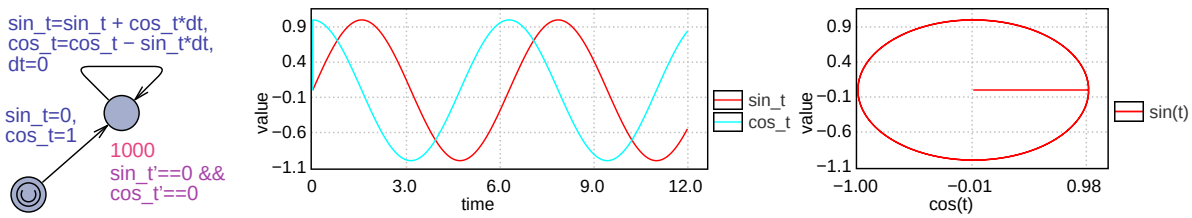


Figure 6: How to use clock arithmetic to integrate complex functions.

as $sin(t+dt) \approx sin(t) + sin'(t)dt$ for small steps of $dt \to 0$, whereas $sin'(t) = cos(t)$ and $sin(0) = 0$, and similarly for $cos(t)$. The interesting trick on the model is the high exponential rate (1000) that tells the engine to take small (random) time steps and record the duration in clock `dt`. The other clocks are stopped and updated on transition. The value evolution of variables `sin_t` and `cos_t` in terms of time are plotted in Figure 6(b). Figure 6(c) shows `sin_t` values with corresponding `cos_t` which form almost perfect circle. These plots are rendered using value monitoring features described in Section 4.

3 Properties and Queries

For specifying properties of NPTAs, we use weighted temporal properties over runs expressed in the logic WMTL$_{\leq}$[9], defined by the grammar $\varphi ::= ap \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{O}\varphi \mid \varphi_1 \mathsf{U}^{x}_{\leq d} \varphi_2$, where ap is an atomic proposition, d is a natural number and x is a clock. Here, the logical operators are interpreted as usual, and O is a next state operator. An WMTL$_{\leq}$-formula $\varphi_1 \mathsf{U}^{x}_{\leq d} \varphi_2$ is satisfied by a run if φ_1 is satisfied on the run until φ_2 is satisfied, and this will happen before the value of the clock x increases with more than d. For an NPTA M we define $\mathbb{P}_M(\psi)$ to be the probability that a random run of M satisfies ψ.

The problem of checking $\mathbb{P}_M(\psi) \geq p$ ($p \in [0,1]$) is unfortunately undecidable in general [2]. For the sub-logic of cost-bounded reachability problems $\mathbb{P}_M(\Diamond_{x \leq C} \phi) \geq p$, where ϕ is a state-predicate, x is a clock and C is bound, we approximate the answer using simulation-based algorithms known under the name of statistical model checking algorithms. We briefly recap statistical algorithms permitting to answer the following three types of questions:

1. *Hypothesis Testing:* Is the probability $\mathbb{P}_M(\Diamond_{x \leq C} \phi)$ for a given NPTA M greater or equal to a certain threshold $p \in [0,1]$?
2. *Probability evaluation:* What is the probability $\mathbb{P}_M(\Diamond_{x \leq C} \phi)$ for a given NPTA M?
3. *Probability comparison:* Is the probability $\mathbb{P}_M(\Diamond_{x \leq C} \phi_2)$ greater than the probability $\mathbb{P}_M(\Diamond_{y \leq D} \phi_2]$?

From a conceptual point of view solving the above questions using SMC is simple. First, each run of the system is encoded as a Bernoulli random variable that is true if the run satisfies the property and false otherwise. Then a statistical algorithm groups the observations to answer the three questions. For the qualitative questions (1 and 3), we shall use sequential hypothesis testing, while for the quantitative question (2) we will use an estimation algorithm that resemble the classical Monte Carlo simulation. The two solutions are detailed hereafter.

Hypothesis Testing This approach reduces the qualitative question to the test the hypothesis $H : p = \mathbb{P}_M(\Diamond_{x \leq C} \phi) \geq \theta$ against $K : p < \theta$. To bound the probability of making errors, we use strength parameters α and β and we test the hypothesis $H_0 : p \geq p_0$ and $H_1 : p \leq p_1$ with $p_0 = \theta + \delta_0$ and $p_1 = \theta - \delta_1$. The interval $p_0 - p_1$ defines an indifference region, and p_0 and p_1 are used as thresholds in the algorithm. The parameter α is the probability of accepting H_0 when H_1 holds (false positives) and the parameter β is the probability of accepting H_1 when H_0 holds (false negatives). The above test can be solved by using Wald's *sequential hypothesis testing* [38]. This test computes a proportion r among those runs that satisfy the property. With probability 1, the value of the proportion will eventually cross $\log(\beta/(1-\alpha)$ or $\log((1-\beta)/\alpha)$ and one of the two hypothesis will be selected. In UPPAAL-SMC we use the following query: `Pr[`*bound*`](`ϕ`)>=`p_0, where *bound* defines how to bound the runs. The three ways to bound them are 1) implicitly by time by specifying `<=`M (where M is a positive integer), 2) explicitly by cost with

[2] Exceptions being PTA with 0 or 1 clocks.

x<=*M* where *x* is a specific clock, or 3) by number of discrete steps with #<=*M*. In the case of hypothesis testing p_0 is the probability to test for. The formula ϕ is either <> q or [] q where q is a state predicate.

Probability Estimation This algorithm [25] computes the number of runs needed in order to produce an approximation interval $[p-\varepsilon, p+\varepsilon]$ for $p = Pr(\psi)$ with a confidence $1-\alpha$. The values of ε and α are chosen by the user and the number of runs relies on the Chernoff-Hoeffding bound. In UPPAAL-SMC we use the following query: `Pr[`*bound*`](`ϕ`)`

Probability Comparison This algorithm, which is detailed in [20], exploits an extended Wald testing. In UPPAAL-SMC, we use the following query: `Pr[`$bound_1$`](`ϕ_1`)>= Pr[`$bound_2$`](`ϕ_2`)`.

In addition to those three classical tests, UPPAAL-SMC also supports the evaluation of expected values of min or max of an expression that evaluates to a clock or an integer value. The syntax is as follows: `E[`*bound*`;`*N*`](min:`*expr*`)` or `E[`*bound*`;`*N*`](max:`*expr*`)`, where *bound* is as explained in this section, *N* gives the number of runs explicitly, and *expr* is the expression to evaluate. For this property, no confidence is given (yet).

Full WMTL$_{\leq}$ Regarding implementation, the reader shall observe that both of the above statistical algorithms are trivially implementable. To support the full logic of WMTL$_{\leq}$is slightly more complex as our simulation engine needs to rely on monitors for such logic. In [9], we proposed an extension of UPPAAL-SMC that can handle arbitrary formulas of WMTL$_{\leq}$. Given a property φ, our implementation first constructs deterministic under- and over-approximation monitoring PTAs for φ. Then it puts these monitors in parallel with a given model M, and applies SMC-based algorithms to bound the probability that φ is satisfied on M.

4 Graphical User Interface

Besides short 'yes' or 'no' answers and probability estimates, UPPAAL-SMC verifier also provides a few statistical measures in terms of time (or cost), including frequency histogram, average time (or cost), probability density distribution, cumulative probability distribution (the last two with confidence intervals, e.g. using the Clopper-Pearson method [18]).

These statistical data can also be superposed onto a single plot for comparison purposes using the plot composer tool. Figure 7 shows the superposed probability distributions of trains 0, 3 and 5 crossing from our train-gate example. On the left side of the plot composer window the user can select a particular data to be added to the plot and on the right side user can see the superposed plot and can also change some details such as labels, shapes and colors.

Monitoring Expressions UPPAAL-SMC now allows the user to visualize the values of expressions (evaluating to integers or clocks) along runs. This gives insight to the user on the behavior of the system so that more interesting properties can be asked to the model-checker. To demonstrate this on our previous train-gate example, we can monitor when `Train(0)` and `Train(5)` are crossing as well as the length of the queue. The query is `simulate 1 [<=300]{Train(0).Cross,Train(5).Cross,Gate.len}`. This gives us the plot of Figure 8. Interestingly `Train(5)` crosses more often (since it has a higher arrival rate). Secondly, it seems unlikely that the gate length drops below 3 after some time (say 20), which is not an obvious property from the model. We can confirm this by asking `Pr[<=300](<> Gate.len < 3 and t > 20)` and adding a clock `t`. The probability is in $[0.102, 0.123]$.

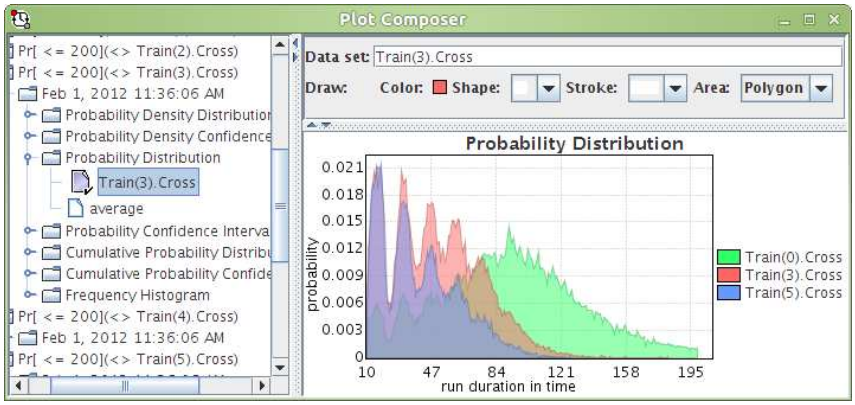


Figure 7: Snapshot of the plot composer displaying three probability distributions.

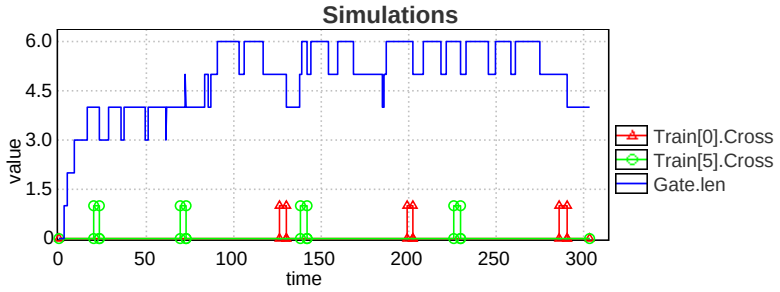


Figure 8: Visualizing the gate length and when `Train(0)` and `Train(5)` cross on one random run.

As a second example to illustrate this feature, we consider the modeling of chemical reactions. Figure 9(a) and 9(b) show two symmetric timed automata that model the concentrations of reactants `a` and `b` (here as integers). The exponential rate for taking the transition is given by the concentration of `a` and `b`. Figure 9(c) shows the evolution of the system when it is started with `a=99` and `b=1`: `a` is consumed to produce `b` and vice-versa, and the concentrations oscillate.

The simulations are obtained by querying `simulate 1 [<=10]{a,b}`. Figure 9(c) is showing one evolution of `a` and `b` over time. The tool can also plot clouds of trajectories, which is useful to identify patterns in the behavior, as shown in figure 9(d).

It is important to notice that generating such curves is not as trivial as it seems. In fact, on such models, if the exponential rates are higher, then the time steps are much smaller, which generates a lot of points, up to consuming several GB of memory. Drawing such plots is not practical. The tool would not work due to out-of-memory problems or in the best case will take around 30s to transfer the data and several seconds for every redraw. To solve this the engine applies an on-the-fly filtering of the points based on the principle that if two points are too close to each other to be distinguished on the screen, then they are considered to be the same. A resolution parameter is used to define the maximal resolution of the plot and eliminates the memory and speed problems completely (down to almost not measurable).

This plot in Figure 6(b) is obtained by asking `simulate 1 [<=12]{sin_t,cos_t}` to the model-checker. Interestingly, UPPAAL-SMC can generate a run bounded by any clock so we can also plot `simulate 1 [cos_t<=1]{sin_t}` and obtain a circle as shown in Figure 6(c).

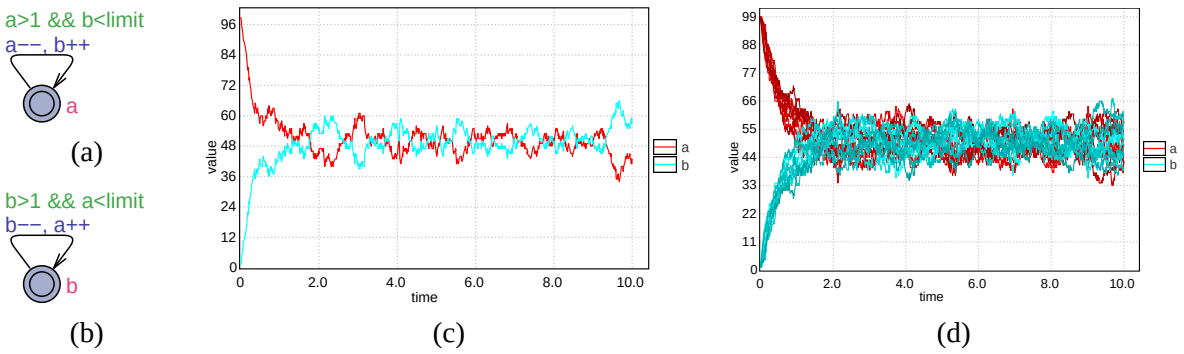


Figure 9: Evolution of the concentrations of two reactants `a` and `b`.

5 Engine

The actual techniques to achieve the current performance of the tool were never exposed before. In this section, we present a few key optimizations to implement the algorithms presented and new features that were not available in earlier versions of UPPAAL-SMC.

Distributed SMC The problem in distributing the implementation of the sequential SMC algorithm is that a *bias* may be introduced. The reason is that sequential testing relies on collecting outcomes of the generated runs on-the-fly. If some computation cores generate some accepting runs faster, which is possible if rejecting runs happen to be longer or simply more expensive to compute, then the result will be biased. The solution of this problem is to force all the cores to generate the same amount of simulations. The paper [39] proposes a method to ensure this by splitting the simulations into batches of the same size, and this method has been generalized and implemented in UPPAAL-SMC [12]. The distributed implementation gives a linear speed-up in the number of cores used.

Detection of States When choosing the delays, the engine does not know if it will *skip* the state that should be observed by the query or not. This problem is present when picking delays to take transitions as well. For example, the query could be `<> A.critical and x >= 2 and x <= 3` where `x` is a clock. The engine should not delay 4 time units from a state where `x=0` because the first possible transition is enabled at this point. Special care is taken to make sure that the formula is part of the next *interesting* points that are computed when choosing the delays. Now comes the question of how to detect those interesting points in both the formula and the guards.

The technique we use follows the decorator pattern where we evaluate guards (for detecting which transitions will be enabled in the future) and formulas in the query to keep track of the lower bounds. We wrap a state inside a decorator state that keeps track of the constraints on-the-fly, only remembering the bounds that we need. The point of the technique here is to avoid *symbolic* states that would require zones typically implemented with different bound matrices.

Early Termination The engine checks for query on-the-fly on every generated run. If a query is satisfied then the computation of the run is stopped before it reaches the specified bound. In addition, in order to give the user a way to stop runs earlier, the engine supports an *until* property: `p U q` can be queried instead of `<> q` and cut the runs as soon as `p` stops to hold.

Dependencies and Reuse of Choice When a process takes an action, if may not affect other processes, which means that from a stochastic point-of-view, picking a new delay from scratch or keeping the old choice (that was random) is equivalent. The engine exploit this independence: it remembers the previous delays chosen by the processes and invalidates them when dependent transitions are taken. A process has its delay invalidated if there is a dependency with another transition being taken, which happens in case of synchronization or a dependency through a clock rate, invariant, guard, or update. A static analysis is made at the granularity of *how transitions affect processes*[3].

The result is that whenever a process *needs* to pick a delay, it does so. Whenever a process takes a transition, the processes that may be affected by it must pick a new delay at the next step. Otherwise, processes keep their choices from the previous step in the simulation[4].

Checking the query `Pr[<=300](<> Train(0).Cross and (forall (i:id_t) i!=0 imply Train(i).Stop))` to evaluate the probability of `Train(0)` crossing while all the others are stopped gives the results in table 1 for different numbers of trains. The results are obtained with the parameter $\varepsilon = 0.005$ and the probability results agree with or without reuse within ε. The experiments are made on a core i7 at 2.66GHz. This optimization is designed to improve on systems with large number of components, which is shown by the increasing improvement relative to verifications without reuse.

Trains	5	10	20	40
Proba.	0.985-0.995	0.286-0.297	0-0.008	0-0.005
Time$^-$	3.9s	17.3s	41.1s	98.1s
Time$^+$	3.5s	14.8s	33.2s	74.8s
Gain	10.2%	14.4%	19.2%	23.8%

Table 1: Probability and time results without (-) and with (+) reuse.

6 Case-Studies

In this section we evaluate the applicability of the developed techniques on practical case studies.

Robot Control In the paper [9] we considered a case – explored in [4] – of a robot moving on a two-dimensional grid. Each field of the grid is either `normal`, on `fire`, cold as `ice` or it is a wall which cannot be passed. Also, there is a `goal` field that the robot must reach. The robot is moving in a random fashion i.e. it stays in a field for some time, and then moves to a neighboring field at random (if it is not a wall).

We are interested in the probability that the robot reaches its goal location without staying on consecutive fire fields for more than one time unit and on consecutive ice fields for more than two time units. This property is captured by the WMTL$_{\leq}$ formula $\varphi \equiv (\varphi_1 \wedge \varphi_2)\mathsf{U}^{\tau}_{\leq 10}\,\texttt{goal}$, where τ is a special clock that grows with rate 1 and is never reset, and:

$$\varphi_1 \equiv \texttt{ice} \implies \Diamond^{\tau}_{\leq 2}(\texttt{fire} \vee \texttt{normal} \vee \texttt{goal})$$
$$\varphi_2 \equiv \texttt{fire} \implies \Diamond^{\tau}_{\leq 1}(\texttt{ice} \vee \texttt{normal} \vee \texttt{goal})$$

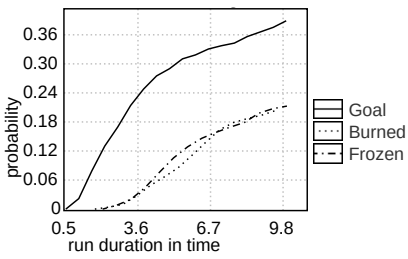


Figure 10: Cumulative Probability

We applied UPPAAL-SMC to compute the probability of the robot reaching the goal φ, staying too long in the fire or too long on the ice. Figure 10 shows the cumulative distribution for these probabilities.

[3] We judge that keeping track of the dependencies down to the locations may have a too large overhead.

[4] If time elapses then of course the delays chosen are updated.

Firewire. IEEE 1394 High Performance Serial Bus or Firewire for short is used to transport multimedia signals among a network of consumer devices. The protocol has been extensively studied (see [36] for comparison) and in particular [30] uses probabilistic timed automata in PRISM [29]. In paper [21] we adopt the model from [30] and demonstrate how UPPAAL-SMC can be used to evaluate fairness of a node becoming a root (leader) with respect to the mode of operation. UPPAAL-SMC provides two methods for comparing probabilities: estimating the probabilities and then comparing them (slow method) or using indirect probability comparison from [38] (fast method). Figure 11 contains a resulting plot of estimated probabilities (red and blue lines) and a comparison (yellow area). The red and blue probability estimates appear very close to each other in entire range, while the yellow area shows that at the beginning the probabilities are indistinguishable (yellow area is at 0.5 level), then the *fast* node has higher probability to become a *root* (at 1.0 level), and later the probabilities become too close to be distinguishable again (at 0.5 level).

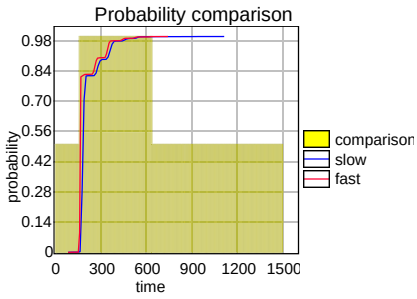


Figure 11: Probability Comparison

Bluetooth [33] is a wireless telecommunication protocol using frequency-hopping to cope with interference between the devices in the wireless network. In paper [21] we adopted the model from [22], annotated the model to record the power utilization and evaluated the probability distributions of likely response times and energy consumption. Figure 12 shows that after 70s the cost of a device operation is at least 2440 energy units and the mean is about 2853 energy units.

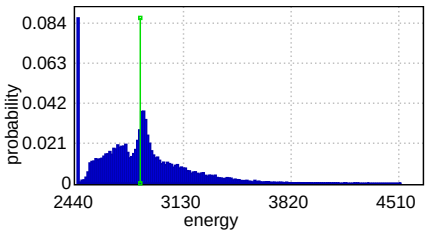


Figure 12: Energy consumption.

Lightweight Medium Access Protocol (LMAC) [37] is a communication scheduling protocol based on time slot distribution for nodes sharing the same medium. The protocol is designed for wireless sensor networks in mind: it is simple enough to fit on a modest hardware and at the same time robust against topology reconfiguration, minimizing collisions and power consumption. Paper [24] studies LMAC protocol using classical UPPAAL verification techniques by systematically exploring networks of up to five nodes but the state space explosion prevents formal verification of larger networks. In paper [20] we adopt the model by removing verification optimizations and parameterizing with probabilistic weights, and show how collisions can be analyzed and power consumption estimated using statistical model checking techniques. The study showed that there are still perpetual collisions in a ring topology but the probability that the network will not recover is very low (0.35%). The likely energy consumption of different network topologies is compared in UPPAAL plot (Figure 13), which shows that on average the likely energy consumption after 1000 time units in a ring is higher than in a chain by 10%, possibly due to more collisions in a ring. In [12] distributed techniques are applied in exploring over 10000 larger networks of up to 10 nodes, the worst (star-like) and the best (chain-like) topologies in

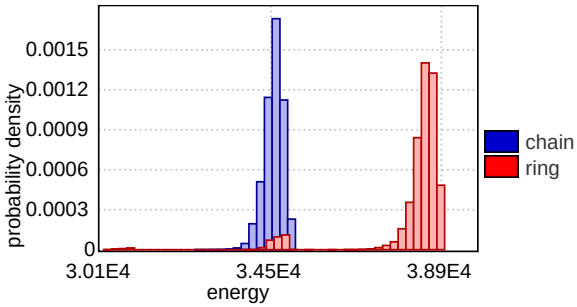


Figure 13: Likely energy consumption.

terms of collisions are identified and evaluated.

Computing Nash Equilibrium in Wireless Ad Hoc Networks One of the important aspects in designing wireless ad-hoc networks is to make sure that a network is robust to the selfish behavior of its participants, i.e. that its configuration satisfies Nash equilibrium (NE).

In paper [10] we proposed an SMC-based algorithm for computing NE for the case when a network nodes are modeled by SPTA and an utility function of a single node is equal to a probability that the node will reach its goal. Our algorithm consists of two phases. First, we use UPPAAL-SMC to find a strategy that most likely (heuristic) satisfies NE. In the second phase we apply statistics to test the hypothesis that this strategy actually satisfies NE.

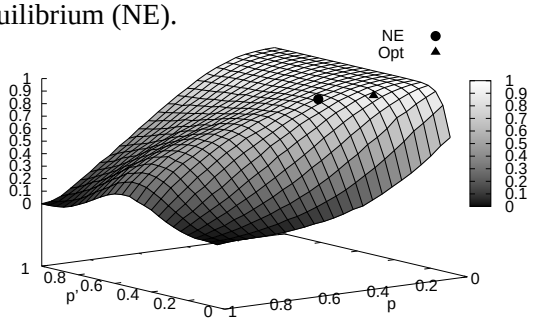


Figure 14: Nash Equilibrium for Aloha CSMA/CD

We applied this algorithm to compute NE for Aloha CSMA/CD and IEEE 802.15.4 CSMA/CA protocols. Figure 14 depicts the utility function plot for Aloha CSMA/CD protocol with two nodes. Here p and p' axis correspond to the strategies of the honest and cheater nodes (a strategy defines how persistent these nodes are in sending their data). We see, that NE strategy is slightly less efficient than the symmetric optimal strategy (Opt), but it still results in a high value of the utility function.

Duration Probabilistic Automata In [19] we compared UPPAAL-SMC to Prism [29] in the context of Duration Probabilistic Automata (DPA) [31]. A Duration Probabilistic Automaton (DPA) is a composition of Simple Duration Probabilistic Automata (SDPA). An SDPA is a linear sequence of tasks that must be performed in a sequential order.

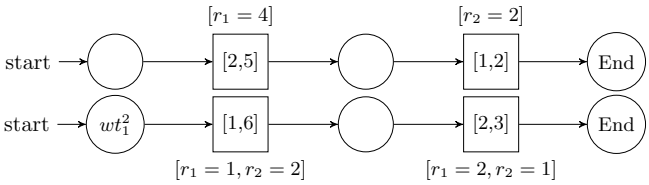


Figure 15: Rectangles are busy states and circles are for waiting when resources are not available. There are $r_1 = 5$ and $r_2 = 3$ resources available.

Each task is associated with a duration interval which gives the possible durations of the task. The actual duration of the tasks is given by a uniform choice from this interval. To model races between the SDPAs we introduce resources to the model such that an SDPA might have to wait for resources before processing a task. When two SDPAs are in waiting position for the same resource, a scheduler decides which SDPA is given the resource in a deterministic manner.

The comparison with Prism was made by randomly generating models with a specific number of SDPAs and a

Param.			**Estim.**				**Hyp. Testing**			
n	k	m	Prism	Up_p	Up_d	Up_c	Prism	Up_p	Up_d	Up_c
4	4	3	2.7	0.3	0.2	0.2	2.0	0.1	0.1	0.1
6	6	3	7.7	0.6	0.5	0.4	3.9	0.2	0.2	0.3
8	8	3	26.5	1.2	0.9	0.7	16.4	0.5	0.4	0.3
20	40	20		>300			>300	35.5	26.2	20.7
30	40	20		>300			>300	61.2	41.8	33.2
40	40	20		>300			>300	92.2	56.9	59.5
40	20	20		>300			>300	41.1	31.2	26.5
40	30	20		>300			>300	68.8	46.7	46.1
40	55	40		>300				>300		219.5

Table 2: Performance of SMC (sec). The n column is the number SDPAs, the k column is the number of tasks per SDPA and the m column is the number resource types in the model. Up_p is the UPPAAL model that matches Prism, Up_d the discrete encoding and Up_c the continuous time encoding.

specific number of tasks per SDPA and
translate these into Prism and UPPAAL models. The Prism model uses a discrete time semantics whereas three models were made for UPPAAL- one with continuous time semantics, one that matches the Prism model as close as possible and one with discrete semantics that makes full use of our formalism.
The queries to the models were *What is the probability of all SDPAs ending within t time units* (Estimation)and *Is the probability that all SDPAs end within t time units greater than* 40% (Hypothesis testing). The value of t is different for each model as it was computed by simulating the system 369 times and represent the value for which at least 60% of the runs finished all their tasks.

The result of the experiments are shown in Table 2 and indicates that UPPAAL is notably faster than Prism, even with a encoding that closely matches that of Prism.

Checking of Distributed Statistical Model Checking As we wrote in the Section 5, a naive (and incorrect) distributed implementation of the sequential SMC algorithms might introduce a bias towards the results that are generated by shorter simulations.

The interesting question is how much this bias affects the SMC results. In the paper [11] we answered this question by modeling the naive distributed SMC algorithm in UPPAAL-SMC itself. The comparison was made on the basis of the SPTA model that ends up in the `OK` location after 100 time units with probability 0.58, otherwise it ends up in the `NOK` location after 1 time unit (thus producing `NOK` requires 100 times less time than producing `OK`).

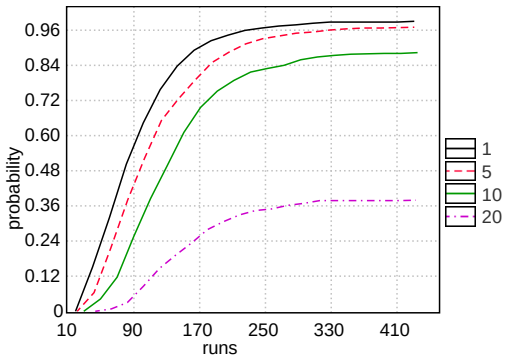


Figure 16: Probability distributions obtained with 1, 5, 10, and 20 cores.

We used UPPAAL-SMC to compute the probability that the naive distributed SMC algorithm will accept the hypothesis `Pr[<=100](◇ OK)≥ 0.5`. The results for the different numbers of computational cores are given at the plot at Figure 16. The x axis denotes the total number of runs of the SPTA model on all the cores, and the y axis depicts the probability that an SMC algorithm accepts the hypothesis not later than after this number of runs. You can see, that the probability of accepting the hypothesis tends (incorrectly) to 0 as the number of computational cores increases.

7 Conclusions

This paper overviews the features of UPPAAL-SMC, our new efficient extension of UPPAAL for Statistical Model Checking. Contrary to other existing SMC-based tool-sets, UPPAAL-SMC allows to handle systems with real-time features. The tool has been applied to a series of case studies that are beyond the scope of classical model checkers. UPPAAL-SMC has a large potential for future work and applications.

Among others, the following extensions of UPPAAL-SMC are contemplated.

Floating Point So far the support of floating point is done via misusing and extending clock operations. A better more general support is needed since the tool has now departed from traditional timed automata and model-checking.

Since the tool now supports floating point arithmetic and we can integrate complex functions, it is a natural extension to add differential equations as well to support hybrid systems in a more general

way. To fit with the stochastic semantics (in particular how to pick delays), only simple equations whose analytical solutions are known are planned.

New Applications With the extended expressivity of our hybrid modeling language, our tool can be applied to different domains, in particular for biological systems. UPPAAL-SMC now offers powerful visualization capabilities needed by biologists and a logic to do statistical model-checking.

Another application is to analyze performance of controllers generated by UPPAAL-TIGA, in particular their stability or energy consumption. SMC can also be used in the domain of refinement checking, which is in the end just another type of game.

Rare Events Statistical model checking avoids the exponential growth of states associated with probabilistic model checking by estimating properties from multiple executions of a system and by giving results within confidence bounds. Rare properties are often very important but pose a particular challenge for simulation-based approaches, hence a key objective under these circumstances is to reduce the number and length of simulations necessary to produce a given level of confidence. Importance sampling is a well-established technique that achieves this, however to maintain the advantages of statistical model checking it is necessary to find good importance sampling distributions without considering the entire state space. Such problem has been recently investigated for the case of discrete stochastic systems. As an example, in [27] we presented a simple algorithm that uses the notion of cross-entropy to find the optimal parameters for an importance sampling distribution. Our Objective is to extend our results to PTAs by exploiting pure timed model checking to improve the search for efficient distribution.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis & S. Yovine (1995): *The algorithmic analysis of hybrid systems*. *Theoretical Computer Science* 138(1), pp. 3–34.

[2] Rajeev Alur, Salvatore La Torre & George J. Pappas (2001): *Optimal Paths in Weighted Timed Automata*. In Benedetto & Sangiovanni-Vincentelli [7], pp. 49–62. Available at `http://link.springer.de/link/service/series/0558/bibs/2034/20340049.htm`.

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

[4] Benoît Barbot, Taolue Chen, Tingting Han, Joost-Pieter Katoen & Alexandru Mereacre (2011): *Efficient CTMC Model Checking of Linear Real-Time Objectives*. In ParoshAziz Abdulla & K.RustanM. Leino, editors: *Tools and Algorithms for the Construction and Analysis of Systems*, *Lecture Notes in Computer Science* 6605, Springer Berlin Heidelberg, pp. 128–142, doi:`10.1007/978-3-642-19835-9_12`. Available at `http://dx.doi.org/10.1007/978-3-642-19835-9_12`.

[5] A. Basu, S. Bensalem, M. Bozga, B. Caillaud, B. Delahaye & A. Legay (2010): *Statistical Abstraction and Model-Checking of Large Heterogeneous Systems*. In: *FORTE*, *LNCS* 6117, Springer, pp. 32–46.

[6] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn & Frits W. Vaandrager (2001): *Minimum-Cost Reachability for Priced Timed Automata*. In Benedetto & Sangiovanni-Vincentelli [7], pp. 147–161. Available at `http://link.springer.de/link/service/series/0558/bibs/2034/20340147.htm`.

[7] Maria Domenica Di Benedetto & Alberto L. Sangiovanni-Vincentelli, editors (2001): *Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Proceedings*. *LNCS* 2034, Springer.

[8] Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen & Nicolas Markey (2010): *Timed automata with observers under energy constraints*. In: *HSCC*, ACM ACM, pp. 61–70.

[9] Peter Bulychev, Alexandre David, Kim G. Larsen, Axel Legay, Guangyuan Li, Danny Bøgsted Poulsen & Amelie Stainer (2012): *Monitor-Based Statistical Model Checking for Weighted Metric Temporal Logic*. In: *LPAR-18*. To appear in LNCS.

[10] Peter Bulychev, Alexandre David, Kim G. Larsen, Axel Legay & Marius Mikučionis (2012): *Computing Nash Equilibrium in Wireless Ad Hoc Networks: A Simulation-Based Approach*. In: *iWIGP*. To appear in EPTCS.

[11] Peter Bulychev, Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis & Danny Bøgsted Poulsen (2012): *Checking & Distributing Statistical Model Checking*. In: *NFM 2012 : Fourth NASA Formal Methods Symposium*. To appear in LNCS.

[12] Peter Bulychev, Alexandre David, Kim Guldstrand Larsen, Marius Mikučionis & Axel Legay: *Distributed Parametric and Statistical Model Checking*. In Jiří Barnat & Keijo Heljanko, editors: *Proceedings 10th International Workshop on Parallel and Distributed Methods in Verification*, EPTCS, arxiv.org, doi:10.4204/EPTCS.72.

[13] F. Ciesinski & M. Größer (2004): *On Probabilistic Computation Tree Logic*. In: *Validation of Stochastic Systems*, LNCS, 2925, Springer, pp. 147–188.

[14] E. M. Clarke, A. Donzé & A. Legay (2008): *Statistical Model Checking of Mixed-Analog Circuits with an Application to a Third Order Delta-Sigma Modulator*. In: *Haifa Verification Days, LNCS* 5394, Springer, pp. 149–163.

[15] E. M. Clarke, A. Donzé & A. Legay (2009): *On Simulation-based Probabilistic Model Checking of Mixed-Analog Circuits*. *Formal Methods in System Design* To appear.

[16] E. M. Clarke, J. R. Faeder, C. James Langmead, L. A. Harris, S. K. Jha & A. Legay (2008): *Statistical Model Checking in BioLab: Applications to the Automated Analysis of T-Cell Receptor Signaling Pathway*. In: *CMSB, LNCS* 5307, Springer, pp. 231–250.

[17] Edmund M. Clarke & Paolo Zuliani (2011): *Statistical Model Checking for Cyber-Physical Systems*. In: *ATVA, Lecture Notes in Computer Science* 6996, Springer, pp. 1–12.

[18] C. J. Clopper & E. S. Pearson (1934): *The Use of Confidence or Fiducial Limits Illustrated in the Case of the Binomial*. *Biometrika* 26(4), pp. 404–413, doi:10.1093/biomet/26.4.404. Available at http://biomet.oxfordjournals.org/content/26/4/404.short.

[19] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, Jonas Van Vliet & Zheng Wang (2011): *Statistical model checking for networks of priced timed automata*. In: *Proceedings of the 9th international conference on Formal modeling and analysis of timed systems*, FORMATS'11, Springer-Verlag, Berlin, Heidelberg, pp. 80–96. Available at http://dl.acm.org/citation.cfm?id=2044973.2044982.

[20] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, Jonas van Vliet & Zheng Wang (2011): *Statistical Model Checking for Networks of Priced Timed Automata*. In Uli Fahrenberg & Stavros Tripakis, editors: *Formal Modeling and Analysis of Timed Systems, LNCS* 6919, springer, pp. 80–96.

[21] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis & Zheng Wang (2011): *Time for Statistical Model Checking of Real-Time Systems*. In Ganesh Gopalakrishnan & Shaz Qadeer, editors: *Computer Aided Verification, Lecture Notes in Computer Science* 6806, Springer, pp. 349–355.

[22] Marie Duflot, Marta Kwiatkowska, Gethin Norman & David Parker (2006): *A formal analysis of bluetooth device discovery*. *International Journal on Software Tools for Technology Transfer (STTT)* 8, pp. 621–632, doi:10.1007/s10009-006-0014-x.

[23] Uli Fahrenberg, Line Juhl, Kim G. Larsen & Jirí Srba (2011): *Energy Games in Multiweighted Automata*. In: *ICTAC, Lecture Notes in Computer Science* 6916, Springer, pp. 95–115.

[24] Ansgar Fehnker, Lodewijk van Hoesel & Angelika Mader (2007): *Modelling and Verification of the LMAC Protocol for Wireless Sensor Networks*. In Jim Davies & Jeremy Gibbons, editors: *Integrated Formal Methods, LNCS* 4591, Springer Berlin / Heidelberg, pp. 253–272.

[25] Thomas Hérault, Richard Lassaigne, Frédéric Magniette & Sylvain Peyronnet (2004): *Approximate Probabilistic Model Checking*. In: *VMCAI, LNCS* 2937, Springer, pp. 73–84.

[26] D. N. Jansen, J. Katoen, M. Oldenkamp, M. Stoelinga & I. S. Zapreev (2008): *How Fast and Fat Is Your Probabilistic Model Checker? An Experimental Performance Comparison*. In: *Haifa Verification Conference, LNCS* 4899, Springer, pp. 69–85.

[27] Cyrille Jégourel, Axel Legay & Sean Sedwards (2012): *Cross-entropy optimisation of importance sampling parameters for statistical model checking*. *CoRR* abs/1201.5229.

[28] Sumit Kumar Jha, Edmund M. Clarke, Christopher James Langmead, Axel Legay, André Platzer & Paolo Zuliani (2009): *A Bayesian Approach to Model Checking Biological Systems*. In: *CMSB, LNCS* 5688, Springer, pp. 218–234.

[29] M. Z. Kwiatkowska, G. Norman & D. Parker (2004): *PRISM 2.0: A Tool for Probabilistic Model Checking*. In: *QEST*, IEEE, pp. 322–323.

[30] Marta Kwiatkowska, Gethin Norman & Jeremy Sproston (2003): *Probabilistic Model Checking of Deadline Properties in the IEEE 1394 FireWire Root Contention Protocol*. *Formal Aspects of Computing* 14, pp. 295–318. Available at `http://dx.doi.org/10.1007/s001650300007`. 10.1007/s001650300007.

[31] Oded Maler, Kim G. Larsen & Bruce H. Krogh (2010): *On Zone-Based Analysis of Duration Probabilistic Automata*. In: *INFINITY, EPTCS* 39, pp. 33–46.

[32] João Martins, André Platzer & João Leite (2011): *Statistical Model Checking for Distributed Probabilistic-Control Hybrid Automata with Smart Grid Applications*. In: *ICFEM, Lecture Notes in Computer Science* 6991, Springer, pp. 131–146.

[33] P. McDermott-Wells (2005): *What is Bluetooth? Potentials, IEEE* 23(5), pp. 33 – 35, doi:`10.1109/MP.2005.1368913`.

[34] Johan Oudinet, Alain Denise, Marie-Claude Gaudel, Richard Lassaigne & Sylvain Peyronnet (2011): *Uniform Monte-Carlo Model Checking*. In: *FASE, Lecture Notes in Computer Science* 6603, Springer, pp. 127–140.

[35] Koushik Sen, Mahesh Viswanathan & Gul Agha (2004): *Statistical Model Checking of Black-Box Probabilistic Systems*. In: *CAV*, LNCS 3114, Springer, pp. 202–215.

[36] Mariëlle Stoelinga (2003): *Fun with FireWire: A Comparative Study of Formal Verification Methods Applied to the IEEE 1394 Root Contention Protocol*. *Formal Aspects of Computing* 14, pp. 328–337. Available at `http://dx.doi.org/10.1007/s001650300009`. 10.1007/s001650300009.

[37] L.F.W. van Hoesel & P.J.M. Havinga (2004): *A Lightweight Medium Access Protocol (LMAC) for Wireless Sensor Networks: Reducing Preamble Transmissions and Transceiver State Switches*. In: *1st International Workshop on Networked Sensing Systems (INSS*, Society of Instrument and Control Engineers (SICE), Tokio, Japan, pp. 205–208. Available at `http://doc.utwente.nl/64756/`.

[38] R. Wald (2004): *Sequential Analysis*. Dove Publisher.

[39] Håkan L. S. Younes (2005): *Verification and Planning for Stochastic Processes with Asynchronous Events*. Ph.D. thesis, Carnegie Mellon.

[40] Håkan L. S. Younes, Marta Z. Kwiatkowska, Gethin Norman & David Parker (2006): *Numerical vs. statistical probabilistic model checking*. *STTT* 8(3), pp. 216–228.

Efficient computation of exact solutions for quantitative model checking

Sergio Giro
Department of Computer Science, University of Oxford, Oxford, OX1 3QD, UK*

Quantitative model checkers for Markov Decision Processes typically use finite-precision arithmetic, since exact techniques are generally too expensive or limited in scalability. In this paper we propose a method for obtaining exact results starting from an approximated solution in finite-precision arithmetic. The input of the method is a description of a scheduler, which can be obtained by a model checker using finite precision. Given a scheduler, we show how to obtain a corresponding basis in a linear-programming problem, in such a way that the basis is optimal whenever the scheduler attains the worst-case probability. This correspondence is already known for discounted MDPs, we show how to apply it in the undiscounted case provided that some preprocessing is done. Using the correspondence, the linear-programming problem can be solved in exact arithmetic starting from the basis obtained. As a consequence, the method finds the worst-case probability even if the scheduler provided by the model checker was not optimal. In our experiments, the calculation of exact solutions from a candidate scheduler is significantly faster than the calculation using the simplex method under exact arithmetic starting from a default basis.

1 Introduction

Model checking of Markov Decision Processes (MDPs) has been proven to be a useful tool to verify and evaluate systems with both probabilistic and non-deterministic choices. Given a model of the system under consideration and a qualitative property concerning probabilities, such as "the system fails to deliver a message with probability at most 0.05", a model checker deduces whether the property holds or not for the model. As different resolutions of the non-deterministic choices lead to different probability values, verification techniques for MDPs rely on the concept of *schedulers* (also called policies, or adversaries), which are defined as functions choosing an option for each of the paths of an MDP. Model-checking algorithms for MDPs proceed by reducing the model-checking problem to that of finding the maximum (or minimum) probability to reach a set of states under all schedulers [5].

Different techniques for calculating these extremal probabilities exist: for an up-to-date tutorial, see [10]. Some of them (for instance, value iteration) are approximate in nature, while a technique using linear programming (LP) can be used to obtain exact solutions. However, even the linear programming method is often carried out using finite-precision, and so the results are always approximations. Exact solutions are hard to get in practice, because linear programming methods for MDPs using exact arithmetic do not scale well. (To support this claim we performed some experiments showing how costly it is to compute exact probabilities using LP without our method.) In addition, the native operators in programming languages like Java have finite precision: the extension to exact arithmetic involves significant reworking of the existing code.

*The authors are part supported by DARPA/Air Force Research Laboratory contract FA8650-10-C-7077 (PRISMATIC)

Preliminary Report. Final version to appear in:
QAPL 2012

We propose a method for computing exact solutions. Given any approximative algorithm being able to provide a description of a scheduler, our method shows how to extend the algorithm in order to get exact solutions. The method exploits the well-known correspondence between model-checking problems and linear programming problems [5], which allows to compute worst-case probabilities by computing optimal solutions for LP problems.

The simplex algorithm [6] for linear programming works by iterating over different *bases*, which are submatrices of the matrix associated to the LP problem. Each basis defines a *solution*, that is, a valuation on the variables of the problem. The simplex method stops when the basis yields a solution with certain properties, more precisely, a so-called *feasible* and *dual feasible* solution. By algebraic properties, such a solution is guaranteed to be optimal.

The core of our method is the interpretation of the scheduler as a basis for the linear programming problem. Given a scheduler complying with certain natural conditions, a basis corresponding to the scheduler can be used as a starting point for the simplex algorithm. We show that, if the scheduler is optimal, then the solution associated to the corresponding basis is feasible and dual feasible, and so a simplex solver provided with this basis needs only to check dual feasibility and compute the solution corresponding to the basis. As our experiments show, these computations can be done in exact arithmetic without a huge impact in the overall model-checking time. In fact, using the dual variant of the simplex method, the time to obtain the exact solution is less than the time spent by value iteration. If the scheduler is not optimal, the solver starts the iterations from the basis. This is useful for two reasons: we can let the simplex solver finish in order to get the exact solution; or, once we know that we are not getting the optimal solution, we can perform some tuning in the model checker as, for instance, reduce the convergence threshold (we also show a case in which the optimal scheduler cannot be found with thresholds within the 64-bit IEEE 754 floating point precision).

The correspondence between schedulers and bases is already known for discounted MDPs (see, for instance [8]). We show the correspondence for the undiscounted case in case some states of the system are eliminated in preprocessing steps. The preprocessing steps we consider are usual in model checking [10]: given a set of target states, one of the preprocessing algorithms removes the states that cannot reach the target, while the other one removes the states that can avoid reaching the target. These are qualitative algorithms based on graphs that do not perform any arithmetical operations.

The next section introduces the preliminary concepts we need along the paper. Section 3 presents our method and the proof of correctness. The experiments are shown in Section 4. The last section discusses related results concerning complexity and policy iteration.

2 Preliminaries

We introduce the definitions and known-results used throughout the paper, concerning both Markov decision process and linear programming.

2.1 Markov decision processes

Definition 1. Let $\mathrm{Dist}(A)$ denote the set of discrete probability distributions over the set A. A Markov Decision Process (MDP) M is a pair (S,T) where S is a finite set of *states* and $\mathsf{T} \subseteq \mathsf{S} \times \mathrm{Dist}(\mathsf{S})$ is a set

of *transitions* [1]. Given $\mu = (s,d) \in \mathsf{T}$, the value $d(t)$ is the probability of making a transition to t from s using μ. We write $\mu(t)$ instead of $d(t)$, and write state(μ) for s. We define the set en(s) as the set of all transitions μ with state$(\mu) = s$. For simplicity, we make the usual assumption that every state has at least one enabled transition: en$(s) \neq \emptyset$ for all $s \in \mathsf{S}$.

We write $s \xrightarrow{\mu} t$ to denote $\mu \in \text{en}(s) \wedge \mu(t) > 0$. A path in an MDP is a (possibly infinite) sequence $\rho = s^0.\mu^1.s^1.\cdots..\mu^n.s^n$, where $\mu^i \in \text{en}(s^{i-1})$ and $\mu^i(s^i) > 0$ for all i. If ρ is finite, the last state of ρ is denoted by last(ρ), and the length is denoted by len(ρ) (a path having a single state has length 0). Given a set of states U, we define reach(U) to be the set of all infinite paths $\rho = s^0.\mu^1.s^1.\cdots$ such that $s^i \in U$ for some i.

The semantics of MDPs is given by schedulers. A scheduler η for an MDP M is a function $\eta : \mathsf{S} \to \mathsf{T}$ such that $\eta(s) \in \text{en}(s)$ for all s. In words, the scheduler chooses an enabled transition based on the current state. For all schedulers η, $t \in \mathsf{S}$, the set Paths(t,η) contains all the paths $s^0.\mu^1.s^1.\cdots.\mu^n.s^n$ such that $s^0 = t$, $\mu^i = \eta(s^{i-1})$ and $s^{i-1} \xrightarrow{\mu^i} s^i$ for all i. The reader familiar with MDPs might note that we are restricting to Markovian non-randomized schedulers (that is, they map states to transitions, instead of the more general schedulers mapping paths to distributions on transitions). As explained later on, these schedulers suffice for our purposes.

The probability $\Pr_M^{t,\eta}(\rho)$ of the path ρ under η starting from t is $\prod_{i=1}^{\text{len}(\rho)} \mu^i(s^i)$ if $\rho \in \text{Paths}(t,\eta)$. If $\rho \notin \text{Paths}(t,\eta)$, then the probability is 0. We often omit the subindices M and/or t if they are clear from the context.

We are interested on the probability of (sets of) infinite paths. Given a finite path ρ, the probability of the set $\rho^\uparrow$ comprising all the infinite paths that have ρ as a prefix is defined by $\Pr^\eta(\rho^\uparrow) = \Pr^\eta(\rho)$. In the usual way (that is, by resorting to the Carathéodory extension theorem) it can be shown that the definition on the sets of the form $\rho^\uparrow$ can be extended to σ-algebra generated by the sets $\rho^\uparrow$.

The verification of PCTL* [5] and ω-regular formulae [7] (for example LTL) can be reduced to the problem of calculating $\max_{s,\eta} \Pr_{M'}^{s,\eta}(\text{reach}(U))$ (or $\min_{s,\eta} \Pr_{M'}^{s,\eta}(\text{reach}(U))$) for MDPs M', states s and sets U obtained from the formula.

In consequence, in the rest of the paper we concentrate on the following problems.

Definition 2. Given an MDP M, an initial state s and set of *target* states U, a *reachability problem* consists of computing $\max_{s,\eta} \Pr_M^{s,\eta}(\text{reach}(U))$ (or $\min_\eta \Pr_M^{s,\eta}(\text{reach}(U))$).

From classic results in MDP theory (for these results applied to model checking see, for instance, [2, Chapter 3]) there exists a scheduler η^* such that

$$\eta^* = \arg\max_\eta \Pr_M^{s,\eta}(\text{reach}(U)) \tag{1}$$

for all $s \in \mathsf{S}$. That is, η^* attains the maximum probability for all states.

An analogous result holds for the the case of minimum probabilities. There exists η^* such that

$$\eta^* = \arg\min_\eta \Pr_M^{s,\eta}(\text{reach}(U)) \tag{2}$$

for all $s \in \mathsf{S}$.

Even in a more general setting allowing for non-Markovian and randomized schedulers, it can be proven that we can assume η^* to be Markovian and non-randomized. The existence of η^* justifies our restriction to Markovian and non-randomized schedulers.

[1] Defining transitions as pairs helps to deal with the case in which the same distribution is enabled in several states

Markov chains A Markov chain (MC) is an MDP such that $|\mathrm{en}(s)| = 1$ for all $s \in \mathsf{S}$. Note that a Markov chain has exactly one scheduler, namely the one that chooses the only transition enabled in each state. Hence, for Markov chains, we often disregard the scheduler and denote the probability of reaching U as $\Pr^s_M(\mathrm{reach}(U))$.

Definition 3. Given an MDP $M = (\mathsf{S}, \mathsf{T})$ and a scheduler η, we define the Markov chain $M \downarrow \eta = (\mathsf{S}, \mathsf{T}')$ where $\mu \in \mathsf{T}'$ iff $\eta(\mathrm{state}(\mu)) = \mu$.

A simple application of the definitions yields

$$\Pr^{s,\eta}_M(\mathrm{reach}(U)) = \Pr^s_{M\downarrow\eta}(\mathrm{reach}(U)) \ . \tag{3}$$

2.2 Linear programming

We use a particular canonical form of linear programs suitable for our needs. It is based on [6, Appendix B], which is also a good reference for all the concepts and results given in this subsection.

A linear programming problem consists in computing

$$\min_{\mathbf{x}} \{\mathbf{cx} \mid A\mathbf{x} = \mathbf{b} \wedge \mathbf{x} \geq \mathbf{0}\} \ , \tag{4}$$

given a *constraint matrix* A, a *constraint vector* $\mathbf{b}$ and a *cost vector* $\mathbf{c}$. In the following, we assume that A has m rows and $m+n$ columns, for some $m > 0$ and $n \geq 0$. Hence, $\mathbf{c}$ is a row vector with $m+n$ components, and $\mathbf{b}$ is a column vector with m components.

A *solution* is any vector $\mathbf{x}$ of size $m+n$. The i-th component of $\mathbf{x}$ is denoted by x_i. We say that that a solution is *feasible* if $A\mathbf{x} = \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$; it is optimal if is feasible and $\mathbf{cx}$ is minimum over all feasible $\mathbf{x}$. A problem is *feasible* if it has a feasible solution, and *bounded* if it has an optimal solution. A non-singular $m \times m$ submatrix of A is called a *basis*. We overload the letter B to denote both the basis and the set of indices of the corresponding columns in A. A variable x_k is basic if $k \in B$. Note that, given our assumptions on the dimension of the constraint matrix, for all bases there are m basic variables and n non-basic variables. Given a basis B, and any vector $\mathbf{t}$, let $\mathbf{t}^B$ be the subvector of $\mathbf{t}$ having only the components in B. When B is clear from the context, we use N to denote the set of columns *not in* B, and use $\mathbf{t}^N$ accordingly. For a matrix A, let A^N be submatrix of A having only the columns that are not in B. The solution $\mathbf{x}$ *induced* by the basis B is defined as $x_k = 0$ for all $k \notin B$, while the values for $k \in B$ are given by the vectorial equation $x^B = B^{-1}\mathbf{b}$. A solution $\mathbf{x}$ is *basic* if there is a basis that induces $\mathbf{x}$. Given B and $k \in N$, the *reduced cost* $\overline{c_k}$ of a variable x_k is defined as $c_k - \mathbf{c}^B B^{-1} A_k$, where A_k is the k-th column of A. A solution is *dual feasible* if it correspond to a basis such that $\overline{c_k} \geq 0$ for all $k \in N$.

In our proofs we make use of the following lemma, which is particular to our canonical form.

Lemma 1.

$$A\mathbf{x} = \mathbf{b} \qquad \textit{if } \mathbf{x} \textit{ is basic} \ .$$

Proof. By splitting A into basic and non-basic columns we get $A\mathbf{x} = B\mathbf{x}^B + A^N\mathbf{x}^N = BB^{-1}\mathbf{b} + A^N\mathbf{0} = I\mathbf{b} = \mathbf{b}$. (Note that $\mathbf{x}$ might not be feasible as it could be $\mathbf{x} \ngeq \mathbf{0}$.) □

Correctness of the simplex method relies on the following well-known facts about LP problems:

- Every solution that is both feasible and dual feasible is optimal
- If there exists an optimal solution, then there exists a *basic* solution that is feasible and dual feasible (and hence optimal)

As the problems we deal with are ensured to be bounded and feasible, we assume that there exists an optimal solution. In this context, the simplex algorithm explores different bases until it finds a basis whose corresponding solution is feasible and dual feasible.

In several implementations of the algorithm the starting basis can be specified (when it is not, a default one is used). The initial basis does not need to be feasible nor dual feasible. In case the starting basis complies with both feasibilities, the simplex algorithm finishes after checking that these feasibilities are met, without any further exploration. In Subsection 2.3, we show how reachability problems correspond to LP problems. In Section 3 we show that, under a certain assumption on the model checker (Assumption 1), a basis can be obtained from the scheduler provided by the model checker. In particular, optimal schedulers yield feasible bases (Theorem 3). Under our assumption, all the bases obtained from schedulers are dual feasible (Theorem 4).

Among the different variants of the simplex method, in our experiments (Section 4) we use the *dual* simplex, which first looks for a dual-feasible basis (in the so-called *first phase*) and next tries to find a feasible one while keeping dual feasibility (in the *second phase*). This is appropriate in our case since, under our assumptions, the first phase is not needed (as formalized in Theorem 4). In contrast to the dual simplex, the *primal* simplex (or, simply, simplex) looks for a feasible basis in the first phase. As a consequence, if iterations are required (according to our results in Section 3, this is case in which the model checker fails to provide the optimal scheduler), then the primal simplex performs both phases. However, both variants can be used and, as our experiments show, the starting basis obtained from the scheduler is useful to save iterations. In the few cases in which PRISM did not provide the optimal schedulers, the dual simplex required less iterations than the primal one; both of them perform far better when starting from a basis corresponding to a near-optimal scheduler than when starting from the default basis (see Section 4).

2.3 Linear programming for Markov decision processes

Linear programming can be used to compute optimal probabilities for some of the states in the system. The set of states whose maximum (minimum, resp.) probability is 0 is first calculated using graph-based techniques [10, Sec. 4.1]. This qualitative calculation is often considered as a preprocessing step before the proper quantitative model checking. Given a set of target states U, let $\mathsf{S}^{\max 0}$ be the set of states S such that $\max_\eta \Pr_M^{s,\eta}(\text{reach}(U)) = 0$. Similarly, let $\mathsf{S}^{\min 0}$ be the set of states such that $\min_\eta \Pr_M^{s,\eta}(\text{reach}(U)) = 0$. When focusing on maximum probabilities, we write the set $\mathsf{S} \setminus (\mathsf{S}^{\max 0} \cup U)$ as $\mathsf{S}^?$ (called the set of *maybe* states), while for minimum probabilities $\mathsf{S}^?$ is $\mathsf{S} \setminus (\mathsf{S}^{\min 0} \cup U)$.

The maximum probabilities for $s \in \mathsf{S}^{\max 0}$ are 0 by definition of $\mathsf{S}^{\max 0}$. For $s \in U$ the probabilities are 1, since when starting from a state in U, the set U is reached in the initial state, regardless of the scheduler. The minimum probabilities for $s\mathsf{S}^{\min 0}$ are 0 by definition of $\mathsf{S}^{\min 0}$, and the probabilities for $s \in U$ are again 1. Next we show how to obtain the probabilities for the states in $\mathsf{S}^?$, thus covering all the states in the system.

In order to avoid order issues, we assume that the states are $\mathsf{S}^? = s_1, \cdots, s_n$ and the transitions are:

$$\mathsf{T} = \mu_1, \cdots, \mu_m \tag{5}$$

in such a way that if $s_i = \text{state}(\mu_j)$, $s_{i'} = \text{state}(\mu_{j'})$ and $i < i'$, then $j < j'$ (from Def. 1, recall that $\text{state}(\mu_i)$ is the state in which μ_i is enabled). Roughly speaking, the transitions are ordered with respect to the states in which they are enabled. From now on, we use this orderings consistently throughout the paper.

In the following theorem, the matrix $A|I$ associated to a reachability problem $\max \Pr_M^{s,\eta}(\text{reach}(U))$ is a $m \times (n+m)$ matrix whose last m columns form the identity matrix. We define of $A_{i,j}$ for the column $j \leq n$ as: $A_{i,j} = \mu_i(s_j)$ if $s_j \neq \text{state}(\mu_i)$, or $A_{i,j} = \mu_i(s_j) - 1$ if $s_j = \text{state}(\mu_i)$. The vector $\mathbf{b}$ is defined as $b_i = -\sum_{s \in U} \mu_i(s)$.

Theorem 1. *For all states $s_i \in \mathsf{S}^?$, the value $\max_\eta \Pr_M^{s_i,\eta}(\text{reach}(U))$ is the value of the variable x_i in an optimal solution of the following* LP *problem:*

$$\begin{array}{ll} \min & (\overbrace{1,\cdots,1}^{n},\overbrace{0,\cdots,0}^{m})\mathbf{x} \\ & (A \mid I^{m\times m})\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}\,. \end{array} \tag{6}$$

Analogously, the value $\min_\eta \Pr_M^{s_i,\eta}(\text{reach}(U))$ is the value of the variable x_i in an optimal solution of the following LP *problem.*

$$\begin{array}{ll} \min & -(\overbrace{1,\cdots,1}^{n},\overbrace{0,\cdots,0}^{m})\mathbf{x} \\ & (-A \mid I^{m\times m})\mathbf{x} = -\mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}\,. \end{array} \tag{7}$$

(Note that, in the constraint, the matrix A is negated, while I is not.)

This theorem is just the well-known correspondence between reachability problems and LP problems [12],[10, Section 4.2], written in our LP setting.

The variables that multiply the columns in the identity matrix are called *slack* variables in the LP literature. They are also the variables x_μ in the following notation.

Notation 1. *From now on, we identify each column $1 \leq j \leq n$ of $(A|I)$ with the state s_j, and each column $n < j \leq n+m$ with the transition μ_j. Each row i is identified with μ_i. In consequence, we write $A_{\mu,s}$ for the elements of the matrix, and x_s or x_μ for the components of the solution* $\mathbf{x}$.

3 A method for exact solutions

Our method serves as a complement to a model checker being able to:

- calculate the set $\mathsf{S}^?$, and
- give a description of a scheduler, that the model checker *considers* optimal based on finite precision calculations

We only require a weak "optimality" condition on the scheduler returned by the model checker, which we refer to as *apt*: we say that a scheduler η is apt iff $\Pr_M^{s,\eta}(\text{reach}(U)) > 0$ for all $s \in \mathsf{S}^?$. In order words, we only require the scheduler to reach U for all states that can reach it (no matter with which probability). In the case of minimum probabilities, every scheduler is apt, since if we have $\Pr_M^{s,\eta}(\text{reach}(U)) = 0$ for some η, then $s \notin \mathsf{S}^?$ (by definition of $\mathsf{S}^{\min 0}$). For the case of the maximum, the existence of an apt scheduler follows from the definition of $\mathsf{S}^?$, the scheduler η^* in (1) being a suitable witness.

Assumption 1. *We assume that the model checker is able to provide an apt scheduler, in the sense that our method is not guaranteed to return a value in case the scheduler is not apt.*

input : An MDP M and a set of states U
output: $\mathbf{x}$ such that $x_s = \max_\eta \Pr_M^{s,\eta}(\text{reach}(U))$ ($\min_\eta \Pr_M^{s,\eta}(\text{reach}(U))$, resp.) for all $s \in \mathsf{S}^?$

```
// Use model checker to get the set S? and a scheduler
(S?, η) ← reach_analysis (M, U);
ℒ ← construct_problem (M, S?);
B_η ← construct_basis (ℒ, η);
start_simplex_solver (ℒ, B_η) ;
if the exact simplex solver finishes in one iteration then
    return arg min_x ℒ, obtained from the solver;
else if the solver performs several iterations then                    // η is not optimal
    return arg min_x ℒ, obtained from the solver once it finishes;
    // Or interrupt the solver and change the model checker parameters
else if the solver reports that the basis is singular then
    // For the minimum, this case cannot happen
    error η is not apt;
end
```

Algorithm 1: Method to get exact solutions

Our method is described in the Algorithm 1. The function `construct_problem` constructs the LP problems (6) and (7). Given η, the basis B_η obtained by `construct_basis` is defined as

$$s \in B_\eta, \quad \text{for all } s \in \mathsf{S}^? \qquad\qquad x_\mu \in B_\eta \iff \eta(\text{state}(\mu)) \neq \mu \ . \tag{8}$$

Roughly speaking, the basis contains all states, and all the transitions that are *not* chosen by η. Sometimes (particularly in the proof of Theorem 4) we write $B_{M',\eta}$ to make it clear that the basis belongs to an MDP M'.

The rest of this section is devoted to prove the correctness of the algorithm, in the sense made precise by the following theorem (which is proven later).

Theorem 2. *If the algorithm returns a value, then the value corresponds to the* ***output*** *specification. Moreover, if the scheduler η provided by the model checker is apt, then the matrix defined by* (8) *is a basis, and the algorithm returns optimum values from the* LP *solver. If the scheduler provided by the model checker is optimum as in* (1)*, then the basis in* (8) *is both feasible and dual feasible.*

Recall from Subsection 2.2 that the simplex algorithm stops as soon as it finds a solution that is feasible and dual feasible. Hence, the fact that an optimal scheduler yields a basic, feasible and dual feasible solution causes the simplex solver to stop as soon as the feasibility checks are finished.

The rest of this section is devoted to prove Theorem 2. In our proofs we resort to the following definitions and lemmata. The first definition uses indices as explained in Notation 1.

Definition 4. Given a scheduler η, we write the set of transitions complying with $\eta(\text{state}(\mu)) = \mu$ as $\mathsf{T}_\eta = \{\mu^1, \cdots, \mu^n\}$, and we assume that this ordering respects the ordering in (5). We define C^η to be the $n \times n$ matrix whose elements are as $C^\eta_{i,j} = \mu^i(s_j)$. Consider the matrix A in (6). We define $(A \downarrow \eta)$ to be the $n \times n$ submatrix of A comprising all the rows $\mu \in \mathsf{T}_\eta$ and the columns s for all $s \in \mathsf{S}^?$.

Lemma 2. *The transitions $\mu^i \in \mathsf{T}_\eta$ comply with* $\text{state}(\mu^i) = s_i$ *for all $s_i \in \mathsf{S}^?$. In consequence, $\eta(s_i) = \mu^i$.*

Proof. Note that since the order in T_η respects the order in (5), we have that the sequence state$(\mu^1),\cdots,$ state(μ^n) is a sequence of states $s_{j_1},\cdots,s_{j_n}$ with $j_1 \leq \cdots \leq j_n$. Since there are n states, and for each state s we have exactly one transition μ such that $\eta(s)=\mu$, it must be $s_{j_1}=s_1,\cdots,s_{j_n}=s_n$. This implies state$(\mu^i)=s_{j_i}=s_i$ as desired. Using this equality and $\mu^i \in \mathsf{T}_\eta$ we have $\eta(s_i)=\eta(\text{state}(\mu^i))=\mu^i$. □

Lemma 3. *For all η, we have $(A\downarrow\eta)=C^\eta-I$.*

Proof. By definition of $(A\downarrow\eta)$ and the definition of the matrix A in (6) we have $(A\downarrow\eta)_{i,j}=A_{\mu^i,s_j}=\mu^i(s_j)-Q_{i,j}$, where $Q_{i,j}=1$ if state$(\mu^i)=s_j$, or otherwise $Q_{i,j}=0$. By Lemma 2, we have state$(\mu^i)=s_j$ iff $i=j$. Hence $Q_{i,j}$ is the identity matrix and $(A\downarrow\eta)_{i,j}=\mu^i(s_j)-I_{i,j}=C^\eta_{i,j}-I_{i,j}$, which completes the proof. □

The matrix $(A\downarrow\eta)$ happens to be very important in our proofs. We profit from the fact that it is non-singular provided that η is apt.

Lemma 4. *For all apt η, the matrix $(A\downarrow\eta)$ is non-singular.*

Proof. Suppose, towards a contradiction, that there exists $\mathbf{x}\neq\mathbf{0}$ such that $(A\downarrow\eta)\mathbf{x}=\mathbf{0}$. Then, by Lemma 3, we have $(C^\eta-I)\mathbf{x}=\mathbf{0}$, which implies $C^\eta\mathbf{x}=\mathbf{x}$ and hence $(C^\eta)^z\mathbf{x}=\mathbf{x}$ for all $z\geq 0$. We arrive to a contradiction by showing that for all j there exists z such that

$$|((C^\eta)^z\mathbf{x})_j| < \max_{s'}|x_{s'}|\,. \tag{9}$$

In particular, for $q=\arg\max_{s'}|x_{s'}|$ this yields $|((C^\eta)^z\mathbf{x})_q|<|x_q|$, which contradicts $(C^\eta)^z\mathbf{x}=\mathbf{x}$.

Now we prove (9). Since η is apt, from every $s_j\in\mathsf{S}^?$ there exists a path $\rho\in\text{Paths}(s_j,\eta)$ with last$(\rho)\in U$, such that all the states previous to last(ρ) are not in U. We prove that z can be taken to be len(ρ). We proceed by induction on the length of ρ. If len$(\rho)=1$, by Lemma 2 we have $\eta(s_j)(u)=\mu^j(u)>0$ for some $u\in U$, and hence[2] $\sum_{t\in\mathsf{S}^?}\mu^j(t)<1$. Taking $z=1$ we obtain

$$|(C^\eta\mathbf{x})_j| = |\sum_{t\in\mathsf{S}^?}\mu^j(t)x_t| \leq \sum_{t\in\mathsf{S}^?}\mu^j(t)|x_t| \leq \sum_{t\in\mathsf{S}^?}\mu^j(t)\max_{s'}|x_{s'}| < \max_{s'}|x_{s'}|\,,$$

which proves that we can take $z=1=\text{len}(\rho)$. The last strict inequality holds only if $\max_{s'}|x_{s'}|>0$, which follows from $\mathbf{x}\neq\mathbf{0}$.

If len$(\rho)=l+1$, there exists $s_q\in\mathsf{S}^?$ such that $\mu^j(s_q)>0$ and q reaches U in l steps. The inductive hypothesis holds for q, and hence $|((C^\eta)^l\mathbf{x})_q|<\max_{s'}|x_{s'}|$, from which we obtain:

$$\begin{aligned}
|((C^\eta)^{l+1}\mathbf{x})_j| = |(C^\eta(C^\eta)^l\mathbf{x})_j| &\leq \sum_{t\in\mathsf{S}^?\setminus\{s_q\}}\mu^j(t)\,|((C^\eta)^l\mathbf{x})_t| \;+\; \mu^j(s_q)\,|((C^\eta)^l\mathbf{x})_q| \\
&= \sum_{t\in\mathsf{S}^?\setminus\{s_q\}}\mu^j(t)\,|x_t| \;+\; \mu^j(s_q)\,|((C^\eta)^l\mathbf{x})_q| \leq \sum_{t\in\mathsf{S}^?\setminus\{s_q\}}\mu^j(t)\max_{s'}|x_{s'}| \;+\; \mu^j(s_q)\,|((C^\eta)^l\mathbf{x})_q| \\
&< \sum_{t\in\mathsf{S}^?\setminus\{s_q\}}\mu^j(t)\max_{s'}|x_{s'}| \;+\; \mu^j(s_q)\max_{s'}|x_{s'}| \leq \max_{s'}|x_{s'}|
\end{aligned}$$

This finishes the proof of (9). Assuming that $(A\downarrow\eta)\mathbf{x}=\mathbf{0}$ for some $\mathbf{x}\neq\mathbf{0}$, we derived (9), which contradicts $(C^\eta)^z\mathbf{x}=\mathbf{x}$ for all $z\geq 0$, thus finishing the proof. □

[2] The result for discounted MDPs does not use $\mathsf{S}^?$ as the analogous of this sum is always less than 1 due to the discounts

Lemma 5. *For all apt schedulers η, the basis defined in (8) is non-singular.*

Proof. We show that the equation $B_\eta \mathbf{x} = \mathbf{0}$ holds only if $\mathbf{x} = \mathbf{0}$. Note that the vector $\mathbf{x}$ has one component for each column of the basis, that is, one component for each state in $\mathsf{S}^?$ (called x_s), and one component for each transition such that $\eta(\text{state}(\mu)) \neq \mu$ (called x_μ). The matrix equation $B_\eta \mathbf{x} = \mathbf{0}$ corresponds to m equations, one for each transition. If $\mu \in B_\eta$, since $t \in B_\eta$ for all $t \in \mathsf{S}^?$, the equation corresponding to μ is

$$\sum_{t \in \mathsf{S}^?} A_{\mu,t} x_t + x_\mu = 0 \,. \tag{10}$$

If $\mu \notin B_\eta$, the corresponding equation is

$$\sum_{t \in \mathsf{S}^?} A_{\mu,t} x_t = 0 \,. \tag{11}$$

(Note that the sum term x_μ has disappeared. This corresponds to the fact that the column corresponding to x_μ is not in the basis.) Since the transitions $\mu \notin B_\eta$ are those such that $\eta(\text{state}(\mu)) = \mu$, the set of equations (11) is equivalent to $(A \downarrow \eta)\mathbf{s} = \mathbf{0}$, where $\mathbf{s}$ is the subvector of $\mathbf{x}$ having only the components corresponding to states. In consequence, if $B_\eta \mathbf{x} = \mathbf{0}$ holds, then in particular $(A \downarrow \eta)\mathbf{s} = \mathbf{0}$ and, since η is apt, by Lemma 4 it must be $\mathbf{s} = \mathbf{0}$, that is, $x_t = 0$ for all $t \in \mathsf{S}^?$. Using this in (10) we have $x_\mu = 0$ for all $\mu \in B_\eta$. We have proven $x_j = 0$ for every component j of $\mathbf{x}$, thus showing $\mathbf{x} = \mathbf{0}$. □

Theorem 3. *If a scheduler is optimal as in* (1) *(or* (2)*, resp.) then the solution induced by the basis B_η is feasible.*

Proof. Let $\mathbf{x}$ be the solution induced by B_η for some optimal η. By Lemma 1, we need to prove $\mathbf{x} \geq \mathbf{0}$. We prove this inequality by showing that $x_s = \Pr_M^{s,\eta}(\text{reach}(U)) \geq 0$ for all s and $x_\mu \geq 0$ for all μ.

Since in B_η the variables $x_\mu \in \mathsf{T}_\eta$ are non basic, in the solution $\mathbf{x}^\eta$ induced by B_η we have $x_\mu = 0$ for all $\mu \in \mathsf{T}_\eta$. Then, using Lemma 1 for our particular constraint matrix $A|I$, we obtain

$$x_s = \sum_{t \in \mathsf{S}^?} \eta(s)(t)\, x_t \; + \sum_{t \in U} \eta(s)(t) \,. \tag{12}$$

This is equivalent to $(A \downarrow \eta)\mathbf{x} = \mathbf{q}$ for some vector $\mathbf{q}$. By Lemma 4, there exists exactly one $\mathbf{x}$ satisfying (12). Let v_s^η be $\Pr_M^{s,\eta}(\text{reach}(U))$. A classic result for MDPs (see, for instance, [10, Section 4.2], [2, Theorem 3.10]) states that, for an optimal scheduler η, it holds

$$v_s^\eta = \max_{\mu \in \text{en}(s)} \sum_{t \in \mathsf{S}^?} \mu(t) v_t^\eta + \sum_{t \in U} \mu(t) \tag{13}$$

and

$$\eta(s) \in \arg\max_{\mu \in \text{en}(s)} \sum_{t \in \mathsf{S}^?} \mu(t) v_t^\eta + \sum_{t \in U} \mu(t) \,.$$

for all states s. From the last two equations:

$$v_s^\eta = \sum_{t \in \mathsf{S}^?} \eta(s)(t)\, v_t^\eta \; + \sum_{t \in U} \eta(s)(t) \,.$$

This is equivalent to $(A\downarrow\eta)\mathbf{v}^\eta = \mathbf{q}$ as before. After (12) we have seen that this equation has a unique solution, and so $x_s = v_s^\eta$ for all $s \in \mathsf{S}^?$. By (13) we have

$$x_s \geq \sum_{t\in\mathsf{S}^?} \mu(t)\, x_t \;+\; \sum_{t\in U} \mu(t) \tag{14}$$

for all $s \in \mathsf{S}^?$, $\mu \in \mathrm{en}(s)$. Applying Lemma 1 to our particular constraint matrix $A|I$, we have

$$x_\mu = x_s - \sum_{t\in\mathsf{S}^?} \mu(t) x_t - \sum_{t\in U} \mu(t)\,.$$

Hence, $x_\mu \geq 0$ for all μ by (14). In conclusion, $x_s = v_s^\eta \geq 0$ for all $s \in \mathsf{S}^?$ and $x_\mu \geq 0$ for all μ. Then, the solution $\mathbf{x}$ induced by B_η is feasible.

For the case of the minimum, the analogue of (13) is:

$$v_s^\eta = \min_{\mu\in\mathrm{en}(s)} \sum_{t\in\mathsf{S}^?} \mu(t) v_t^\eta + \sum_{t\in U} \mu(t) \tag{15}$$

The fact that the equation $(A\downarrow\eta)\mathbf{v}^\eta = \mathbf{q}$ has a unique solution again yields $x_s = \mathbf{v}^\eta$. For x_μ, using the constraint matrix $-A|I$ for the minimum and (15) we obtain

$$x_\mu = -x_s + \sum_{t\in\mathsf{S}^?} \mu(t) x_t + \sum_{t\in U} \mu(t) = \sum_{t\in U} \mu(t) + \sum_{t\in\mathsf{S}^?} \mu(t) - x_t \geq 0\,.$$

□

Theorem 4. *Given an apt scheduler η, the solution induced by the basis B_η is dual feasible. (For the definition of* dual feasible *see Subsection 2.2.)*

Proof. First we find a matrix expression for B_η^{-1}. Suppose we reorder the rows of B_η so that the rows corresponding to transitions in the basis occur first. The resulting matrix is

$$B'_\eta = \left(\begin{array}{c|c} A' & I^{(m-n)\times(m-n)} \\ \hline (A\downarrow\eta) & 0 \end{array}\right)$$

where A' is a submatrix of B_η. We can write

$$B'_\eta = PB_\eta \tag{16}$$

where P is a permutation matrix. In order to find the inverse of B'_η we pose the following matrix equation:

$$\left(\begin{array}{c|c} A' & I^{(m-n)\times(m-n)} \\ \hline (A\downarrow\eta) & 0 \end{array}\right)\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c|c} A'A_{11}+A_{21} & A'A_{12}+A_{22} \\ \hline (A\downarrow\eta)A_{11} & (A\downarrow\eta)A_{12} \end{array}\right) = I = \left(\begin{array}{c|c} I^{n\times n} & 0 \\ \hline 0 & I^{(m-n)\times(m-n)} \end{array}\right)$$

These equations, suggest that we can take $A_{11} = 0$, and hence $A_{21} = I$. Moreover, it must be $A_{12} = (A\downarrow\eta)^{-1}$ (which exists by Lemma 5) and hence $A_{22} = -A'(A\downarrow\eta)^{-1}$. The equation below can be easily checked by verifying that $B_\eta'^{-1}B'_\eta = I$

$$B_\eta'^{-1} = \left(\begin{array}{c|c} 0^{n\times(m-n)} & (A\downarrow\eta)^{-1} \\ \hline I^{(m-n)\times(m-n)} & -A'(A\downarrow\eta)^{-1} \end{array}\right) \tag{17}$$

Next, we use (17) to show that the reduced costs depend only on the constraint coefficients of the transitions chosen by the scheduler.

We consider first the case of the maximum. Recall that our constraint matrix is $A|I$ and the costs c_μ associated to the transitions variables are 0 for all μ (see (6)). According to the definition of reduced cost (see Subsection 2.2), to prove dual feasibility we need to show $-\mathbf{c}^{B_\eta}B_\eta^{-1}I_\mu \geq 0$ for all $\mu \notin B_\eta$, where I_μ is the column of the identity matrix corresponding to μ. From (16), we have $B_\eta^{-1} = B_\eta'^{-1}P$, and hence our inequality is $-\mathbf{c}^{B_\eta}B_\eta'^{-1}PI_\mu \geq 0$. Since P is a permutation matrix, we know that PI_μ is a column of the identity matrix, say $I_{k(\mu)}$. Given our costs in (6), and given the definition of B_η, we have that $\mathbf{c}^{B_\eta}$ is the vector $(\overbrace{1,\cdots,1}^{n},\overbrace{0,\cdots,0}^{m-n})$, and hence from (17) we get $\mathbf{c}^{B_\eta}B_\eta'^{-1} = (\mathbf{0}^{1\times(m-n)}, \mathbf{1}^{1\times n}(A\downarrow\eta)^{-1})$. In conclusion, we have proven

$$-\mathbf{c}^{B_\eta}B_\eta^{-1}I_\mu = -(\mathbf{0}^{1\times(m-n)}, \mathbf{1}^{1\times n}(A\downarrow\eta)^{-1})\, I_{k(\mu)}\ , \tag{18}$$

and we must prove that this number is greater than or equal to 0 for all $\mu \notin B_\eta$.

Whenever $k(\mu) \leq m-n$, the result holds since (18) is 0.

In case $k(\mu) > m-n$, we prove the result using the fact that these values depend only on the transitions chosen by η. In fact, given the MDP M and the scheduler η, if we write (18) for the Markov chain $M\downarrow\eta$ (see Def. 3), we obtain

$$-\mathbf{c}^{B_{(M\downarrow\eta),\eta}}\, B^{-1}_{(M\downarrow\eta),\eta}\, I_\mu \;=\; -\mathbf{1}^{1\times n}(A\downarrow\eta)^{-1}\, I_\mu \tag{19}$$

for all $\mu \notin B_{(M\downarrow\eta),\eta}$. Note that for $M\downarrow\eta$ there is no need to reorder (as there are no transitions in the basis) and so $\mu = k(\mu)$. Given that all the transitions $M\downarrow\eta$ are chosen by η, the basis $B_{(M\downarrow\eta),\eta}$ contains all the states and no transitions. In this equation, I_μ can be *any* column of $I^{n\times n}$ (again, due to the fact that there are no transitions in the basis).

Suppose, towards a contradiction, that (18) is less than 0 for some $k(\mu) > m-n$. This is equivalent to $-\mathbf{1}^{1\times n}(A\downarrow\eta)^{-1}\, I_{k(\mu)-(m-n)} < 0$. By (19) we have $-\mathbf{1}^{1\times n}(A\downarrow\eta)^{-1}\, I_{\mu'} < 0$ for some μ' in $M\downarrow\eta$. Then, the solution induced by the basis is not dual feasible for the problem associated to $M\downarrow\eta$. As there is at least one optimal basic and dual feasible solution (the one found by simplex method), there exists an optimal solution $\mathbf{x}^C$ such that the corresponding basis B^C is not $B_{(M\downarrow\eta),\eta}$. As in $M\downarrow\eta$ there exists only one basis containing all states (namely $B_{(M\downarrow\eta),\eta}$), there exists $s \notin B^C$. In consequence, we have $x^C_s = 0$. Since x^C is optimal, by Theorem 1, we obtain $\Pr^{s,\eta}_{M\downarrow\eta}(\text{reach}(U)) = 0$, from which (3) yields $\Pr^{s,\eta}_{M}(\text{reach}(U)) = 0$. This contradicts the fact that η is apt.

The proof for the case of the minimum is completely analogous: despite the differences in the constraints and the cost vector, the reduced costs in (18) are the same as before:

$$-\mathbf{c}^{B_\eta}B_\eta^{-1}I_\mu = -(-\mathbf{0}^{1\times(m-n)}, -\mathbf{1}^{1\times n}(-(A\downarrow\eta)^{-1}))\, I_{k(\mu)} = -(\mathbf{0}^{1\times(m-n)}, \mathbf{1}^{1\times n}(A\downarrow\eta)^{-1})\, I_{k(\mu)}\ .$$

These values again coincide with the ones in a system having only the transitions chosen by η. □

Proof (of Theorem 2). If the algorithm returns a value, then it is $\arg\min_{\mathbf{x}} \mathscr{L}$, where $\mathscr{L}$ is the problem (6) (or (7) for the minimum). Hence, by Theorem 1, the returned value coincides with the **output** specification. We have that if η is apt, then B_η is a basis by Lemma 5. As a consequence, the algorithm never enters the branch in line 10, and so the result is returned. The termination in a single iteration is a consequence of the fact that the solution corresponding to an optimal scheduler η is both feasible (Theorem 3) and dual feasible (Theorem 4). □

4 Experimental results

Implementation. We implemented our method by extending the model checker PRISM [11], using the LP library `glpk` [3]. We compiled `glpk` using the library for arbitrary precision `gmp`. We needed to modify the code of `glpk`: although there is a solver function that uses exact arithmetic internally, this function does not allow us to retrieve the exact value. Aside from these changes to `glpk` and some additional code scattered around the PRISM code (in order to gather information about the scheduler), the specific code for implementing our method is less than 300 lines long. With these modifications, PRISM is able to print the numerator and the denominator of the probabilities calculated.

Our implementation works as follows: in the first step, we use the value iteration already implemented in PRISM to calculate a candidate scheduler. In the next step, the LP problem is constructed by iterating over each state: for each transition enabled, the corresponding probabilities are inserted in the matrix. The basis is constructed along this process: when a transition is considered, the description of the scheduler (implemented as an array) is queried about whether this transition is the one chosen by the scheduler. Next we solve the LP problem. For the reasons explained in Subsection 2.2, in Algorithm 1 we use the dual simplex method, except when we compare it to the primal one. The reader familiar with `glpk` might notice that the dual variant is not implemented under exact arithmetic on `glpk`: to overcome this, instead of providing `glpk` with the original problem, we provided the dual problem and retrieved the values of the dual variables (the dual problem is obtained by providing the transpose of the constraint matrix and by negating the cost coefficients, and so it does not affect the running time).

The experiments were carried out on an Intel i7 @3.40Ghz with 8Gb RAM, running Windows 7.

Case studies. We studied three known models available from the PRISM benchmark suite [1], where the reader can look for matters not explained here (for instance, details about the parameters of each model). For the parameters whose values are not specified here, we use the default values. In the IEEE 802.11 Wireless LAN model, two stations use a randomised exponential backoff rule to minimise the likelihood of transmission collision. The parameter N is the number of maximum backoffs. We compute the maximum probability that the backoff counters of both stations reach their maximum value. The second model concerns the consensus algorithm for N processes of Aspnes & Herlihy [4], which uses shared coins. We calculate the maximum probability that the protocol finishes without an agreement. The parameter K is used to bound a shared counter. Our third case study is the IEEE 1394 FireWire Root Contention Protocol (using the PRISM model which is based on [13]). We calculate the minimum probability that a leader is elected before a deadline of D time units.

Linear programming versus Algorithm 1. Table 1 allows us to compare (primal and dual) simplex starting from a default basis, against Algorithm 1, which provides a starting basis from a candidate scheduler. Aside from the construction of the MDP from the PRISM language description (which is the same either using LP or Algorithm 1, and is thus disregarded in our comparisons), the steps in our implementation are: (1) perform value iteration to obtain a candidate scheduler; (2) construct the LP problem; (3) solve the problem in exact arithmetic in zero or more iterations (the latter is the case in which the scheduler is not optimal). All these times are shown in Table 1, as well as its sum, expressed in seconds. The experiments for LP were run with a time-out of one hour (represented with a dash).

Our method always outperforms the naive application of LP. The case with the lowest advantage is Consensus (3,5), and still our method takes less than $1/6$ of the time required by dual simplex.

With respect to the time devoted to exact arithmetic in Algorithm 1, in all cases the simplex under

Model	Parameters	$n = \|S^?\|$	m	Time (seconds)					
				LP without Alg. 1		Algorithm 1			
				Primal	Dual	Value iter.	LP constr.	Dual simplex	Total
Wlan	3	2529	96302	19.53	11.76	0.36	0.05	0.03	0.44
(N)	4	5781	345000	110.32	61.83	2.30	0.21	0.06	2.57
	5	12309	1295218	535.76	326.64	14.93	1.32	0.15	16.40
Consensus	3,3	3607	3968	251.74	35.32	2.93	0.04	0.15	3.12
(N, K)	3,4	4783	5216	488.84	64.00	6.47	0.06	0.58	7.11
	3,5	5959	6464	1085.70	105.36	12.74	0.06	1.87	14.67
	4,1	11450	12416	-	432.98	2.88	0.11	0.19	3.18
	4,2	21690	22656	-	1951.91	20.41	0.23	0.37	21.01
	4,3	31930	32896	-	-	59.73	0.49	0.58	60.80
	4,4	42170	43136	-	-	134.62	0.64	0.78	136.04
	4,5	52410	53376	-	-	246.90	0.91	0.96	248.77
Firewire	200	1071	80980	4.50	2.65	0.28	0.04	0.01	0.33
(D)	300	23782	213805	-	1314.32	2.89	1.04	0.24	4.17
	400	81943	434364	-	-	11.05	8.74	0.88	20.67

Table 1: Comparison of primal and dual simplex starting from a default basis against Algorithm 1

exact arithmetic takes a fraction of the time spent by the other operations of the algorithm (namely, to perform value iteration and to construct the LP problem). In Consensus (3,5), the simplex algorithm takes less than $1/6$ of the time devoted to the other operations. In all other cases the ratio is even lower.

The greatest number found was 28821938103543398400, the denominator in the solution of Firewire 400. It needs 65 bits to be stored. The computations were performed using 32 bit libraries, and so the exact arithmetic computations used around 3 words in the worst case (which is not really a challenge for an arbitrary precision library). We can conclude that, even for systems with more than 10000 states (up to 80000, in our experiments), the overhead introduced by exact arithmetic is manageable.

Suboptimal schedulers as suboptimal bases. Other than measuring whether the calculation is reasonably quick in case the scheduler from PRISM is optimal, a secondary measuring concerns how close is the basis to an optimal one in case the scheduler provided by PRISM is *not* optimal.

Except in cases Consensus (3,·), simplex stopped after 0 iterations, thus indicating that PRISM was able to find the optimal scheduler. For optimal schedulers there is no difference between using primal or dual simplex in Algorithm 1 (we ran the experiments and the running time of the simplex variants differed by at most 0.05 seconds).

The probabilities obtained in each step of the value iteration converge to those of an optimal scheduler. Given a threshold ε, value iteration stops only after $|x_s - x'_s| \leq \varepsilon$ for all s, where $\mathbf{x}$ and $\mathbf{x}'$ are the vectors obtained in the last two iterations.

In Table 2 we compare the amount of iterations and the time spent by primal and dual simplex for schedulers obtained using different thresholds. We considered only the cases Consensus (3,·), as in other cases the scheduler returned by PRISM was optimum except for gross thresholds above 0.05, which are rarely used in practice (the default ε in PRISM is 10^{-6}). In addition to the default value, we considered representatives the value 10^{-7} (since 10^{-8} already yields the exact solution for $(3,3)$ in the dual case: a value smaller than 10^{-7} would have yielded uninteresting numbers for this case) and the value 10^{-16}, since in $(3,5)$ the scheduler does not improve beyond such threshold. In fact, for 10^{-16} the result is the same as for 10^{-323}, and 10^{-324} is not a valid `double`. In Java, the type `double` corresponds to a IEEE 754 64-bit floating point.

In consequence, we have one case (namely, Consensus (3,5)), where PRISM cannot find the worst-

	Primal						Dual					
	Iterations			Time (seconds)			Iterations			Time (seconds)		
ε (10^{-n})	6	7	16	6	7	16	6	7	16	6	7	16
Consensus (3,3)	187	134	0	3.43	2.43	0.10	10	6	0	0.19	0.15	0.10
Consensus (3,4)	2497	6278	0	74.42	202.58	0.14	37	28	0	0.63	0.51	0.13
Consensus (3,5)	4990	4340	1239	190.53	160.44	49.487	94	61	6	1.93	1.24	0.25

Table 2: Time spent when the starting basis is not optimal

case scheduler for any `double` threshold (and thus should be recoded to use another arithmetic primitives to get exact results), while our method is able to calculate exact results using less than two seconds after value iteration, as shown in Table 1.

For Consensus (3,·) we see that dual simplex performs betters than primal simplex. Consensus (3,4) shows that the primal simplex can behave worse when starting from B_η than the dual simplex starting from the default basis (compare with the corresponding row in Table 1). Moreover, it can be the case that it takes *more* time as the threshold decreases (note that, in contrast, in Consensus (3,5) the time decreases with the threshold, as expected). This suggests that the dual variant should be preferred over the primal.

Comparing against Table 1, we see that, for each variant of the simplex method, starting from the basis B_η results in a quicker calculation than starting from the default basis.

5 Discussion and further work

Linear programming versus policy iteration. It is known that the dual simplex method applied for discounted MDPs is just the same as *policy iteration* (for an introduction to this method see [10]) seen from a different perspective. Indeed, this has been used to obtain complexity bounds (see [14]). Theorems 3 and 4 establish for undiscounted MDPs the same correspondence between basis and schedulers as known for the discounted case, and as a consequence the dual simplex is policy iteration disguised, also in the undiscounted case.

Even without considering the results in this paper at all, exact solutions can also be calculated by implementing policy iteration with exact arithmetic as, in each iteration, the method calculates the probabilities corresponding to a scheduler and checks whether they can be improved by another scheduler. Roughly speaking, if the calculation and the check are performed using exact arithmetic, then the result is also exact.

Despite this existing alternative, the correspondence between bases and schedulers we presented in this paper allows to obtain an exact solution by using LP solvers, thus profiting from all the knowledge concerning LP problems (and from existing implementations such as `glpk`).

Complexity. To the best of our knowledge, the precise complexity of the simplex method in our case is unknown. There are recent results for the simplex applied to similar problems. For instance, in [14] it is proven that simplex is strongly polynomial for discounted MDPs. Nevertheless, [9] shows an exponential lower bound to calculate rewards in the undiscounted case. Unfortunately (or not, as there is still hope that we can prove the time to be polynomial in our case), the construction used in [9] cannot be carried out easily to our setting, as some of the rewards in the construction are negative (and the equivalent to the rewards in our setting are the sums $\sum_{t\in U} \mu(t)$).

Further work. In the comparison of our method against LP, we considered only the simplex method, as `glpk` only implements this method in exact arithmetic. The feasibility/applicability of other algorithms to solve LP problems using exact arithmetic is yet to be studied.

The fact that the probabilities obtained are exact allows to prove additional facts about the system under consideration. For instance, the exact values can be used in correctness certificates, or be the input of automatic theorem provers, if they require exact values to prove some other properties of the system. We plan to concentrate on these uses of exact probabilities.

Acknowledgements. The author is grateful to David Parker, Vojtech Forejt and Marta Kwiatkowska for useful comments and proofreading.

References

[1] `http://www.prismmodelchecker.org/benchmarks/`.

[2] Luca de Alfaro (1998): *Formal Verification of Probabilistic Systems*. Thesis CS-TR-98-1601, Stanford University, Department of Computer Science.

[3] Copyright by Andrew Makhorin: *glpk*. `http://www.gnu.org/s/glpk/`.

[4] James Aspnes & Maurice Herlihy (1990): *Fast Randomized Consensus Using Shared Memory*. J. Algorithms 11(3), pp. 441–461. Available at `http://dx.doi.org/10.1016/0196-6774(90)90021-6`.

[5] Andrea Bianco & Luca de Alfaro (1995): *Model Checking of Probabalistic and Nondeterministic Systems*. In: *FSTTCS 1995*, pp. 499–513. Available at `http://dx.doi.org/10.1007/3-540-60692-0_70`.

[6] Stephen P. Bradley, Arnoldo C. Hax & Thomas L. Magnanti (1977): *Applied Mathematical Programming*. Addison-Wesley.

[7] Costas Courcoubetis & Mihalis Yannakakis (1990): *Markov Decision Processes and Regular Events (Extended Abstract)*. In Mike Paterson, editor: *ICALP*, *Lecture Notes in Computer Science* 443, Springer, pp. 336–349. Available at `http://dx.doi.org/10.1007/BFb0032043`.

[8] F. D'Epenoux (1963): *A probabilistic production and inventory problem*. *Management Science* 10, pp. 98–108.

[9] John Fearnley (2010): *Exponential Lower Bounds for Policy Iteration*. In: *ICALP 2010 (2)*, pp. 551–562. Available at `http://dx.doi.org/10.1007/978-3-642-14162-1_46`.

[10] Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman & David Parker (2011): *Automated Verification Techniques for Probabilistic Systems*. In: *SFM*, pp. 53–113. Available at `http://dx.doi.org/10.1007/978-3-642-21455-4`.

[11] M. Kwiatkowska, G. Norman & D. Parker (2011): *PRISM 4.0: Verification of Probabilistic Real-time Systems*. In G. Gopalakrishnan & S. Qadeer, editors: *CAV 2011*, *LNCS* 6806, Springer, pp. 585–591. Available at `http://dx.doi.org/10.1007/978-3-642-22110-1_47`.

[12] Martin L. Puterman (1994): *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st edition. John Wiley & Sons, Inc., New York, NY, USA.

[13] M. Stoelinga & F. Vaandrager (1999): *Root Contention in IEEE 1394*. In J.-P. Katoen, editor: *Proc. 5th International AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)*, *LNCS* 1601, Springer, pp. 53–74. Available at `http://dx.doi.org/10.1007/3-540-48778-6_4`.

[14] Yinyu Ye (2011): *The Simplex and Policy-Iteration Methods Are Strongly Polynomial for the Markov Decision Problem with a Fixed Discount Rate*. *Mathematics of Operations Research* 36(4), pp. 593–603. Available at `http://dx.doi.org/10.1287/moor.1110.0516`.

Measuring Progress of Probabilistic LTL Model Checking

Elise Cormie-Bowins* and Franck van Breugel†
DisCoVeri Group, Department of Computer Science, York University,
4700 Keele Street, Toronto, ON, M3J 1P3, Canada

Recently, Zhang and Van Breugel introduced the notion of a progress measure for a probabilistic model checker. Given a linear-time property ϕ and a description of the part of the system that has already been checked, the progress measure returns a real number in the unit interval. The real number captures how much progress the model checker has made towards verifying ϕ. If the progress is zero, no progress has been made. If it is one, the model checker is done. They showed that the progress measure provides a lower bound for the measure of the set of execution paths that satisfy ϕ. They also presented an algorithm to compute the progress measure when ϕ is an invariant.

In this paper, we present an algorithm to compute the progress measure when ϕ is a formula of a positive fragment of linear temporal logic. In this fragment, we can express invariants but also many other interesting properties. The algorithm is exponential in the size of ϕ and polynomial in the size of that part of the system that has already been checked. We also present an algorithm to compute a lower bound for the progress measure in polynomial time.

1 Introduction

Due to the infamous state space explosion problem, model checking a property of source code that contains randomization often fails. In many cases, the probabilistic model checker simply runs out of memory without reporting any useful information. In [11], Zhang and Van Breugel proposed a progress measure for probabilistic model checkers. This measure captures the amount of progress the model checker has made with its verification effort. Even if the model checker runs out of memory, the amount of progress may provide useful information.

Our aim is to develop a theory that is applicable to probabilistic model checkers in general. Our initial development has been guided by a probabilistic extension of the model checker Java PathFinder (JPF) [9]. This model checker can check properties, expressed in linear temporal logic (LTL), of Java code containing probabilistic choices.

We model the code under verification as a probabilistic transition system (PTS), and the systematic search of the system by the model checker as the set of explored transitions of the PTS. We focus on linear-time properties, in particular those expressed in LTL. The progress measure is defined in terms of the set of explored transitions and the linear-time property under verification. The progress measure returns a real number in the interval $[0,1]$. The larger this number, the more progress the model checker has made with its verification effort.

Zhang and Van Breugel showed that their progress measure provides a lower bound for the measure of the set of execution paths that satisfy the linear-time property under verification. If, for example, the progress is 0.9999, then the probability that we encounter a violation of the linear-time property when we run the code is at most 0.0001. Hence, despite the fact the model checker may fail by running out of memory, the verification effort may still be a success by providing an acceptable upper bound on the probability of a violation of the property.

*Supported by an Ontario Graduate Scholarship.
†Supported by the Natural Sciences and Engineering Research Council of Canada and the Leverhulme Trust.

Preliminary Report. Final version to appear in:
QAPL 2012

The two main contributions of this paper are

1. a characterization of the progress measure for a positive fragment of LTL. This fragment includes invariants, and most examples found in, for example, [2, Section 5.1] can be expressed in this fragment. This characterization forms the basis for an algorithm to compute the progress measure.
2. a polynomial time algorithm to compute a lower bound for the progress measure for the positive fragment of LTL. The lower bound is tight for invariants, that is, this algorithm computes the progress for invariants.

2 A Progress Measure

In this section, we review some of the key notions and results of [11]. We represent the system to be verified by the probabilistic model checker as a probabilistic transition system.

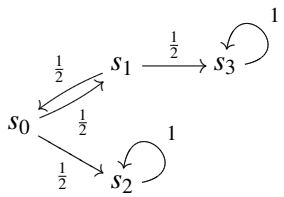
Definition 1 *A probabilistic transition system is a tuple* $\langle S, T, AP, s_0, \text{source}, \text{target}, \text{prob}, \text{label}\rangle$ *consisting of*

- *a countable set S of states,*
- *a countable set T of transitions,*
- *a set AP of atomic propositions,*
- *an initial state* s_0*,*
- *a function* $\text{source} : T \to S$*,*
- *a function* $\text{target} : T \to S$*,*
- *a function* $\text{prob} : T \to (0,1]$*, and*
- *a function* $\text{label} : S \to 2^{AP}$

such that

- $s_0 \in S$ *and*
- *for all* $s \in S$, $\sum\{\, \text{prob}(t) \mid \text{source}(t) = s \,\} = 1$.

Example 2 *The probabilistic transition system* $\mathscr{S}$ *depicted by*



has three states and six transitions. In this example, we use the indices of the source and target to name the transitions. For example, the transition from s_0 *to* s_2 *is named* t_{02}*. Given this naming convention, the functions* $\text{source}_{\mathscr{S}}$ *and* $\text{target}_{\mathscr{S}}$ *are defined in the obvious way. For example,* $\text{source}_{\mathscr{S}}(t_{02}) = s_0$ *and* $\text{target}_{\mathscr{S}}(t_{02}) = s_2$*. The function* $\text{prob}_{\mathscr{S}}$ *can be easily extracted from the above diagram. For example,* $\text{prob}_{\mathscr{S}}(t_{02}) = \frac{1}{2}$*. All states are labelled with the atomic proposition a and the states* s_1 *and* s_2 *are also labelled with the atomic proposition b. Hence, for example,* $\text{label}_{\mathscr{S}}(s_2) = \{a, b\}$.

Instead of $\langle S, T, AP, s_0, \text{source}, \text{target}, \text{prob}, \text{label}\rangle$ we usually write $\mathscr{S}$ and we denote, for example, its set of states by $S_{\mathscr{S}}$. We model the potential executions of the system under verification as execution paths of the PTS.

Definition 3 *An execution path of a PTS $\mathscr{S}$ is an infinite sequence of transitions $t_1 t_2 \ldots$ such that*

- *for all $i \geq 1$, $t_i \in T_{\mathscr{S}}$,*
- $\text{source}_{\mathscr{S}}(t_1) = s_{0,\mathscr{S}}$, *and*
- *for all $i \geq 1$,* $\text{target}_{\mathscr{S}}(t_i) = \text{source}_{\mathscr{S}}(t_{i+1})$.

The set of all execution paths is denoted by $\text{Exec}_{\mathscr{S}}$.

Example 4 *Consider the PTS of Example 2. For this system, $t_{02}{t_{22}}^{\omega}$, $t_{01}t_{13}{t_{33}}^{\omega}$, and $t_{01}t_{10}t_{02}{t_{22}}^{\omega}$ are examples of execution paths.*

To define the progress measure, we use a measurable space of execution paths. We assume that the reader is familiar with the basics of measure theory as can be found in, for example, [3]. Recall that a measurable space consists of a set, a σ-algebra and a measure. In our case, the set is $\text{Exec}_{\mathscr{S}}$. The σ-algebra $\Sigma_{\mathscr{S}}$ is generated from the basic cylinder sets defined below. We denote the set of finite prefixes of execution paths in $\text{Exec}_{\mathscr{S}}$ by $\text{pref}(\text{Exec}_{\mathscr{S}})$.

Definition 5 *Let $e \in \text{pref}(\text{Exec}_{\mathscr{S}})$. Its basic cylinder set $B^e_{\mathscr{S}}$ is defined by*

$$B^e_{\mathscr{S}} = \{\, e' \in \text{Exec}_{\mathscr{S}} \mid e \text{ is a prefix of } e' \,\}.$$

The measure $\mu_{\mathscr{S}}$ is defined on a basic cylinder set $B^{t_1 \ldots t_n}_{\mathscr{S}}$ by

$$\mu_{\mathscr{S}}(B^{t_1 \ldots t_n}_{\mathscr{S}}) = \prod_{1 \leq i \leq n} \text{prob}_{\mathscr{S}}(t_i).$$

The measurable space $\langle \text{Exec}_{\mathscr{S}}, \Sigma_{\mathscr{S}}, \mu_{\mathscr{S}} \rangle$ is a sequence space as defined, for example, in [5, Chapter 2].

The verification effort of the probabilistic model checker is represented by its search of the PTS. The search is captured by the set of transitions that have been explored during the search.

Definition 6 *A search of a PTS $\mathscr{S}$ is a finite subset of $T_{\mathscr{S}}$.*

Example 7 *Consider the PTS of Example 2. The sets $\emptyset$, $\{t_{01}\}$, $\{t_{02}\}$, $\{t_{01}, t_{02}\}$ and $\{t_{01}, t_{02}, t_{10}, t_{13}, t_{22}, t_{33}\}$ are examples of searches.*

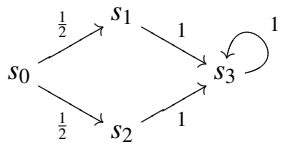
A PTS is said to extend a search if the transitions of the search are part of the PTS. We will use this notion in the definition of the progress measure.

Definition 8 *The PTS $\mathscr{S}'$ extends the search T of the PTS $\mathscr{S}$ if for all $t \in T$,*

- $t \in T_{\mathscr{S}'}$,
- $s_{0,\mathscr{S}} = s_{0,\mathscr{S}'}$,
- $\text{source}_{\mathscr{S}'}(t) = \text{source}_{\mathscr{S}}(t)$,
- $\text{target}_{\mathscr{S}'}(t) = \text{target}_{\mathscr{S}}(t)$,
- $\text{prob}_{\mathscr{S}'}(t) = \text{prob}_{\mathscr{S}}(t)$,
- $\text{label}_{\mathscr{S}'}(\text{source}_{\mathscr{S}'}(t)) = \text{label}_{\mathscr{S}}(\text{source}_{\mathscr{S}}(t))$, *and*

- $\text{label}_{\mathscr{S}'}(\text{target}_{\mathscr{S}'}(t)) = \text{label}_{\mathscr{S}}(\text{target}_{\mathscr{S}}(t))$.

Example 9 *Consider the PTS of Example 2 and the search* $\{t_{01}, t_{02}\}$. *The PTS*



extends the search.

Since the PTSs we will consider in the remainder of this paper all extend a search T of a PTS $\mathscr{S}$, we write s_0 instead of $s_{0,\mathscr{S}}$ to avoid clutter. PTSs that extend a particular search give rise to the same set of execution paths if we restrict ourselves to those execution paths that only consist of transitions explored during the search.

Proposition 10 *If the PTS* $\mathscr{S}'$ *extends the search* T *of the PTS* $\mathscr{S}$, *then*

(a) $T^* \cap \text{pref}(\text{Exec}_{\mathscr{S}}) = T^* \cap \text{pref}(\text{Exec}_{\mathscr{S}'})$ *and*

(b) $T^\omega \cap \text{Exec}_{\mathscr{S}} = T^\omega \cap \text{Exec}_{\mathscr{S}'}$.

PTSs that extend a particular search also assign the same measure to basic cylinder sets of prefixes of execution paths only consisting of transitions explored during the search.

Proposition 11 *If the PTS* $\mathscr{S}'$ *extends the search* T *of the PTS* $\mathscr{S}$, *then* $\mu_{\mathscr{S}}(B^e_{\mathscr{S}}) = \mu_{\mathscr{S}'}(B^e_{\mathscr{S}'})$ *for all* $e \in T^* \cap \text{pref}(\text{Exec}_{\mathscr{S}})$.

The function $\text{label}_{\mathscr{S}}$ assigns to each state the set of atomic propositions that hold in the state. This function is extended to (prefixes of) execution paths as follows.

Definition 12 *The function* $\text{trace}_{\mathscr{S}} : \text{Exec}_{\mathscr{S}} \to (2^{AP_{\mathscr{S}}})^\omega$ *is defined by*

$$\text{trace}_{\mathscr{S}}(t_1 t_2 \ldots) = \text{label}_{\mathscr{S}}(\text{source}_{\mathscr{S}}(t_1))\text{label}_{\mathscr{S}}(\text{source}_{\mathscr{S}}(t_2)) \ldots$$

The function $\text{trace}_{\mathscr{S}} : \text{pref}(\text{Exec}_{\mathscr{S}}) \to (2^{AP_{\mathscr{S}}})^*$ *is defined by*

$$\text{trace}_{\mathscr{S}}(t_1 \ldots t_n) = \text{label}_{\mathscr{S}}(\text{source}_{\mathscr{S}}(t_1)) \ldots \text{label}_{\mathscr{S}}(\text{source}_{\mathscr{S}}(t_n))\text{label}_{\mathscr{S}}(\text{target}_{\mathscr{S}}(t_n))$$

Example 13 *Consider the PTS* $\mathscr{S}$ *of Example 2.*

$$\begin{aligned} \text{trace}_{\mathscr{S}}(t_{02}t_{22}{}^\omega) &= \{a\}\{a,b\}^\omega \\ \text{trace}_{\mathscr{S}}(t_{01}t_{13}t_{33}{}^\omega) &= \{a\}\{a,b\}\{a\}^\omega \\ \text{trace}_{\mathscr{S}}(t_{01}t_{10}t_{02}t_{22}{}^\omega) &= \{a\}\{a,b\}\{a\}\{a,b\}^\omega \end{aligned}$$

For the definition of linear-time property and the satisfaction relation $\models$ we refer the reader to, for example, [2, Section 3.2]. Based on these notions, we define when an execution path of a PTS satisfies a linear-time property.

Definition 14 *The satisfaction relation* $\models_{\mathscr{S}}$ *is defined by*

$$e \models_{\mathscr{S}} \phi \text{ if } \text{trace}_{\mathscr{S}}(e) \models \phi.$$

For PTSs that extend a particular search, those execution paths that only consist of transitions explored by the search satisfy the same linear-time properties.

Proposition 15 *Let ϕ be a linear-time property. If the PTS $\mathscr{S}'$ extends the search T of the PTS $\mathscr{S}$, then $e \models_{\mathscr{S}} \phi$ iff $e \models_{\mathscr{S}'} \phi$ for all $e \in T^{\omega} \cap \mathrm{Exec}_{\mathscr{S}}$.*

Proof Since $\mathscr{S}'$ extends T of $\mathscr{S}$, $\mathrm{trace}_{\mathscr{S}}(e) = \mathrm{trace}_{\mathscr{S}'}(e)$ for all $e \in T^{\omega} \cap \mathrm{Exec}_{\mathscr{S}}$. □

Next, we introduce the notion of a progress measure. Given a search of a PTS and a linear-time property, it captures the amount of progress the search of the probabilistic model checker has made towards verifying the linear-time property.

Definition 16 *Let the PTS $\mathscr{S}'$ extend the search T of PTS $\mathscr{S}$ and let ϕ be a linear-time property. The set $\mathscr{B}^{\phi}_{\mathscr{S}'}(T)$ is defined by*

$$\mathscr{B}^{\phi}_{\mathscr{S}'}(T) = \bigcup \{\, B^{e}_{\mathscr{S}'} \mid e \in T^{*} \wedge \forall e' \in B^{e}_{\mathscr{S}'} : e' \models_{\mathscr{S}'} \phi \,\}.$$

The set $\mathscr{B}^{\phi}_{\mathscr{S}'}(T)$ is the union of those basic cylinder sets $B^{e}_{\mathscr{S}'}$ the execution paths of which satisfy the linear-time property ϕ. Hence, $B^{e}_{\mathscr{S}'}$ does not contain any execution paths violating ϕ. The set $\mathscr{B}^{\phi}_{\mathscr{S}'}(T)$ is measurable, as shown in [11, Proposition 1]. Hence, the measure $\mu_{\mathscr{S}'}$ assigns it a real number in the unit interval. This number represents the "size" of the basic cylinder sets that do not contain any violations of ϕ. This number captures the amount of progress of the search T verifying ϕ, *provided that* the PTS under consideration is $\mathscr{S}'$. However, we have no knowledge of the transitions other than the search. Therefore, we consider all extensions $\mathscr{S}'$ of T and consider the worst case in terms of progress.

Definition 17 *The progress of the search T of the PTS $\mathscr{S}$ of the linear-time property ϕ is defined by*

$$\mathrm{prog}_{\mathscr{S}}(T,\phi) = \inf \left\{\, \mu_{\mathscr{S}'} \left(\mathscr{B}^{\phi}_{\mathscr{S}'}(T) \right) \mid \mathscr{S}' \textit{ extends } T \textit{ of } \mathscr{S} \,\right\}.$$

Example 18 *Consider the PTS $\mathscr{S}$ of Example 2 and the linear temporal logic formulae $\Box a$, $\Diamond a$, $\Diamond b$ and $\bigcirc b$. In the table below, we present the progress of these properties for a number of searches.*

search	$\Box a$	$\Diamond a$	$\Diamond b$	$\bigcirc b$
$\emptyset$	0	1	0	0
$\{t_{01}\}$	0	1	$\frac{1}{2}$	$\frac{1}{2}$
$\{t_{02}\}$	0	1	$\frac{1}{2}$	$\frac{1}{2}$
$\{t_{01}, t_{02}\}$	0	1	1	1
$\{t_{01}, t_{13}, t_{33}\}$	$\frac{1}{4}$	1	$\frac{1}{2}$	$\frac{1}{2}$
$\{t_{01}, t_{10}, t_{13}, t_{33}\}$	$\frac{1}{3}$	1	$\frac{1}{2}$	$\frac{1}{2}$

In [11, Theorem 1], Zhang and Van Breugel prove the following key property of their progress measure. They show that it is a lower bound for the probability that the linear-time property holds.

Theorem 19 *Let T be a search of the PTS $\mathscr{S}$ and let ϕ be a linear-time property. Then*

$$\mathrm{prog}_{\mathscr{S}}(T,\phi) \leq \mu_{\mathscr{S}}(\{\, e \in \mathrm{Exec}_{\mathscr{S}} \mid e \models_{\mathscr{S}} \phi \,\}).$$

The setting in this paper is slightly different from the one in [11]. In this paper we assume that PTSs do not have final states. This assumption can be made without loss of any generality: simply add a self loop with probability one to each final state.

3 Negation and Violations

In this section, we consider the relationship between making progress towards verifying a linear-time property and finding a violation of its negation. First, we formalize that a search has not found a violation of a linear-time property.

Definition 20 *The search T of the PTS $\mathscr{S}$ has not found a violation of the linear-time property ϕ if there exists a PTS $\mathscr{S}'$ which extends T of $\mathscr{S}$ such that $e \models_{\mathscr{S}'} \phi$ for all $e \in \text{Exec}_{\mathscr{S}'}$.*

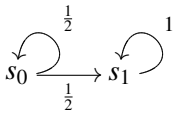
This definition is slightly stronger than the one given in [11, Definition 7]. All results of [11] remain valid for this stronger version. Next, we prove that if a search has made some progress towards verifying a linear-time property $\neg\phi$, then that search has also found a violation of ϕ.

Proposition 21 *Let T be a search of the PTS $\mathscr{S}$ and let ϕ be a linear-time property. If* $\text{prog}_{\mathscr{S}}(T, \neg\phi) > 0$ *then T has found a violation of ϕ.*

Proof By the definition of prog, $\mu_{\mathscr{S}'}(\mathscr{B}^{\neg\phi}_{\mathscr{S}'}(T)) > 0$ for each PTS $\mathscr{S}'$ which extends T of $\mathscr{S}$. Hence, $\mathscr{B}^{\neg\phi}_{\mathscr{S}'}(T) \neq \emptyset$. Therefore, there exists $e \in T^*$ such that $B^e_{\mathscr{S}'} \neq \emptyset$ and $\forall e' \in B^e_{\mathscr{S}'} : e' \models_{\mathscr{S}'} \neg\phi$. Hence, $e' \not\models \phi$ and $e' \in \text{Exec}_{\mathscr{S}'}$. Therefore, T has found a violation of ϕ. □

The reverse implication does not hold in general, as shown in the following example.

Example 22 *Consider the PTS*



Assume that the state s_0 satisfies the atomic proposition a and the state s_1 does not. Consider the linear-time property $\Box a$ and the search $\{t_{00}\}$. Note that ${t_{00}}^\omega \not\models \neg\Box a$ and, hence, $\{t_{00}\}$ has found a violation of $\neg\Box a$. Also note that $\text{prog}_{\mathscr{S}}(\{t_{00}\}, \Box a) = 0$.

We conjecture that the reverse implication does hold for safety properties (see, for example, [2, Definition 3.22] for a formal definition of safety property). However, so far we have only been able to prove it for invariants.

Proposition 23 *If the search T of the PTS $\mathscr{S}$ has found a violation of the invariant ϕ then* $\text{prog}_{\mathscr{S}}(T, \neg\phi) > 0$.

Proof For every PTS $\mathscr{S}'$ that extends T, $e \not\models_{\mathscr{S}'} \Box a$ for some $e \in \text{Exec}_{\mathscr{S}'}$. Hence, $e = e_f t e_\ell$ for some $e_f \in T^* \cap \text{pref}(\text{Exec}_{\mathscr{S}'})$ and $t \in T$ such that $a \notin \text{label}_{\mathscr{S}'}(\text{source}_{\mathscr{S}'}(t))$. Therefore, for all $e' \in B^{e_f}_{\mathscr{S}'}$ we have that $e' \models_{\mathscr{S}'} \neg\Box a$ and $B^{e_f}_{\mathscr{S}'} \neq \emptyset$. Hence, $\mu_{\mathscr{S}'}(B^{e_f}_{\mathscr{S}'}) > 0$ and, therefore, $\text{prog}_{\mathscr{S}}(T, \neg\Box a) > 0$. □

4 A Positive Fragment of LTL

Next, we introduce a positive fragment of linear temporal logic (LTL). This fragment lacks negation. In Section 5 we will show how to compute the progress measure for this fragment.

Definition 24 *The logic LTL_+ is defined by*

$$\phi ::= \text{true} \mid \text{false} \mid a \mid \phi \wedge \phi \mid \phi \vee \phi \mid \bigcirc\phi \mid \phi_1 \,\mathscr{U}\, \phi_2 \mid \phi_1 \,\mathscr{R}\, \phi_2$$

where $a \in AP$.

The grammar defining LTL_+ is the same as the grammar defining the logic PNF introduced in [2, Definition 5.23], except that the grammar of LTL_+ does not contain $\neg a$. For each LTL formula, there exists an equivalent PNF formula (see, for example, [2, Section 5.1.5]). Such a result, of course, does not hold for LTL_+.

A property of LTL_+ that is key for our development is presented next.

Proposition 25 *For all LTL_+ formulae ϕ and $\sigma \in (2^{AP})^*$, $\sigma\emptyset^\omega \models \phi$ iff $\forall \rho \in (2^{AP})^\omega : \sigma\rho \models \phi$.*

Proof We prove two implications. Let ϕ be a LTL_+ formula and let $\sigma \in (2^{AP})^*$. Assume that $\forall \rho \in (2^{AP})^\omega : \sigma\rho \models \phi$. Since $\emptyset^\omega \in (2^{AP})^\omega$, we can immediately conclude that $\sigma\emptyset^\omega \models \phi$.
The other implication is proved by structural induction on ϕ. Let $\sigma \in (2^{AP})^*$. We distinguish the following cases.

- In case $\phi = \text{true}$, clearly $\forall \rho \in (2^{AP})^\omega : \sigma\rho \models \phi$ and, hence, the property is satisfied.
- In case $\phi = \text{false}$, obviously $\sigma\emptyset^\omega \models \phi$ is not satisfied and, therefore, the property holds.
- Let $\phi = a$. If $\sigma\emptyset^\omega \models \phi$, then $|\sigma| > 0$ and $a \in \sigma[0]$ and, hence, $\forall \rho \in (2^{AP})^\omega : \sigma\rho \models \phi$.
- Let $\phi = \phi_1 \wedge \phi_2$. Assume that $\sigma\emptyset^\omega \models \phi$. Then $\sigma\emptyset^\omega \models \phi_1$ and $\sigma\emptyset^\omega \models \phi_2$. By induction, $\forall \rho \in (2^{AP})^\omega : \sigma\rho \models \phi_1$ and $\forall \rho \in (2^{AP})^\omega : \sigma\rho \models \phi_2$. Hence, $\forall \rho \in (2^{AP})^\omega : \sigma\rho \models \phi$.
- The case $\phi = \phi_1 \vee \phi_2$ is similar to the previous case.
- For $\bigcirc\phi$ we distinguish the following two cases. Assume $|\sigma| = 0$. Suppose $\sigma\emptyset^\omega \models \bigcirc\phi$. Then $\emptyset^\omega[1\ldots] = \emptyset^\omega \models \phi$. By induction, $\forall \rho \in (2^{AP})^\omega : \rho \models \phi$. Hence, $\forall \rho \in (2^{AP})^\omega : \rho \models \bigcirc\phi$.
  Assume $|\sigma| \geq 1$. Suppose $\sigma\emptyset^\omega \models \bigcirc\phi$. Then $(\sigma\emptyset^\omega)[1\ldots] = \sigma[1\ldots]\emptyset^\omega \models \phi$. By induction, $\forall \rho \in (2^{AP})^\omega : \sigma[1\ldots]\rho \models \phi$. Since $\sigma[1\ldots]\rho = (\sigma\rho)[1\ldots]$, we have that $\forall \rho \in (2^{AP})^\omega : \sigma\rho \models \bigcirc\phi$.
- Next, let $\phi = \phi_1 \mathscr{U} \phi_2$. Assume that $\sigma\emptyset^\omega \models \phi$. Then there exists some $j \geq 0$ such that
  (a) $(\sigma\emptyset^\omega)[i\ldots] \models \phi_1$ for all $0 \leq i < j$ and
  (b) $(\sigma\emptyset^\omega)[j\ldots] \models \phi_2$.
  We distinguish two cases. Suppose $j < |\sigma|$. From (a) we can conclude that for all $0 \leq i < j$, $(\sigma\emptyset^\omega)[i\ldots] = \sigma[i\ldots]\emptyset^\omega \models \phi_1$. By induction, $\forall \rho \in (2^{AP})^\omega : \sigma[i\ldots]\rho \models \phi_1$. Since $\sigma[i\ldots]\rho = (\sigma\rho)[i\ldots]$, we have that $\forall \rho \in (2^{AP})^\omega : (\sigma\rho)[i\ldots] \models \phi_1$. From (b) we can deduce that $(\sigma\emptyset^\omega)[j\ldots] = \sigma[j\ldots]\emptyset^\omega \models \phi_2$. By induction, $\forall \rho \in (2^{AP})^\omega : \sigma[j\ldots]\rho \models \phi_2$. Since $\sigma[j\ldots]\rho = (\sigma\rho)[j\ldots]$, we have that $\forall \rho \in (2^{AP})^\omega : (\sigma\rho)[j\ldots] \models \phi_2$. Combining the above, we get $\forall \rho \in (2^{AP})^\omega : \sigma\rho \models \phi_1 \mathscr{U} \phi_2$.
  Suppose $j \geq |\sigma|$. For $0 \leq i < |\sigma|$, the argument for (a) is the same as above. For $|\sigma| \leq i < j$, (a) simply says that $\emptyset^\omega \models \phi_1$, which, by induction, implies that $\forall \rho \in (2^{AP})^\omega : \rho \models \phi_1$. Hence, $\forall \rho \in (2^{AP})^\omega : (\sigma\rho)[i\ldots] \models \phi_1$ for all $0 \leq i < j$. In this case, (b) means $\emptyset^\omega \models \phi_2$, which, by induction, implies that $\forall \rho \in (2^{AP})^\omega : \rho \models \phi_2$. Hence, $\forall \rho \in (2^{AP})^\omega : (\sigma\rho)[j\ldots] \models \phi_2$. Combining the above, we obtain that $\forall \rho \in (2^{AP})^\omega : \sigma\rho \models \phi_1 \mathscr{U} \phi_2$.
- Finally, we consider $\phi_1 \mathscr{R} \phi_2$. According to [2, page 256], $\phi_1 \mathscr{R} \phi_2 \equiv \neg(\neg\phi_1 \mathscr{U} \neg\phi_2)$ and $\neg(\phi_1 \mathscr{U} \phi_2) \equiv (\neg\phi_2) \mathscr{W} (\neg\phi_1 \wedge \neg\phi_2)$. According to [2, page 252], $\phi_1 \mathscr{W} \phi_2 \equiv (\phi_1 \mathscr{U} \phi_2) \vee \Box\phi_1$. Hence, we can derive that $\phi_1 \mathscr{R} \phi_2 \equiv (\phi_2 \mathscr{U} (\phi_1 \wedge \phi_2)) \vee \Box\phi_2$. Therefore, proving that the property is satisfied by $\Box\phi$, combined with the proofs for $\wedge$, $\vee$ and $\mathscr{U}$ above, suffices as proof for $\phi_1 \mathscr{R} \phi_2$.
  Thus, we consider $\Box\phi$. Suppose that $\sigma\emptyset^\omega \models \Box\phi$. Then $(\sigma\emptyset^\omega)[j\ldots] \models \phi$ for all $j \geq 0$. We distinguish two cases. For all $0 \leq j < |\sigma|$, we have that $(\sigma\emptyset^\omega)[j\ldots] = \sigma[j\ldots]\emptyset^\omega \models \phi$. By induction, $\forall \rho \in (2^{AP})^\omega : \sigma[j\ldots]\rho \models \phi$ and, hence, $\forall \rho \in (2^{AP})^\omega : (\sigma\rho)[j\ldots] \models \phi$.
  For all $j \geq |\sigma|$, we have that $(\sigma\emptyset^\omega)[j\ldots] = \emptyset^\omega \models \phi$. By induction, $\forall \rho \in (2^{AP})^\omega : \rho \models \phi$ and, therefore, $\forall \rho \in (2^{AP})^\omega : (\sigma\rho)[j\ldots] \models \phi$. Combining the above, we get $\forall \rho \in (2^{AP})^\omega : \sigma\rho \models \Box\phi$.

□

The above result does not hold for all LTL formulae, as shown in the following example.

Example 26 *Consider the LTL formula $\neg a$. Note that this formula is not equivalent to any LTL$_+$ formula. Let $\sigma = \varepsilon$. Obviously, $\emptyset^\omega \models \neg a$, but it is not the case that $\forall \rho \in (2^{AP})^\omega : \rho \models \neg a$ (just take a $\rho \in (2^{AP})^\omega$ with $a \in \rho[0]$).*

5 An Algorithm to Compute Progress

To obtain an algorithm to compute the progress for the positive fragment of LTL, we present an alternative characterization of the progress measure. This alternative characterization is cast in terms of a PTS built from the search as follows. We start from the transitions of the search and their source and target states. We add a sink state, which has a transition to itself with probability one and which does not satisfy any atomic proposition. For each state which has not been fully explored yet, that is, the sum of the probabilities of its outgoing transitions is less than one, we add a transition to the sink state with the remaining probability. This PTS can be viewed as the minimal extension of the search (we will formalize this in Proposition 34). The PTS is defined as follows.

Definition 27 *Let T be a search of the PTS $\mathscr{S}$. The set $S^T_{\mathscr{S}}$ is defined by*

$$S^T_{\mathscr{S}} = \{\, \text{source}_{\mathscr{S}}(t) \mid t \in T \,\} \cup \{\, \text{target}_{\mathscr{S}}(t) \mid t \in T \,\} \cup \{s_0\}.$$

For each $s \in S^T_{\mathscr{S}}$,

$$\text{out}_{\mathscr{S}}(s) = \sum \{\, \text{prob}_{\mathscr{S}}(t) \mid t \in T \wedge \text{source}_{\mathscr{S}}(t) = s \,\}.$$

The PTS $\mathscr{S}_T$ is defined by

- $S_{\mathscr{S}_T} = S^T_{\mathscr{S}} \cup \{s_\perp\}$,
- $T_{\mathscr{S}_T} = T \cup \{\, t_s \mid s \in S^T_{\mathscr{S}} \wedge \text{out}_{\mathscr{S}}(s) < 1 \,\} \cup \{t_\perp\}$,
- $\text{source}_{\mathscr{S}_T}(t) = \begin{cases} \text{source}_{\mathscr{S}}(t) & \text{if } t \in T \\ s & \text{if } t = t_s \\ s_\perp & \text{if } t = t_\perp \end{cases}$
- $\text{target}_{\mathscr{S}_T}(t) = \begin{cases} \text{target}_{\mathscr{S}}(t) & \text{if } t \in T \\ s_\perp & \text{if } t = t_\perp \text{ or } t = t_s \end{cases}$
- $\text{prob}_{\mathscr{S}_T}(t) = \begin{cases} \text{prob}_{\mathscr{S}}(t) & \text{if } t \in T \\ 1 - \text{out}_{\mathscr{S}}(s) & \text{if } t = t_s \\ 1 & \text{if } t = t_\perp \end{cases}$
- $\text{label}_{\mathscr{S}_T}(s) = \begin{cases} \emptyset & \text{if } s = s_\perp \\ \text{label}_{\mathscr{S}}(s) & \text{otherwise} \end{cases}$

The above definition is very similar to [11, Definition 10]. The main difference is that we do not have final states.

Proposition 28 *Let T be a search of the PTS $\mathscr{S}$. Then the PTS $\mathscr{S}_T$ extends T.*

Proof Follows immediately from the definition of $\mathscr{S}_T$. □

Next, we will show that the PTS $\mathscr{S}_T$ is the minimal extension of the search T of the PTS $\mathscr{S}$. More precisely, we will prove that for any other extension $\mathscr{S}'$ of T we have that $\mu_{\mathscr{S}_T}(\mathscr{B}^\phi_{\mathscr{S}_T}) \leq \mu_{\mathscr{S}'}(\mathscr{B}^\phi_{\mathscr{S}'})$. To prove this result, we introduce two new notions and some of their properties.

Definition 29 *Let T be a search of the PTS $\mathscr{S}$ and let ϕ be a linear-time property. The set $E^{\phi}_{\mathscr{S}}(T)$ is defined by*

$$E^{\phi}_{\mathscr{S}}(T) = \{\, e \in T^* \cap \mathrm{pref}(\mathrm{Exec}_{\mathscr{S}}) \mid \forall e' \in B^{e}_{\mathscr{S}} : e' \models_{\mathscr{S}} \phi \,\}.$$

The set $E^{\phi}_{\mathscr{S}_T}(T)$ is minimal among the $E^{\phi}_{\mathscr{S}'}(T)$ where $\mathscr{S}'$ extends T.

Proposition 30 *Let the PTS $\mathscr{S}'$ extend the search T of the PTS $\mathscr{S}$. For any LTL$_+$ formula ϕ, $E^{\phi}_{\mathscr{S}_T}(T) \subseteq E^{\phi}_{\mathscr{S}'}(T)$.*

Next, we restrict our attention to those elements of $E^{\phi}_{\mathscr{S}}(T)$ which are minimal with respect to the prefix order.

Definition 31 *Let T be a search of the PTS $\mathscr{S}$ and let ϕ be a linear-time property. The set $ME^{\phi}_{\mathscr{S}}(T)$ is defined by*

$$ME^{\phi}_{\mathscr{S}}(T) = \{\, e \in E^{\phi}_{\mathscr{S}}(T) \mid |e| > 0 \Rightarrow \exists e' \in B^{e[|e|-1]}_{\mathscr{S}} : e' \not\models_{\mathscr{S}} \phi \,\}.$$

Note that $e \in ME^{\phi}_{\mathscr{S}}(T)$ if and only if it belongs to $E^{\phi}_{\mathscr{S}}(T)$ and none of its prefixes belong to $E^{\phi}_{\mathscr{S}}(T)$.

Proposition 32 *Let the PTSs $\mathscr{S}'$ and $\mathscr{S}''$ extend the search T of the PTS $\mathscr{S}$ and let ϕ be a linear-time property. Then*

$$\bigcup_{\bar{e} \in ME^{\phi}_{\mathscr{S}''}(T)} B^{\bar{e}}_{\mathscr{S}'} = \bigcup_{e \in E^{\phi}_{\mathscr{S}''}(T)} B^{e}_{\mathscr{S}'}.$$

Proof Since $ME^{\phi}_{\mathscr{S}''}(T) \subseteq E^{\phi}_{\mathscr{S}''}(T)$, we can conclude that the set on the left hand side is a subset of the set on the right hand side. Next, we prove the other inclusion. We show that for each $e \in E^{\phi}_{\mathscr{S}''}(T)$ there exists $\bar{e} \in ME^{\phi}_{\mathscr{S}''}(T)$ such that $B^{e}_{\mathscr{S}'} \subseteq B^{\bar{e}}_{\mathscr{S}'}$ by induction on the length of e. In the base case, $|e| = 0$, then $e \in E^{\phi}_{\mathscr{S}''}(T)$ implies $e \in ME^{\phi}_{\mathscr{S}''}(T)$ and, hence, we take $\bar{e}$ to be e. Let $|e| > 0$. We distinguish two cases. If $\exists e' \in B^{e[|e|-1]}_{\mathscr{S}'} : e' \not\models_{\mathscr{S}} \phi$ then we also take $\bar{e}$ to be e. Otherwise, $e[|e|-1] \in E^{\phi}_{\mathscr{S}''}(T)$. Obviously, $B^{e}_{\mathscr{S}'} \subseteq B^{e[|e|-1]}_{\mathscr{S}'}$ and, by induction, there exists a $\bar{e} \in ME^{\phi}_{\mathscr{S}''}(T)$ such that $B^{e[|e|-1]}_{\mathscr{S}'} \subseteq B^{\bar{e}}_{\mathscr{S}'}$. □

Proposition 33 *Let the PTSs $\mathscr{S}'$ and $\mathscr{S}''$ extend the search T of the PTS $\mathscr{S}$, and let ϕ be a linear-time property. If $E^{\phi}_{\mathscr{S}'}(T) \subseteq E^{\phi}_{\mathscr{S}''}(T)$ then*

$$\mu_{\mathscr{S}''}\Big(\bigcup\{\, B^{e}_{\mathscr{S}''} \mid e \in ME^{\phi}_{\mathscr{S}'}(T) \,\}\Big) = \sum_{e \in ME^{\phi}_{\mathscr{S}'}(T)} \mu_{\mathscr{S}''}(B^{e}_{\mathscr{S}''}). \tag{1}$$

Proof We have that

$$\begin{array}{lcll} ME^{\phi}_{\mathscr{S}'}(T) & \subseteq & E^{\phi}_{\mathscr{S}'}(T) & \text{[by definition]} \\ & \subseteq & E^{\phi}_{\mathscr{S}''}(T) & \text{[by assumption]} \\ & \subseteq & \mathrm{pref}(\mathrm{Exec}_{\mathscr{S}''}) & \text{[by definition]} \end{array}$$

Hence, for all $e \in ME^{\phi}_{\mathscr{S}'}(T)$, we have that $B^{e}_{\mathscr{S}''} \in \Sigma_{\mathscr{S}''}$. Since the set T is finite, the set T^* is countable and, hence, the set $ME^{\phi}_{\mathscr{S}'}(T)$ is countable as well. Since a σ-algebra is closed under countable unions, $\bigcup\{\, B^{e}_{\mathscr{S}''} \mid e \in ME^{\phi}_{\mathscr{S}'}(T) \,\} \in \Sigma_{\mathscr{S}''}$. Hence, the measure $\mu_{\mathscr{S}''}$ is defined on this set.

To conclude (1), it suffices to prove that for all $e_1, e_2 \in ME^{\phi}_{\mathscr{S}'}(T)$ such that $e_1 \neq e_2$, e_1 is not a prefix of e_2, since this implies that $B^{e_1}_{\mathscr{S}'}$ and $B^{e_2}_{\mathscr{S}'}$ are disjoint. Towards a contradiction, assume that e_1 is a prefix of e_2. Since $\forall e'_1 \in B^{e_1}_{\mathscr{S}'} : e'_1 \models_{\mathscr{S}'} \phi$ and e_1 is a prefix of e_2 and $e_1 \neq e_2$, it cannot be the case that $\exists e'_2 \in B^{e_2[|e_2|-1]}_{\mathscr{S}'} : e'_2 \not\models_{\mathscr{S}'} \phi$. This contradicts the assumption that $e_2 \in ME^{\phi}_{\mathscr{S}'}(T)$. □

Now, we are ready to prove that the PTS $\mathscr{S}_T$ is the minimal extension of the search T of the PTS $\mathscr{S}$.

Proposition 34 *Let the PTS $\mathscr{S}'$ extend the search T of the PTS $\mathscr{S}$ and let ϕ be a LTL$_+$ formula. Then*

$$\mu_{\mathscr{S}_T}(\mathscr{B}^{\phi}_{\mathscr{S}_T}(T)) \leq \mu_{\mathscr{S}'}(\mathscr{B}^{\phi}_{\mathscr{S}'}(T)).$$

Proof

$$\begin{aligned}
&\mu_{\mathscr{S}_T}(\mathscr{B}^{\phi}_{\mathscr{S}_T}(T)) \\
&= \mu_{\mathscr{S}_T}(\bigcup\{ B^e_{\mathscr{S}_T} \mid e \in E^{\phi}_{\mathscr{S}_T}(T)\}) \\
&= \mu_{\mathscr{S}_T}(\bigcup\{ B^e_{\mathscr{S}_T} \mid e \in ME^{\phi}_{\mathscr{S}_T}(T)\}) \quad \text{[Proposition 32]} \\
&= \sum_{e \in ME^{\phi}_{\mathscr{S}_T}(T)} \mu_{\mathscr{S}_T}(B^e_{\mathscr{S}_T}) \quad \text{[Proposition 33]} \\
&= \sum_{e \in ME^{\phi}_{\mathscr{S}_T}(T)} \mu_{\mathscr{S}'}(B^e_{\mathscr{S}'}) \quad \text{[Proposition 11]} \\
&= \mu_{\mathscr{S}'}(\bigcup\{ B^e_{\mathscr{S}'} \mid e \in ME^{\phi}_{\mathscr{S}_T}(T)\}) \quad \text{[Proposition 30 and 33]} \\
&= \mu_{\mathscr{S}'}(\bigcup\{ B^e_{\mathscr{S}'} \mid e \in E^{\phi}_{\mathscr{S}_T}(T)\}) \quad \text{[Proposition 32]} \\
&\leq \mu_{\mathscr{S}'}(\bigcup\{ B^e_{\mathscr{S}'} \mid e \in E^{\phi}_{\mathscr{S}'}(T)\}) \quad \text{[Proposition 30]} \\
&= \mu_{\mathscr{S}'}(\mathscr{B}^{\phi}_{\mathscr{S}'}(T))
\end{aligned}$$

□

The above proposition gives us an alternative characterization of the progress measure.

Theorem 35 *Let T be a search of the PTS $\mathscr{S}$ and let ϕ be a LTL$_+$ formula. Then*

$$\text{prog}_{\mathscr{S}}(T, \phi) = \mu_{\mathscr{S}_T}(\mathscr{B}^{\phi}_{\mathscr{S}_T}(T)).$$

Proof This is a direct consequence of the definition of the progress measure and Proposition 34. □

Hence, in order to compute $\text{prog}_{\mathscr{S}}(T, \phi)$, it suffices to compute the measure of $\mathscr{B}^{\phi}_{\mathscr{S}_T}(T)$. Next, we will show that the latter is equal to the measure of the set of execution paths of $\mathscr{S}_T$ that satisfy ϕ. The proof consists of two parts. First, we prove the following inclusion.

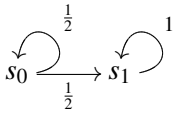
Proposition 36 *Let T be a search of the PTS $\mathscr{S}$ and let ϕ be a linear-time property. Then*

$$\mathscr{B}^{\phi}_{\mathscr{S}_T}(T) \subseteq \{ e \in \text{Exec}_{\mathscr{S}_T} \mid e \models_{\mathscr{S}_T} \phi \}.$$

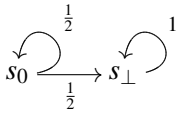
Proof Let $e \in \mathscr{B}^{\phi}_{\mathscr{S}_T}(T)$. Then $e \in B^{e'}_{\mathscr{S}_T}$ for some $e' \in T^*$ such that $\forall e'' \in B^{e'}_{\mathscr{S}_T} : e'' \models_{\mathscr{S}_T} \phi$. Hence, $e \models_{\mathscr{S}_T} \phi$. □

The opposite inclusion does not hold in general, as shown in the following example.

Example 37 *Consider the PTS $\mathscr{S}$*



Consider the search $\{t_{00}\}$. *Then the PTS* $\mathscr{S}_T$ *can be depicted by*



Assume that the state s_0 *satisfies the atomic proposition a. Hence,* ${t_{00}}^{\omega} \models_{\mathscr{S}_T} \Box a$. *By construction, the state* $s_\perp$ *does not satisfy a. Therefore,* ${t_{00}}^{\omega} \notin \mathscr{B}^{\Box a}_{\mathscr{S}_T}$.

However, we will show that the set $\{\, e \in \mathrm{Exec}_{\mathscr{S}_T} \mid e \models_{\mathscr{S}_T} \phi \,\} \setminus \mathscr{B}^{\phi}_{\mathscr{S}_T}(T)$ has measure zero. In the proof, we will use the following proposition.

Proposition 38 *Let* T *be a search of the PTS* $\mathscr{S}$ *and let* ϕ *be a linear-time property. Assume that* T *has not found a violation of* ϕ. *Then for all* $e \in T^{\omega} \cap \mathrm{Exec}_{\mathscr{S}_T}$, $e \models_{\mathscr{S}_T} \phi$.

Proof Let $e \in T^{\omega} \cap \mathrm{Exec}_{\mathscr{S}_T}$. Since T has not found a violation of ϕ, by definition there exists a PTS $\mathscr{S}'$ that extends T of $\mathscr{S}$ such that $e' \models_{\mathscr{S}'} \phi$ for all $e' \in \mathrm{Exec}_{\mathscr{S}'}$. Then $e \in \mathrm{Exec}_{\mathscr{S}'} \cap T^{\omega}$ by Proposition 10(b), because $\mathscr{S}'$ and $\mathscr{S}_T$ both extend T. Hence, $e \models_{\mathscr{S}'} \phi$. Therefore, from Proposition 15 we can conclude that $e \models_{\mathscr{S}_T} \phi$. □

Proposition 39 *Let* T *be a search of the PTS* $\mathscr{S}$ *and let* ϕ *be a LTL*$_+$ *formula. If* T *has not found a violation of* ϕ *then*

$$\mu_{\mathscr{S}_T}(\{\, e \in \mathrm{Exec}_{\mathscr{S}_T} \mid e \models_{\mathscr{S}_T} \phi \,\} \setminus \mathscr{B}^{\phi}_{\mathscr{S}_T}(T)) = 0.$$

Proof To avoid clutter, we denote the set $\{\, e \in \mathrm{Exec}_{\mathscr{S}_T} \mid e \models_{\mathscr{S}_T} \phi \,\} \setminus \mathscr{B}^{\phi}_{\mathscr{S}_T}(T)$ by Z.

First, we show that $Z \subseteq T^{\omega}$. Assume that $e \in Z$. Towards a contradiction, suppose that $e \notin T^{\omega}$. From the construction of $\mathscr{S}_T$ we can deduce that $e = e' t_s {t_\perp}^{\omega}$ for some $e' \in T^*$. Let $\mathrm{trace}_{\mathscr{S}_T}(e') = \sigma$. Then $\mathrm{trace}_{\mathscr{S}_T}(e) = \sigma\emptyset^{\omega}$. Since $e \in Z$, we have that $e \models_{\mathscr{S}_T} \phi$ and, hence, $\sigma\emptyset^{\omega} \models \phi$. By Proposition 25, $\forall \rho \in (2^{AP})^{\omega} : \sigma\rho \models \phi$. Hence, $\forall e'' \in B^{e'}_{\mathscr{S}_T} : e'' \models_{\mathscr{S}_T} \phi$. Since $e \in B^{e'}_{\mathscr{S}_T}$, we have that $e \in \mathscr{B}^{\phi}_{\mathscr{S}_T}(T)$, which contradicts our assumption that $e \in Z$.

Next, we show that each state in $\{\, \mathrm{target}_{\mathscr{S}_T}(e) \mid e \in \mathrm{pref}(Z) \,\}$ is transient. Roughly speaking, a state s is transient if the probability of reaching s in one or more transitions when starting in s is strictly less than one (see, for example, [1, Section 7.3] for a formal definition). It suffices to show that each state in $\{\, \mathrm{target}_{\mathscr{S}_T}(e) \mid e \in \mathrm{pref}(Z) \,\}$ can reach the state $s_\perp$, since in that case the probability of reaching $s_\perp$ and, hence, not returning to the state itself, is greater than zero.

Since T has not found a violation of ϕ, we can conclude from Proposition 38 that $e \models_{\mathscr{S}_T} \phi$ for all $e \in T^{\omega}$. Hence, from the construction of $\mathscr{S}_T$ we can deduce that if $e \not\models_{\mathscr{S}_T} \phi$ then $e \notin T^{\omega}$ and, hence, e reaches $s_\perp$. Let $e \in \mathrm{pref}(Z)$. Hence, there exists $e' \in B^{e}_{\mathscr{S}_T}$ such that $e' \not\models_{\mathscr{S}_T} \phi$. Therefore, e' reaches $s_\perp$ and, hence, $\mathrm{target}_{\mathscr{S}_T}(e)$ can reach $s_\perp$.

Since $Z \subseteq T^{\omega}$, the set $\{\, \mathrm{target}_{\mathscr{S}_T}(e) \mid e \in \mathrm{pref}(Z) \,\}$ is finite. According to [1, page 223], the probability of remaining in a finite set of transient states is zero. As a consequence, the probability of remaining in the set $\{\, \mathrm{target}_{\mathscr{S}'}(e) \mid e \in \mathrm{pref}(Z) \,\}$ is zero. Hence, we can conclude that $\mu_{\mathscr{S}_T}(Z) = 0$. □

From the above, we can derive the following result.

Theorem 40 *Let* T *be a search of the PTS* $\mathscr{S}$ *and let* ϕ *be a LTL*$_+$ *formula. If* T *has not found a violation of* ϕ *then*

$$\mu_{\mathscr{S}_T}(\mathscr{B}^{\phi}_{\mathscr{S}_T}(T)) = \mu_{\mathscr{S}_T}(\{\, e \in \mathrm{Exec}_{\mathscr{S}_T} \mid e \models_{\mathscr{S}_T} \phi \,\}).$$

Proof

$$\begin{aligned}
&\mu_{\mathscr{S}_T}(\mathscr{B}^{\phi}_{\mathscr{S}_T}(T)) \\
&\quad \leq \quad \mu_{\mathscr{S}_T}(\{\, e \in \mathrm{Exec}_{\mathscr{S}_T} \mid e \models_{\mathscr{S}_T} \phi \,\}) \quad \text{[Proposition 36 and } \mu_{\mathscr{S}_T} \text{ is monotone]} \\
&\quad = \quad \mu_{\mathscr{S}_T}(\mathscr{B}^{\phi}_{\mathscr{S}_T}(T)) + \mu_{\mathscr{S}_T}(\{\, e \in \mathrm{Exec}_{\mathscr{S}_T} \mid e \models_{\mathscr{S}_T} \phi \,\} \setminus \mathscr{B}^{\phi}_{\mathscr{S}_T}(T)) \quad \text{[Proposition 36 and } \mu_{\mathscr{S}_T} \text{ is additive]} \\
&\quad = \quad \mu_{\mathscr{S}_T}(\mathscr{B}^{\phi}_{\mathscr{S}_T}(T)) \quad \text{[Proposition 39]}
\end{aligned}$$

□

Combining Theorem 35 and 40, we obtain the following characterization of the progress measure.

Corollary 41 *Let T be a search of the PTS $\mathscr{S}$ and let ϕ be a LTL$_+$ formula. If T has not found a violation of ϕ then*

$$\mathrm{prog}_{\mathscr{S}}(T, \phi) = \mu_{\mathscr{S}_T}(\{\, e \in \mathrm{Exec}_{\mathscr{S}_T} \mid e \models_{\mathscr{S}_T} \phi \,\}).$$

Proof Immediate consequence of Theorem 35 and 40. □

How to compute $\mu_{\mathscr{S}_T}(\{\, e \in \mathrm{Exec}_{\mathscr{S}_T} \mid e \models_{\mathscr{S}_T} \phi \,\})$ can be found, for example, in [4, Section 3.1]. Computing this measure is exponential in the size of ϕ and polynomial in the size of T.

6 An Algorithm to Efficiently Compute a Lower Bound of Progress

The algorithm developed in the previous section to compute $\mathrm{prog}_{\mathscr{S}}(T, \phi)$ is exponential in the size of ϕ. In this section, we trade precision for efficiency. We present an algorithm that does not compute $\mathrm{prog}_{\mathscr{S}}(T, \phi)$, but only provides a lower bound in polynomial time. This lower bound is tight for invariants. However, we also show an example in which the lower bound does not provide us any information.

Next, we show that subsets of $\mathrm{Exec}_{\mathscr{S}}$ can be characterized as countable intersections of countable unions of basic cylinder sets. For $A \subseteq \mathrm{Exec}_{\mathscr{S}}$ and $n \in \mathbb{N}$, we use $A[n]$ to denote the set $\{\, e[n] \mid e \in A \,\}$, where $e[n]$ denotes the execution path e truncated at length n. We prove the characterization by showing two inclusions. The first inclusion holds for arbitrary subsets of $\mathrm{Exec}_{\mathscr{S}}$.

Proposition 42 *For PTS $\mathscr{S}$, let $A \subseteq \mathrm{Exec}_{\mathscr{S}}$. Then*

$$A \subseteq \bigcap_{n \in \mathbb{N}} \bigcup_{e \in A[n]} B^e_{\mathscr{S}}.$$

Proof Let $e' \in A$. It suffices to show that

$$e' \in \bigcup_{e \in A[n]} B^e_{\mathscr{S}} \tag{2}$$

for all $n \in \mathbb{N}$. Let $n \in \mathbb{N}$. To prove (2), it suffices to show that $e' \in B^e_{\mathscr{S}}$ for some $e \in A[n]$. Since $e' \in A$, we have that $e'[n] \in A[n]$. Because $e'[n]$ is a prefix of e' and $e' \in \mathrm{Exec}_{\mathscr{S}}$, we have that $e' \in B^{e'[n]}_{\mathscr{S}}$, which concludes our proof. □

The reverse inclusion does not hold in general. In some of the proofs below we use some metric topology. Those readers unfamiliar with metric topology are referred to, for example, [8]. To prove the reverse inclusion, we use that the set is closed.

Proposition 43 *For PTS $\mathscr{S}$, let $A \subseteq \mathrm{Exec}_{\mathscr{S}}$. If A is closed then*

$$\bigcap_{n\in\mathbb{N}} \bigcup_{e\in A[n]} B^e_{\mathscr{S}} \subseteq A.$$

Proof Let $e' \in \bigcap_{n\in\mathbb{N}} \bigcup_{e\in A[n]} B^e_{\mathscr{S}}$. Then $e' \in \bigcup_{e\in A[n]} B^e_{\mathscr{S}}$ for all $n \in \mathbb{N}$. Hence, for each $n \in \mathbb{N}$ there exists a $e_n \in A[n]$ such that $e' \in B^{e_n}_{\mathscr{S}}$. Thus, for each $n \in \mathbb{N}$ there exists a $e'_n \in A$ such that $e' \in B^{e'_n[n]}_{\mathscr{S}}$ and, hence, $e'_n[n]$ is a prefix of e'.

We distinguish two cases. Assume that for some $n \in \mathbb{N}$, $e'_n[n] = e'_n$. Then e'_n is a prefix of e'. Since also e', $e'_n \in \mathrm{Exec}_{\mathscr{S}}$, we can conclude that $e' = e'_n$. Since $e'_n \in A$ we have that $e' \in A$.

Otherwise, $e'_n[n] \neq e'_n$ for all $n \in \mathbb{N}$. Since also $e'_n[n]$ is a prefix of e', we can conclude that $e'_n[n] = e'[n]$. Let the distance function $d : (\mathrm{pref}(\mathrm{Exec}_{\mathscr{S}}) \cup \mathrm{Exec}_{\mathscr{S}}) \times (\mathrm{pref}(\mathrm{Exec}_{\mathscr{S}}) \cup \mathrm{Exec}_{\mathscr{S}}) \to [0,1]$ be defined by $d(e_1,e_2) = \inf\{\, 2^{-n} \mid e_1[n] = e_2[n] \,\}$. Then, $d(e'_n,e') \leq 2^{-n}$, that is, the sequence $(e'_n)_n$ converges to e'. Because all the elements of the sequence $(e'_n)_n$ are in A and A is closed, we can conclude that the limit e' is in A as well (see, for example, [8, Proposition 3.7.15 and Lemma 7.2.2]). □

PTSs that extend a particular search assign the same measure to closed sets of execution paths consisting only of explored transitions.

Proposition 44 *Let the PTS $\mathscr{S}'$ extend the search T of the PTS $\mathscr{S}$ and let $A \subseteq T^\omega \cap \mathrm{Exec}_{\mathscr{S}}$. If A is closed then $\mu_{\mathscr{S}}(A) = \mu_{\mathscr{S}'}(A)$.*

Proof Obviously, for all $e \in T^*$ and $t \in T$, we have $B^e_{\mathscr{S}} \supseteq B^{et}_{\mathscr{S}}$. As a consequence, $\bigcup_{e\in A[n]} B^e_{\mathscr{S}} \supseteq \bigcup_{e\in A[n+1]} B^e_{\mathscr{S}}$ for all $n \in \mathbb{N}$. Furthermore, $\mu_{\mathscr{S}}(\bigcup_{e\in A[0]} B^e_{\mathscr{S}}) = \mu_{\mathscr{S}}(B^\varepsilon_{\mathscr{S}}) = 1$ and, hence, $\mu_{\mathscr{S}}(\bigcup_{e\in A[0]} B^e_{\mathscr{S}})$ is finite. Since a measure is continuous (see, for example, [3, Theorem 2.1]), we can conclude from the above that

$$\mu_{\mathscr{S}}\left(\bigcap_{n\in\mathbb{N}} \bigcup_{e\in A[n]} B^e_{\mathscr{S}}\right) = \lim_{n\in\mathbb{N}} \mu_{\mathscr{S}}\left(\bigcup_{e\in A[n]} B^e_{\mathscr{S}}\right). \tag{3}$$

Therefore,

$$\begin{aligned}
\mu_{\mathscr{S}}(A) &= \mu_{\mathscr{S}}\left(\bigcap_{n\in\mathbb{N}} \bigcup_{e\in A[n]} B^e_{\mathscr{S}}\right) && \text{[Proposition 42 and 43]}\\
&= \lim_{n\in\mathbb{N}} \mu_{\mathscr{S}}\left(\bigcup_{e\in A[n]} B^e_{\mathscr{S}}\right) && \text{[(3)]}\\
&= \lim_{n\in\mathbb{N}} \sum_{e\in A[n]} \mu_{\mathscr{S}}(B^e_{\mathscr{S}}) && \text{[a measure is countably additive]}\\
&= \lim_{n\in\mathbb{N}} \sum_{t_1\ldots t_n\in A[n]} \prod_{1\leq i\leq n} \mathrm{prob}_{\mathscr{S}}(t_i)\\
&= \lim_{n\in\mathbb{N}} \sum_{t_1\ldots t_n\in A[n]} \prod_{1\leq i\leq n} \mathrm{prob}_{\mathscr{S}'}(t_i) && [\mathscr{S}' \text{ extends } T \text{ of } \mathscr{S}]\\
&= \mu_{\mathscr{S}'}(A) && \text{[by symmetric argument].}
\end{aligned}$$

□

Hence, the PTSs $\mathscr{S}$ and $\mathscr{S}_T$ assign the same measure to the closed set of those execution paths consisting only of explored transitions.

Corollary 45 *Let T be a search of the PTS $\mathscr{S}$. Then $\mu_{\mathscr{S}}(T^\omega \cap \text{Exec}_{\mathscr{S}}) = \mu_{\mathscr{S}_T}(T^\omega \cap \text{Exec}_{\mathscr{S}_T})$.*

Proof Since the sets $\text{Exec}_{\mathscr{S}}$ and T^ω are closed, their intersection is also closed (see, for example, [8, Proposition 3.7.5]) and, hence, the result follows immediately from Proposition 44 and 10(b). □

Now we can show that the measure of the set of execution paths consisting only of explored transitions is a lower bound for the progress measure.

Theorem 46 *Let T be a search of the PTS $\mathscr{S}$ and let ϕ be a LTL$_+$ formula. If T has not found a violation of ϕ then*

$$\mu_{\mathscr{S}_T}(T^\omega \cap \text{Exec}_{\mathscr{S}_T}) \leq \text{prog}_{\mathscr{S}}(T, \phi).$$

Proof

$$\begin{aligned}
&\mu_{\mathscr{S}_T}(T^\omega \cap \text{Exec}_{\mathscr{S}_T}) \\
&\quad \leq \quad \mu_{\mathscr{S}_T}(\{\, e \in \text{Exec}_{\mathscr{S}_T} \mid e \models_{\mathscr{S}_T} \phi \,\}) \quad \text{[Proposition 38]} \\
&\quad = \quad \text{prog}_{\mathscr{S}}(T, \phi) \quad \text{[Corollary 41]}
\end{aligned}$$

□

From the construction of $\mathscr{S}_T$ we can conclude that $\mu_{\mathscr{S}_T}(T^\omega \cap \text{Exec}_{\mathscr{S}_T})$ is the same as $\mu_{\mathscr{S}_T}(\{\, e \in \text{Exec}_{\mathscr{S}_T} \mid e \text{ does not reach } s_\perp \,\})$, which is the same as $1 - \mu_{\mathscr{S}_T}(\{\, e \in \text{Exec}_{\mathscr{S}_T} \mid e \text{ reaches } s_\perp \,\})$. The latter can be computed in polynomial time using, for example, Gaussian elimination (see, for example, [2, Section 10.1.1]). This algorithm has been implemented and incorporated into an extension of the model checker JPF [10]. While JPF is model checking sequential Java code which contains probabilistic choices, our extension also keeps track of the underlying PTS. The amount of memory needed to store this PTS is in general only a small fraction of the total amount of memory needed. Once our extension of JPF runs almost out of memory, it can usually free enough memory so that the progress can be computed from the stored PTS.

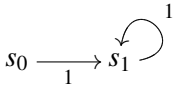
As was shown in [11, Theorem 4], the above bound is tight for invariants.

Proposition 47 *If the search T of the PTS $\mathscr{S}$ has not found a violation of invariant ϕ then*

$$\mu_{\mathscr{S}_T}(T^\omega \cap \text{Exec}_{\mathscr{S}_T}) = \text{prog}_{\mathscr{S}}(T, \phi).$$

In the example below, we present a search of a PTS for a LTL$_+$ formula of which the progress is one whereas the bound is zero. In this case, the bound does not provide us any information.

Example 48 *Consider the PTS*



Assume that the state s_1 satisfies the atomic proposition a. Consider the linear-time property $\bigcirc a$ and the search $\{t_{01}\}$. In this case, we have that $\text{prog}_{\mathscr{S}}(\{t_{01}\}, \bigcirc a) = 1$ but $\mu_{\mathscr{S}_{\{t_{01}\}}}(\{t_{01}\}^\omega \cap \text{Exec}_{\mathscr{S}_{\{t_{01}\}}}) = \mu_{\mathscr{S}_{\{t_{01}\}}}(\emptyset) = 0$.

7 Conclusion

Our work is based on the paper by Zhang and Van Breugel [11]. The work by Pavese, Braberman and Uchitel [6] is also related. They aim to measure the probability that a run of the system reaches a state that has not been visited by the model checker. Also the work by Della Penna et al. [7] seems related. They show how, given a Markov chain and an integer i, the probability of reaching a particular state s within i transitions can be computed.

As we have seen, there seems to be a trade off between efficiency and accuracy when it comes to computing progress. Our algorithm to compute $\text{prog}_{\mathscr{S}}(T,\phi)$ is exponential in the size of the LTL_+ formula ϕ and polynomial in the size of the search T. We even conjecture (and leave it to future work to prove) that the problem of computing progress is PSPACE-hard. However, in general the size of the LTL formula is small, whereas the size of the search is huge. Hence, we expect our algorithm to be useful.

Providing a lower bound for the progress measure can be done in polynomial time. As we have shown, this bound is tight for invariants. Invariants form an important class of properties. Determining the class of LTL_+ formulae for which the bound is tight is another topic for further research.

The approach to handle the positive fragment of LTL seems not applicable to all of LTL. We believe that a different approach is needed and leave this for future research.

Acknowledgments We thank the referees for their constructive feedback.

References

[1] Robert B. Ash (1970): *Basic Probability Theory*. John Wiley & Sons.

[2] Christel Baier & Joost-Pieter Katoen (2008): *Principles of Model Checking*. The MIT Press.

[3] Patrick Billingsley (1995): *Probability and Measure*. John Wiley & Sons.

[4] Costas Courcoubetis & Mihalis Yannakakis (1995): *The Complexity of Probabilistic Verification*. Journal of the ACM 42(4), pp. 857–907.

[5] John G. Kemeny, J. Laurie Snell & Anthony W. Knapp (1966): *Denumerable Markov Chains*. Van Nostrand.

[6] Esteban Pavese, Victor Braberman & Sebastian Uchitel (2010): *My Model Checker Died!: how well did it do?* In: *Proceedings of the 2010 ICSE Workshop on Quantitative Stochastic Models in the Verification and Design of Software Systems*, ACM, Cape Town, pp. 33–40.

[7] Giuseppe Della Penna, Benedetto Intrigila, Igor Melatti, Enrico Tronci & Marisa Venturini Zilli (2006): *Finite Horizon Analysis of Markov Chains with the Murφ Verifier*. International Journal on Software Tools for Technology 8(4/5), pp. 397–409.

[8] Wilson A. Sutherland (1975): *Introduction to Metric and Topological Spaces*. Clarendon Press.

[9] Willem Visser, Klaus Havelund, Guillaume Brat, SeungJoon Park & Flavio Lerda (2003): *Model Checking Programs*. Automated Software Engineering 10(2), pp. 203–232.

[10] Xin Zhang & Franck van Breugel (2010): *Model Checking Randomized Algorithms with Java PathFinder*. In: *Proceedings of 7th International Conference on Quantitative Evaluation of Systems*, IEEE, Williamburgh, pp. 157–158.

[11] Xin Zhang & Franck van Breugel (2011): *A Progress Measure for Explicit-State Probabilistic Model-Checkers*. In: Luca Aceto, Monika Henzinger & Jirí Sgall, editors: *Proceedings of the 38th International Colloquium on Automata, Languages and Programming*, *Lecture Notes in Computer Science* 6756, Springer-Verlag, Zurich, pp. 283–294.

Automated Verification of Quantum Protocols using MCMAS

Francesco Belardinelli Pavel Gonzalez
Alessio Lomuscio
{f.belardinelli, pavel.gonzalez09, a.lomuscio}@imperial.ac.uk
Imperial College London
London, UK

We present a methodology for the automated verification of quantum protocols using MCMAS, a symbolic model checker for multi-agent systems [17]. The method is based on the logical framework developed by D'Hondt and Panangaden [10] for investigating epistemic and temporal properties, built on the model for Distributed Measurement-based Quantum Computation (DMC) [9], an extension of the Measurement Calculus [8] to distributed quantum systems. We describe the translation map from DMC to interpreted systems, the typical formalism for reasoning about time and knowledge in multi-agent systems [14]. Then, we introduce DMC2ISPL, a compiler into the input language of the MCMAS model checker [17]. We demonstrate the technique by verifying the Quantum Teleportation Protocol, and discuss the performance of the tool.

1 Introduction

Quantum computing has gained prominence in the last decade due to theoretical advances as well as applications to security, information processing, and simulation of quantum mechanical systems [19]. With this increase of activity, the need for validation of correctness of quantum algorithms has arisen. Model checking has shown to be a promising verification technique [6]. However, tools and techniques for model checking both temporal and epistemic properties of quantum systems have not been developed yet. In this paper we aim to bridge this gap by introducing a methodology for the automated verification of quantum protocols using MCMAS [17], a symbolic model checker for multi-agent systems (MAS).

The fundamental question from an epistemic point of view is how to model a flow of quantum information. Is it meaningful to talk about "quantum knowledge"? And if it is, how can we express this concept? Several logics, which can be used for reasoning about knowledge in the context of distributed quantum computation, have been recently suggested. One of the first attempts to our knowledge was based on Quantum Message Passing Environments [18]. A different approach, i.e. Quantum Dynamic-Epistemic Logic [1, 2, 3], was developed to model the behaviour of quantum systems. A third account [7, 10, 11] was built on the Distributed Measurement-based Quantum Computation [9], which extends the Measurement Calculus [8], a universal formal model for one-way quantum computations. Among all these accounts, the logic based on Distributed Measurement-based Quantum Computation (DMC) has an underlying operational semantics similar to the semantics of interpreted systems [14]. This feature makes it a well-suited theoretical framework to be used with MCMAS.

In this paper we describe a translation from DMC to interpreted systems (IS). We also report on a source-to-source compiler that performs the translation into the Interpreted Systems Programming Language (ISPL), the modular input language of the MCMAS model checker. The compiler enables the use of MCMAS to verify automatically temporal and epistemic properties of quantum protocols specified in DMC. We verify the Quantum Teleportation protocol [5] against the properties stated and informally proved in [10], and show that one specification does not hold contrary to the paper's claim.

Preliminary Report. Final version to appear in:
QAPL 2012

Related Work. Several approaches to model checking quantum systems have already appeared in the literature. To our knowledge, the only dedicated verification tool for quantum protocols is the Quantum Model Checker (QMC) [15]. The model checker supports specifications in quantum computational temporal logic (QCTL), but quantum operators are restricted to the Clifford group, which is the normalizer of the group of Pauli operators [19]. Although it contains many common operators, quantum circuits that involve only Clifford group operators are not universal. Such circuits can be simulated in polynomial time on a classical computer; however, this leads to a loss of expressive power.

In the same research line [20] a theoretical framework to model check LTL properties using quantum automata is proposed, and an algorithm for checking invariants of quantum systems is presented. Finally, in [13] the Quantum Key Distribution (QKD) protocol is verified against specific eavesdropping security properties. The authors elaborate an *ad hoc* model of the protocol, that they analyse using PRISM [16].

However, we stress that none of these contributions explicitly deal with knowledge. So, these approaches do not allow the verification of the temporal epistemic properties discussed in [10].

Structure. Organizationally, Section 2 gives an overview of the Distributed Measurement-based Quantum Computation, Interpreted Systems, and Quantum Epistemic Logic. Section 3 presents a methodology for translating a protocol specified in DMC into the corresponding IS. Section 4 describes and evaluates an implementation of the formal methodology. Section 5 offers brief conclusions.

2 Preliminaries

We discuss only the issues directly related to the paper and refer the reader to the relevant references for an in-depth coverage of these topics. We assume familiarity with the concepts of quantum computation [19].

2.1 Distributed Measurement-based Quantum Computation

At the heart of the Measurement Calculus are measurement patterns [8]. A *pattern* $\mathscr{P} = (V, I, O, \mathscr{A})$ consists of a *computation space* V, which contains all qubits involved in the execution of $\mathscr{P}$, a set I of input qubits, a set O of output qubits, and a finite sequence $\mathscr{A}$ of commands $A_p \dots A_1$, which are applied to qubits in V from right to left. The possible commands are the entanglement operator E_{qr}, the measurement M_q^{α}, and the corrections X_q and Z_q, where q and r represent the qubits on which these commands operate, and α is a measurement angle in $[0, 2\pi]$.

An *agent* $\mathbf{A}$ [9], denoted as $\mathbf{A}(\mathbf{i}, \mathbf{o}) : Q.\mathscr{E}$, is characterised by its classical input $\mathbf{i}$ and output $\mathbf{o}$, by a set Q of qubits, and by a finite event sequence $\mathscr{E}$, which consists of patterns and commands for classical (c?x, c!y) and quantum (qc?x, qc!q) communication. A *network* $\mathscr{N}$ of agents [9] is defined as a set of concurrently acting agents, together with the global quantum state σ, specifically $\mathscr{N} = \mathbf{A}_1(\mathbf{i}_1, \mathbf{o}_1) : Q_1.\mathscr{E}_1 \mid \dots \mid \mathbf{A}_m(\mathbf{i}_m, \mathbf{o}_m) : Q_m.\mathscr{E}_m \parallel \sigma$, abbreviated as $\mathscr{N} = |_i\, \mathbf{A}_i(\mathbf{i}_i, \mathbf{o}_i) : Q_i.\mathscr{E}_i \parallel \sigma$. The *configuration* C of a network $\mathscr{N}$ at a particular point in time is described by a set of agents, their classical local states, and the quantum state σ, formally $C = \sigma, \Gamma_1, \mathbf{a}_1 \mid \Gamma_2, \mathbf{a}_2 \mid \dots \mid \Gamma_m, \mathbf{a}_m$, abbreviated as $C = \sigma, |_i\, \Gamma_i, \mathbf{a}_i$, where Γ_i represents the classical state of agent $\mathbf{a}_i$, which is defined as a partial mapping from classical variables to values. The set $\mathscr{C}_{\mathscr{N}}$ contains all configurations that potentially occur during the execution of the network $\mathscr{N}$.

Operational and denotational semantics for DMC are defined in [9]; however, here we are more interested in its small-step semantics. The following small-step rules for configuration transitions describe how the network evolves over time. If the quantum state does not change in an evaluation step, the writing $\sigma \vdash$ precedes the rule. Also, we use a shorthand notation for agents: $\mathbf{a}_i = \mathbf{A}_i : Q_i.\mathscr{E}_i$, $\mathbf{a}_i.E = \mathbf{A}_i : Q_i.[\mathscr{E}_i.E]$,

$\mathbf{a}^{-q} = \mathbf{A} : Q \backslash q.\mathscr{E}$, and $\mathbf{a}^{+q} = \mathbf{A} : Q \uplus q.\mathscr{E}[q/x]$, where E is some event.

$$\frac{\sigma, \mathscr{P}(V,I,O,\mathscr{A}) \longrightarrow_\lambda \sigma', \Gamma'}{\sigma, \Gamma, \mathbf{A} : I \uplus R.[\mathscr{E}.\mathscr{P}] \Longrightarrow_\lambda \sigma', \Gamma \cup \Gamma', \mathbf{A} : O \uplus R.\mathscr{E}} \tag{1}$$

$$\frac{\Gamma_2(y) = v}{\sigma \vdash (\Gamma_1, \mathbf{a}_1.\mathrm{c}?x \mid \Gamma_2, \mathbf{a}_2.\mathrm{c}!y \Longrightarrow \Gamma_1[x \mapsto v], \mathbf{a}_1 \mid \Gamma_2, \mathbf{a}_2)} \tag{2}$$

$$\frac{}{\sigma \vdash (\Gamma_1, \mathbf{a}_1.\mathrm{qc}?x \mid \Gamma_2, \mathbf{a}_2.\mathrm{qc}!q \Longrightarrow \Gamma_1, \mathbf{a}_1^{+q} \mid \Gamma_2, \mathbf{a}_2^{-q})} \tag{3}$$

$$\frac{L \Longrightarrow_\lambda R}{L \mid L' \Longrightarrow_\lambda R \mid L'} \tag{4}$$

The first rule refers to local operations. Since a pattern's big-step semantics is given by a probabilistic transition system, described by $\longrightarrow$, a probability λ is introduced here. Also, an agent changes its sort depending on the pattern's output O. The next two rules are for the classical and the quantum rendez-vous. For the quantum rendez-vous a substitution q for x in the event sequence of the receiving agent is performed and agents need to update their qubit sorts. (4) is a metarule, which is required to express that any of the other rules may fire in the context of a larger system.

2.2 Interpreted Systems and MCMAS

Interpreted systems [14] are the typical formalism for reasoning about time and knowledge in multi-agent systems. In IS each agent i from a non-empty set Ag of agents is modelled by a set of local states L_i, a set of actions Act_i that she may perform according to her protocol function P_i, and an evolution function t_i. A special agent E, representing the environment in which the other agents operate, is also described by a set of local states L_E, a set of actions Act_E, a protocol P_E, and an evolution function t_E. For every $j \in Ag \cup \{E\}$, the protocol P_j is defined as a function $P_j : L_j \to 2^{Act_j}$, assigning a set of actions to a given local state. Intuitively, $\alpha_j \in P_j(l_j)$ means that action α_j is enabled in l_j. The evolution function t_j is a transition function returning the target local state given the current local state and the set of actions performed by all agents, formally $t_j : L_j \times Act_1 \times \cdots \times Act_n \times Act_E \to L_j$ under the constraint $\alpha_j \in P_j(l_j)$. Agents evolve simultaneously in every state of the system according to the joint transition function t.

The set Act of joint actions is defined as the Cartesian product of all agents' actions, formally $Act = Act_1 \times \cdots \times Act_n \times Act_E$. The Cartesian product $S = L_i \times \cdots \times L_n \times L_E$ of the agents' local states is the set of all global states of the system. The local state of agent i in the global state $g \in S$ is denoted as $l_i(g)$. The description of an interpreted system is concluded by including a set of atomic propositions $AP = \{p_1, p_2, \ldots\}$ and an evaluation relation $V \subseteq AP \times S$. Formally, an interpreted system is defined as a tuple $IS = \langle (L_i, Act_i, P_i, t_i)_{i \in Ag}, (L_E, Act_E, P_E, t_E), V \rangle$.

Interpreted systems can be used to interpret CTLK, a logic combining the branching-time temporal logic CTL with epistemic modalities. The formal language $\mathscr{L}$ is built from propositional atoms $p \in AP$ and agents $i \in Ag$ as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\varphi \mid EG\varphi \mid E\varphi U \psi \mid K_i$$

The formulae in $\mathscr{L}$ have the following intuitive meaning. $EX\varphi$: there is a path where φ holds in the next state; $EG\varphi$: there is a path where φ always holds; $E\varphi U\psi$: there is a path where φ holds at least until at some state ψ holds; $K_i\varphi$: agent i knows φ. The other standard CTL formulae, e.g., $AF\varphi$: for all paths φ eventually holds, can be derived from the above. The formal definition of satisfaction in interpreted systems follows.

In an interpreted system $\mathscr{M}$ the evolution function t determines a transition relation $\xrightarrow{\mathscr{M}}$ on states such that $s \xrightarrow{\mathscr{M}} s'$ iff there is a joint action $\alpha \in Act$ such that $t(s,\alpha) = s'$. A *path* π is an infinite sequence of states $s_0 \xrightarrow{\mathscr{M}} s_1 \xrightarrow{\mathscr{M}} \ldots$. Further, π^n denotes the n-th state in the sequence, i.e, s_n. Finally, for each agent $i \in Ag$, we introduce the epistemic equivalence relation $\sim_i^{\mathscr{M}}$ such that $s \sim_i^{\mathscr{M}} s'$ iff $l_i(s) = l_i(s')$.

Given the IS $\mathscr{M}$, a state s, and a formula $\phi \in \mathscr{L}$, the satisfaction relation $\vDash$ is defined as follows:

$(\mathscr{M},s) \vDash p$ iff $V(p,s)$
$(\mathscr{M},s) \vDash \neg\phi$ iff $(\mathscr{M},s) \nvDash \phi$
$(\mathscr{M},s) \vDash \phi \vee \phi'$ iff $(\mathscr{M},s) \vDash \phi$ or $(\mathscr{M},s) \vDash \phi'$
$(\mathscr{M},s) \vDash EX\phi$ iff there is a path π such that $\pi^0 = s$, and $(\mathscr{M},\pi^1) \vDash \phi$
$(\mathscr{M},s) \vDash EG\phi$ iff there is a path π such that $\pi^0 = s$, and for all $n \in \mathbb{N}$, $(\mathscr{M},\pi^n) \vDash \phi$
$(\mathscr{M},s) \vDash E\phi U\phi'$ iff there is a path π such that $\pi^0 = s$, for some $n \in \mathbb{N}$, $(\mathscr{M},\pi^n) \vDash \phi'$, and for all n', $0 \leq n' < n$ implies $(\mathscr{M},\pi^{n'}) \vDash \phi$
$(\mathscr{M},s) \vDash K_i\phi$ iff for all $s' \in S$, $s \sim_i^{\mathscr{M}} s'$ implies $(\mathscr{M},s') \vDash \phi$

A formula $\phi \in \mathscr{L}$ is true in an IS $\mathscr{M}$, or $\mathscr{M} \vDash \phi$, iff for all $s \in S$, $(\mathscr{M},s) \vDash \phi$.

In [17] the authors present a methodology for the verification of IS based on model checking [6] via ordered binary decision diagrams. These verification techniques have been implemented in the MCMAS model checker. The input to the model checker is given as an ISPL program, which is essentially a machine readable IS.

2.3 Quantum Epistemic Logic

A formal framework for reasoning about temporal and epistemic properties of distributed quantum systems was developed in [10] on top of DMC. The authors argue that quantum knowledge is not a meaningful concept, but it is of interest to reason about classical knowledge pertaining to a quantum system. In this sense, the quantum information possessed by an agent concerns the qubits she owns, the local operations she applies to these qubits, the non-local entanglement she shares initially, and possibly prior knowledge of her local quantum inputs. All this information is contained in her local state Γ_i and her event sequence $\mathscr{E}_i$. Given a network $\mathscr{N}$, the epistemic accessibility relation $\sim_i^{\mathscr{N}}$ for an agent $\mathbf{A}_i$ is defined in [10] as follows: for all configurations $C = \sigma, |_i\, \Gamma_i, \mathbf{A}_i : Q_i.\mathscr{E}_i$ and $C' = \sigma', |_i\, \Gamma_i', \mathbf{A}_i : Q_i'.\mathscr{E}_i'$ in $\mathscr{C}_{\mathscr{N}}$, C and C' are *indistinguishable to agent* $\mathbf{A}_i$, written as $C \sim_i^{\mathscr{N}} C'$, if $\Gamma_i = \Gamma_i'$ and $\mathscr{E}_i = \mathscr{E}_i'$. The semantics for the modal operator K_i for the knowledge of agent $\mathbf{A}_i$ is then defined in the usual way: $(C,\mathscr{N}) \vDash K_i\varphi$ iff for all C', $C' \sim_i^{\mathscr{N}} C$ implies $(C',\mathscr{N}) \vDash \varphi$.

We now give the truth conditions for all formulae in $\mathscr{L}$ in a network $\mathscr{N}$.The set of atomic propositions $AP = \{x = v, x = y, \mathbf{A}_i \text{ has } q, q_1 \ldots q_n = |\psi\rangle, q_i = q_j\}$ is considered in [7]. In a configuration C of a network $\mathscr{N}$ the truth conditions for these atomic propositions are given as follows:

$(C,\mathscr{N}) \vDash x = v$ iff there is an agent i such that $\Gamma_i(x) = v$
$(C,\mathscr{N}) \vDash x = y$ iff there are agents i, j such that $\Gamma_i(x) = \Gamma_i(y)$
$(C,\mathscr{N}) \vDash \mathbf{A}_i \text{ has } q$ iff $q \in Q_i$
$(C,\mathscr{N}) \vDash q_1 \ldots q_n = |\psi\rangle$ iff $q_1 \ldots q_n = |\psi\rangle$
$(C,\mathscr{N}) \vDash q_i = q_j$ iff there is $|\psi\rangle$ such that $|\psi\rangle = q_i = q_j$

In networks the small-step rules given in Section 2.1 determine a transition relation $\xrightarrow{\mathscr{N}}$ such that $C \xrightarrow{\mathscr{N}} C'$ iff there is a rule that applied to C returns C'. A *path* γ is an infinite sequence of configurations $C_0 \xrightarrow{\mathscr{N}} C_1 \xrightarrow{\mathscr{N}} \ldots$. Further, γ^n denotes the n-th state in the sequence, i.e, C_n. Finally, for each agent $\mathbf{A}_i$

in the network, we introduce the epistemic equivalence relation $\sim_i^{\mathscr{N}}$ such that $C \sim_i^{\mathscr{N}} C'$ iff $\Gamma_i = \Gamma_i'$ and $\mathscr{E}_i = \mathscr{E}_i'$.

Given the network $\mathscr{N}$, a configuration C, and a formula $\phi \in \mathscr{L}$, the satisfaction relation $\models$ is defined as follows:

$(C,\mathscr{N}) \models p$ iff C satisfies the corresponding condition above for atomic $p \in AP$
$(C,\mathscr{N}) \models \neg\phi$ iff $(C,\mathscr{N}) \not\models \phi$
$(C,\mathscr{N}) \models \phi \vee \phi'$ iff $(C,\mathscr{N}) \models \phi$ or $(C,\mathscr{N}) \models \phi'$
$(C,\mathscr{N}) \models EX\phi$ iff there is a path γ such that $\gamma^0 = C$, and $(\gamma^1,\mathscr{N}) \models \phi$
$(C,\mathscr{N}) \models EG\phi$ iff there is a path γ such that $\gamma^0 = C$, and for all $n \in \mathbb{N}$, $(\gamma^n,\mathscr{N}) \models \phi$
$(C,\mathscr{N}) \models E\phi U\phi'$ iff there is a path γ such that $\gamma^0 = C$, for some $n \in \mathbb{N}$, $(\gamma^n,\mathscr{N}) \models \phi'$, and for all n', $0 \leq n' < n$ implies $(\gamma^{n'},\mathscr{N}) \models \phi$
$(C,\mathscr{N}) \models K_i\phi$ iff for all $C' \in \mathscr{N}$, $C \sim_i^{\mathscr{N}} C'$ implies $(C',\mathscr{N}) \models \phi$

A formula $\phi \in \mathscr{L}$ is true in a network $\mathscr{N}$, or $\mathscr{N} \models \phi$, iff for all configurations C, $(C,\mathscr{N}) \models \phi$.

2.4 Quantum Teleportation Protocol

The goal of the Quantum Teleportation Protocol (QTP) is to transmit a qubit from one party to another with the aid of an entangled pair of qubits and classical resources. For reasons of space we refer to [5] for a detailed presentation of QTP. The DMC specification of the protocol is given in [9] as:

$$\mathscr{N}_{QTP} = \mathbf{A} : \{1,2\}.[(\mathrm{c}!s_2s_1).M_{12}^{0,0}E_{12}] \mid \mathbf{B} : \{3\}.[X_3^{x_2}Z_3^{x_1}.(\mathrm{c}?x_2x_1)] \parallel E_{23}.$$

The informal reading is as follows: Alice **A** and Bob **B** share the entangled pair E_{23} of qubits 2 and 3, and Alice wants to transmit the input qubit 1. In the first step, she entangles (E_{12}) her qubits 1 and 2. Then she measures $(M_{12}^{0,0})$ both of them. Next, she sends via classical communication $(\mathrm{c}!s_2s_1)$ the measurement outcomes to Bob. Upon receipt $(\mathrm{c}?x_2x_1)$, Bob applies corrections $(X_3^{x_2}Z_3^{x_1})$ to his qubit 3 depending on these measurements. The result is that Bob's qubit 3 is guaranteed to be in the same state as Alice's input qubit 1.

3 Formal Mapping

In this section we present a methodology for translating a protocol specified in DMC into the corresponding IS. Formally, we define a mapping $f : DMC \to IS$, such that f preserves satisfaction of formulae in the specification language $\mathscr{L}$. First, we describe the translation of the global quantum state and classical states of agents. Then we cover the rules in DMC. Finally, we show that f is sound.

3.1 Classical States of Agents and Global Quantum State

Given a network $\mathscr{N}$ we introduce an agent $i \in Ag$ for each agent $\mathbf{A}_i(\mathbf{i}_i,\mathbf{o}_i) : Q_i.\mathscr{E}_i$ in $\mathscr{N}$, as well as the Environment agent E. We take a local state $l_i \in L_i$ of agent i to be a tuple of vector variables $(\vec{x},\vec{y},\vec{s},\vec{q},pc)$ defined as follows:

- Each classical input bit in $\mathbf{i}_i$ is mapped to a variable $y \in l_i$ in the domain $\{0,1\}$.
- A bit received from an agent via the classical receive event $\mathrm{c}?x$ in the event sequence $\mathscr{E}_i$ is mapped to a variable $x \in l_i$ in the domain $\{0,1,\perp\}$, where $\perp$ denotes the undefined value before communication.

- A variable $s \in l_i$, called *signal*, represents the outcome of a measurement event M_q^α in the event sequence $\mathscr{E}_i$, where q is the measured qubit and α is a measurement angle. A signal can attain values $\{0,1,\perp\}$, where $\perp$ denotes the undefined value of the signal before the agent executes the measurement.
- A variable $q \in l_i$ in the domain $\{0,1,2\}$ represents the ownership relation between agent $\mathbf{A}_i$ and qubit q with the following meaning: if $\mathbf{A}_i$ is not in possession of q, i.e., $q \notin Q_i$, then we take $q = 0$. If $\mathbf{A}_i$ owns the qubit q, i.e., $q \in Q_i$, then $q = 1$ or $q = 2$. The former value represents that $\mathbf{A}_i$ does not know the exact state of the qubit, the latter value represents that she knows it. We assume that the agent knows the state of the qubit once she measures it or prepares it in a specific state. This is motivated as there is classical information involved in both cases. However, the agent loses this knowledge when she sends the qubit to another agent, as it is no longer in her possession, or entangles it with another qubit. Note that correction commands preserve knowledge because they are deterministic actions that neither entangle nor separate qubits.
- $pc \in l_i$ is a counter for the events in the event sequence $\mathscr{E}_i$ executed by agent $\mathbf{A}_i$.

Example 1. Consider the specification of QTP in DMC as given in Section 2.4. The local state of Alice is described by the tuple $l_A = (s_1, s_2, q_1, q_2, q_3, pc)$, and similarly the local state of Bob is $l_B = (x_1, x_2, q_1, q_2, q_3, pc)$. In the initial state Alice owns the input qubits q_1 and q_2 in the entangled pair, while Bob owns the qubit q_3, and neither of them knows anything about their qubits. Alice has not yet measured any qubit nor has she sent anything to Bob. The program counters of both agents point to the first event in their event sequences. All this is captured in variable assignments $(\perp, \perp, 1, 1, 0, 1)$ for Alice and $(\perp, \perp, 0, 0, 1, 1)$ for Bob.

A local state $l_E \in L_E$ of the Environment represents the quantum state σ of the network. l_E is a tuple of vector variables $(\vec{q}, \vec{q}', \vec{e}, gc)$ defined as follows:

- We divide the global quantum state at any given time into the smallest possible substates - individual qubits and/or systems of entangled qubits - such that these are in pure states, i.e., they can be represented as a vector in a Hilbert space. We generate the reachable quantum state space of the network using the small-step rules for patterns and enumerate all such encountered substates. Thus, every reachable substate has an associated name qs_n, $n \in \mathbb{N}$.
- For every qubit $q \in \mathscr{N}$ we introduce a variable $q \in l_E$. The domain of q is the set of names of quantum states that q may attain in any run of the protocol, together with the value $\perp$ indicating that the qubit is not in a pure state but entangled with other qubits.
- Similarly, for every system of entangled qubits we introduce a variable $e \in l_E$. The domain of e is the set of names of quantum states that the system may attain, together with the value $\perp$ indicating that either the system is not in a pure state or its qubits are not entangled.
- Each variable q and e is assigned a name if only if they are pure and cannot be further separated. Otherwise, they are assigned the value $\perp$. The global state σ is then the tensor product of these substates.
- In addition, we make use of an auxiliary variable q' for each qubit $q \in \mathscr{N}$ recording the name of its initial state, and introduce the global counter $gc \in l_E$ that increases with every action in the network. This is used to track the global time and to enumerate the configurations in $\mathscr{C}_\mathscr{N}$ according to their occurrence in the path.

Action	Qubit/Entangled System	State	Name
Initially	q_1	$[a,b]^T$	qs_1
	e_{23}	$\frac{1}{2}[1,1,1,-1]^T$	qs_2
E_{12}	e_{123}	$\frac{1}{2}[a,a,a,-a,b,b,-b,b]^T$	qs_3
M_1^0	q_1	$\frac{1}{2}[\sqrt{2},\sqrt{2}]^T$	qs_4
		$\frac{1}{2}[\sqrt{2},-\sqrt{2}]^T$	qs_5
	e_{23}	$\frac{1}{2}[a+b,a+b,a-b,-a+b]^T$	qs_6
		$\frac{1}{2}[a-b,a-b,a+b,-a-b]^T$	qs_7
M_2^0	q_2	$\frac{1}{2}[\sqrt{2},\sqrt{2}]^T$	qs_4
		$\frac{1}{2}[\sqrt{2},-\sqrt{2}]^T$	qs_5
	q_3	$[a,b]^T$	qs_1
		$[a,-b]^T$	qs_8
		$[b,a]^T$	qs_9
		$[-b,a]^T$	qs_{10}
$X_3^{x_2}Z_3^{x_1}$	q_3	$[a,b]^T$	qs_1

Table 1: Enumeration of quantum substates in the evolution of QTP.

Example 2. The global quantum state of QTP is represented in the local state of the Environment E as the tuple $l_E = (q_1,q_2,q_3,q_1',q_2',q_3',e_{23},e_{123},gc)$. The initial state of the input qubit q_1 is $[a,b]^T$, for $a,b \in \mathbb{C}$. We assume that it is not equal to states $[1,0]^T$ and $[0,1]^T$ of the standard basis, nor to states $\frac{1}{2}[\sqrt{2},\sqrt{2}]^T$ and $\frac{1}{2}[\sqrt{2},-\sqrt{2}]^T$ of the measurement basis. In these cases there are fewer states, but the procedure is analogous. Table 1 shows the enumeration of substates occurring in all possible runs of the network, as Alice and Bob execute quantum commands according to QTP. For instance, the initial state of the network is $(qs_1,\perp,\perp,qs_1,\perp,\perp,qs_2,\perp,1)$. Note that only the input qubit q_1 and the system of two entangled qubit e_{23} have assigned named states. This is because the individual qubits q_2 and q_3 are not in a pure state and the system of all three qubit e_{123} can be further separated. Indeed, the whole quantum state can be expressed as the tensor product $[a,b]^T \otimes \frac{1}{2}[1,1,1,-1]^T$, or by using names $qs_1 \otimes qs_2$.

3.2 Transition Rules

Events in the event sequence $\mathscr{E}_i$ of agent $\mathbf{A}_i$ are mapped into actions in Act_i. Actions are executed according to a protocol function P_i and their effects are described by evolution functions t_i and t_E depending on whether the classical state of agent $\mathbf{A}_i$ changes, or the quantum state σ of the system changes, or both. Before introducing the mapping for events, note that DMC is a probabilistic calculus, whereas IS have a Boolean semantics. We deal with this issue by allowing all admissible transitions, abstracting away from the probability distribution. As a result, we lose the ability to reason about the probability of reaching a state. However, this is not an issue for us as we need to reason about non-probabilistic properties only as the choice of the language $\mathscr{L}$ demonstrates.

Note also that the execution of a pattern $\mathscr{P}$ in DMC occurs in a single transition step and depends on the big-step semantics of the pattern (see Rule 1). However, we handle transitions at the level of individual commands of $\mathscr{P}$, and so the execution depends on the small-step semantics of patterns and may span across several time steps. This leads to a finely grained state space. In the rest of this section we present the actions, the protocols, and the evolution functions associated with the classical and quantum communication and the quantum commands presented in Section 2.1.

Classical rendez-vous. Assume that agent $\mathbf{A}_i$ sends the value of y to agent $\mathbf{A}_j$ who stores it in x, specified in DMC as $\Gamma_i,\mathbf{A}_i : Q_i.\mathrm{c}!y$ and $\Gamma_j,\mathbf{A}_j : Q_j.\mathrm{c}?x$, and that this is the vth (resp. wth) event in $\mathscr{E}_i$

(resp. $\mathscr{E}_j$). We translate this by considering the actions $snd\_j\_y_0$ and $snd\_j\_y_1$ in the set Act_i of actions for agent i, and action rcv_i_x in Act_j. The protocol functions are:

$$P_i(l_i) = \{snd\_j\_y_0\}, \text{ if } pc = v \wedge y = 0,$$
$$P_i(l_i) = \{snd\_j\_y_1\}, \text{ if } pc = v \wedge y = 1,$$
$$P_j(l_j) = \{rcv\_i\_x\}, \text{ if } pc = w.$$

The configuration transition, described by Rule 2, is translated into the following evolution functions for the agents i and j:

$$t_i(l_i, Act_i, Act_j) = pc \mapsto pc + 1, \text{ if } (Act_i = snd\_j\_y_0 \vee Act_i = snd\_j\_y_1) \wedge Act_j = rcv\_i\_x,$$
$$t_j(l_j, Act_i, Act_j) = pc \mapsto pc + 1 \wedge x \mapsto 0, \text{ if } Act_i = snd\_j\_y_0 \wedge Act_j = rcv\_i\_x,$$
$$t_j(l_j, Act_i, Act_j) = pc \mapsto pc + 1 \wedge x \mapsto 1, \text{ if } Act_i = snd\_j\_y_1 \wedge Act_j = rcv\_i\_x.$$

The rationale behind the above equations is that when agents perform paired send/receive actions at the same time step, their program counters are incremented, and variable x of agent $\mathbf{A}_j$ is assigned the transmitted value.

Quantum communication. Assume that agent $\mathbf{A}_i$ sends a qubit $q \in Q_i$ to agent $\mathbf{A}_j$, described as $\Gamma_i, \mathbf{A}_i : Q_i.\text{qc!}q$ and $\Gamma_j, \mathbf{A}_j : Q_j.\text{qc?}q$, and that this is the vth (resp. wth) event in $\mathscr{E}_i$ (resp. $\mathscr{E}_j$). We introduce actions $qsnd\_j\_q$ and $qrcv\_i\_q$ in Act_i and Act_j respectively. The protocol functions are:

$$P_i(l_i) = \{qsnd\_j\_q\}, \text{ if } pc = v,$$
$$P_j(l_j) = \{qrcv\_i\_q\}, \text{ if } pc = w.$$

Rule 3 defines the configuration transition in terms of sets of qubits Q_i and Q_j. When $\mathbf{A}_i$ sends the qubit q, it is removed from her set, and when $\mathbf{A}_j$ receives q, it is added to her set. This is translated into IS by the evolution functions:

$$t_i(l_i, Act_i, Act_j) = pc \mapsto pc + 1 \wedge q \mapsto 0, \text{ if } Act_i = qsnd\_j\_q \wedge Act_j = qrcv\_i\_q,$$
$$t_j(l_j, Act_i, Act_j) = pc \mapsto pc + 1 \wedge q \mapsto 1, \text{ if } Act_i = qsnd\_j\_q \wedge Act_j = qrcv\_i\_q.$$

This means that when both agents concurrently execute the respective quantum communication events, their local program counters are incremented, and the ownership of the qubit changes, i.e., $\mathbf{A}_i$ is no longer in possession of q while $\mathbf{A}_j$ owns it but does not know its state.

Corrections. The events X_q^s and Z_q^s differ only in their matrix representations, so we describe them together. Assume that agent $\mathbf{A}_i$ executes the Pauli operator X or the Pauli operator Z on a qubit q at step v of $\mathscr{E}_i$ if signal $s = 1$, otherwise she skips the event. This scenario has the following DMC description: $\Gamma_i, \mathbf{A}_i : q \uplus R_i.U_q^s$, with $U_q^s \in \{X_q^s, Z_q^s\}$. We introduce actions x_q and z_q in Act_i, and since the agent applies the event conditionally, we also include the action $skip$. In the rest of the description we refer to both actions x_q and z_q as u_q. The protocol function is then given as:

$$P_i(l_i) = \{skip\} \qquad \text{if } pc = v \wedge s = 0;$$
$$P_i(l_i) = \{u\_q\} \qquad \text{if } pc = v \wedge s = 1.$$

For example, we have the following ground protocol function for Bob in QTP:

$$P_B(l_B) = \{skip\}, \text{ if } pc = 3 \wedge x_1 = 0; \qquad P_B(l_B) = \{skip\}, \text{ if } pc = 4 \wedge x_2 = 0;$$
$$P_B(l_B) = \{z\_q_3\}, \text{ if } pc = 3 \wedge x_1 = 1; \qquad P_B(l_B) = \{x\_q_3\}, \text{ if } pc = 4 \wedge x_2 = 1.$$

The small-step semantics for corrections is defined as $\sigma,\Gamma_i \xrightarrow{U_q^s} U_r^{s_{\Gamma_i}}\sigma,\Gamma_i$. Assume that the qubit q is in system e, which again may be just q or some entangled system. The local state of the agent $\mathbf{A}_i$ changes only through the *pc* increment. We define the evolution functions as:

$$t_E(l_E,Act_i) = gc \mapsto gc+1 \wedge e \mapsto qs_y, \text{ if } e = qs_x \wedge Act_i = u\_q;$$
$$t_i(l_i,Act_i) = pc \mapsto pc+1, \text{ if } Act_i = u\_q \vee Act_i = skip;$$

where qs_x (resp. qs_y) is the name of the state before (resp. after) the execution. The ground evolution function of E in QTP with respect to Bob's corrections $X_3^{x_2}Z_3^{x_1}$ is given as the following equations corresponding to measurement outcomes $x_1x_2 \mapsto 10$, $x_1x_2 \mapsto 01$, and $x_1x_2 \mapsto 11$ respectively. Note that in the last case Bob executes both actions z_q_3 and x_q_3 sequentially, while in the first two cases he executes only one of them and skips the other.

$$t_E(l_E,Act_B) = gc \mapsto gc+1 \wedge q_3 \mapsto qs_1, \text{ if } q_3 = qs_8 \wedge Act_B = z\_q_3;$$
$$t_E(l_E,Act_B) = gc \mapsto gc+1 \wedge q_3 \mapsto qs_1, \text{ if } q_3 = qs_9 \wedge Act_B = x\_q_3;$$
$$t_E(l_E,Act_B) = gc \mapsto gc+1 \wedge q_3 \mapsto qs_9, \text{ if } q_3 = qs_{10} \wedge Act_B = z\_q_3.$$

Entanglement. Assume that agent $\mathbf{A}_i$ applies at step v of $\mathscr{E}_i$ the entanglement operator E_{qr} on qubits q and r. The DMC definition of the agent in this case is $\Gamma_i, \mathbf{A}_i : q,r \uplus R_i.E_{qr}$. Since this event is independent of signals, we add only one corresponding action ent_q_r to Act_i and define the following protocol function: $P_i(l_i) = \{ent\_q\_r\}$, if $pc = v$. The small-step rule for entanglement is given as $\sigma,\Gamma_i \xrightarrow{E_{qr}} {}^CZ_{qr}\sigma,\Gamma_i$, where ${}^CZ_{qr}$ is the controlled-Z operator realising the entanglement. Since we divide the global state σ into its smallest pure substates, we have two possible situations. In the first case $q \in e'$ and $r \in e''$, where e' and e'' are isolated qubits, distinct entangled systems, or combination of both. The resulting entangled system e is the union of the two systems e' and e'', and we define the evolution function of the Environment E as:

$$t_E(l_E,Act_i) = gc \mapsto gc+1 \wedge e \mapsto qs_z \wedge e' \mapsto \perp \wedge e'' \mapsto \perp, \text{ if } e' = qc_x \wedge e'' = qc_y \wedge Act_i = ent\_q\_r;$$

where qs_x, qs_y, and qs_z are the names of the quantum states in which the systems e', e'' are during the execution of the event, and e after the execution. For instance, the ground evolution function in QTP for Alice's entanglement E_{12} is:

$$\begin{aligned} t_E(l_E,Act_A) \quad = \quad & gc \mapsto gc+1 \wedge e_{123} \mapsto qs_3 \wedge q_1 \mapsto \perp \wedge e_{23} \mapsto \perp, \\ & \text{if } q_1 = qc_1 \wedge e_{23} = qc_2 \wedge Act_A = ent\_q_1\_q_2. \end{aligned}$$

Note that there may be many possible combinations of various states for e' and e'', and we have to define the evolution function for all of them. In the second case the qubits q and r are part of the same system e and we simply have the evolution function:

$$t_E(l_E,Act_i) = gc \mapsto gc+1 \wedge e \mapsto qs_y, \text{ if } e = qs_x \wedge Act_i = ent\_q\_r;$$

where qs_x (resp. qs_y) is the name of the state before (resp. after) the execution. In both cases the local state of agent $\mathbf{A}_i$ is updated as follows:

$$t_i(l_i,Act_i) = pc \mapsto pc+1 \wedge q \mapsto 1 \wedge r \mapsto 1, \text{ if } Act_i = ent\_q\_r.$$

This equation states that the counter of $\mathbf{A}_i$ is incremented and the agent loses any knowledge about the state of q and r she might have had, since neither qubit is in a pure state anymore.

Measurement. This is a complex event modifying the quantum state of the network as well as the local states of agents. Suppose that agent $\mathbf{A}_i$ in step v of $\mathscr{E}_i$ measures her qubit q in the $\{|+\alpha\rangle, |-\alpha\rangle\}$ basis, specified in DMC as $\Gamma_i, \mathbf{A}_i : q \uplus R_i.{}^t[M_q^\alpha]^s$, where s and t are signals. A measurement is a stochastic event

$\emptyset\ \emptyset$	Actions	$m\_q\_{+\alpha}, m\_q\_{-\alpha}$
	Protocol	$P_i(l_i(g)) = \{m\_q\_{+\alpha}, m\_q\_{-\alpha}\}$, if $pc = v$
$s\ \emptyset$	Actions	$m\_q\_s0\_{+\alpha}, m\_q\_s0\_{-\alpha}, m\_q\_s1\_{+\alpha}, m\_q\_s1\_{-\alpha}$
	Protocol	$P_i(l_i(g)) = \{m\_q\_s0\_{+\alpha}, m\_q\_s0\_{-\alpha}\}$, if $pc = v \wedge s = 0$ $P_i(l_i(g)) = \{m\_q\_s1\_{+\alpha}, m\_q\_s1\_{-\alpha}\}$, if $pc = v \wedge s = 1$
$\emptyset\ t$	Actions	$m\_q\_t0\_{+\alpha}, m\_q\_t0\_{-\alpha}, m\_q\_t1\_{+\alpha}, m\_q\_t1\_{-\alpha}$
	Protocol	$P_i(l_i(g)) = \{m\_q\_t0\_{+\alpha}, m\_q\_t0\_{-\alpha}\}$, if $pc = v \wedge t = 0$ $P_i(l_i(g)) = \{m\_q\_t1\_{+\alpha}, m\_q\_t1\_{-\alpha}\}$, if $pc = v \wedge t = 1$
$s\ t$	Actions	$m\_q\_s0\_t0\_{+\alpha}, m\_q\_s0\_t0\_{-\alpha}, m\_q\_s0\_t1\_{+\alpha}, m\_q\_s0\_t1\_{-\alpha},$ $m\_q\_s1\_t0\_{+\alpha}, m\_q\_s1\_t0\_{-\alpha}, m\_q\_s1\_t1\_{+\alpha}, m\_q\_s1\_t1\_{-\alpha}$
	Protocol	$P_i(l_i(g)) = \{m\_q\_s0\_t0\_{+\alpha}, m\_q\_s0\_t0\_{-\alpha}\}$, if $pc = v \wedge s = 0 \wedge t = 0$ $P_i(l_i(g)) = \{m\_q\_s0\_t1\_{+\alpha}, m\_q\_s0\_t1\_{-\alpha}\}$, if $pc = v \wedge s = 0 \wedge t = 1$ $P_i(l_i(g)) = \{m\_q\_s1\_t0\_{+\alpha}, m\_q\_s1\_t0\_{-\alpha}\}$, if $pc = v \wedge s = 1 \wedge t = 0$ $P_i(l_i(g)) = \{m\_q\_s1\_t1\_{+\alpha}, m\_q\_s1\_t1\_{-\alpha}\}$, if $pc = v \wedge s = 1 \wedge t = 1$

Table 2: Actions and protocol rules for various degree of dependency of measurements.

and may also depend on signals s and t. We express this non-determinism by associating two actions to a given local state l_i of agent i. However, due to a possible dependency on signals s and t, there are four different sets of actions and protocol rules. We list them in Table 2, where $\emptyset$ means that the measurement does not depend on a particular signal.

The following two transitions are defined in the small-step semantics for the measurement event: $\sigma, \Gamma_i \xrightarrow{t[M_q^\alpha]^r} \lambda\ \langle +_{\alpha_{\Gamma_i}}|_q \sigma, \Gamma_i[0/q]$ and $\sigma, \Gamma_i \xrightarrow{t[M_r^\alpha]^{s_1}} \lambda\ \langle -_{\alpha_{\Gamma_i}}|_q \sigma, \Gamma_i[1/q]$. This is the source of non-determinism in the transition system, but we do not consider the probability λ as long as it is non-zero.

There are again four types of evolution functions. They differ in the computation of quantum states, and since we give only the general rules, here we describe the evolution functions only for the independent measurement, i.e., when $s = t = \emptyset$. As far as the translation rules are concerned, the other three types differ only in the names of the actions and the actual names of quantum states. We can translate them analogously.

We now consider two cases where both measurement outcome are possible. First, for the measurement of an isolated qubit q we define the evolution function of the Environment E as follows:

$$t_E(l_E, Act_i) = gc \mapsto gc + 1 \wedge q \mapsto qs_{+_\alpha}, \text{ if } q = qc_x \wedge Act_i = m\_q\_{+\alpha};$$
$$t_E(l_E, Act_i) = gc \mapsto gc + 1 \wedge q \mapsto qs_{-_\alpha}, \text{ if } q = qc_x \wedge Act_i = m\_q\_{-\alpha};$$

where qs_{+_α} and qs_{-_α} are names of the $\{|+_\alpha\rangle, |-_\alpha\rangle\}$ measurement basis. If the qubit q is part of an entangled system e, then the system becomes separated on measurement. The measured qubit q collapses and the rest of qubits form a new system e'. We define the evolution function as follows:

$$t_E(l_E, Act_i) = gc \mapsto gc + 1 \wedge q \mapsto qs_{+_\alpha} \wedge e \mapsto \perp \wedge e' \mapsto qs_y, \text{ if } e = qc_x \wedge Act_i = m\_q\_{+\alpha},$$
$$t_E(l_E, Act_i) = gc \mapsto gc + 1 \wedge q \mapsto qs_{-_\alpha} \wedge e \mapsto \perp \wedge e' \mapsto qs_z, \text{ if } e = qc_x \wedge Act_i = m\_q\_{-\alpha}\,.$$

In both cases the measurement outcome is assigned to a signal variable s' of agent $\mathbf{A}_i$ and her evolution function is given by:

$$t_i(l_i, Act_i) = pc \mapsto pc + 1 \wedge s' \mapsto 0 \wedge q \mapsto 2, \text{ if } Act_i = m\_q\_{+\alpha};$$
$$t_i(l_i, Act_i) = pc \mapsto pc + 1 \wedge s' \mapsto 1 \wedge q \mapsto 2, \text{ if } Act_i = m\_q\_{-\alpha}\,.$$

For instance, consider the first measurement that Alice performs in QTP. All three qubits are entangled together and therefore measuring the input qubit q_1 causes separation of the system e_{123} into two parts,

q_1 and e_{23}, and has two possible outcomes. Both have probability $\lambda = 0.5$, but we do not take this into account since all we require is that they are non-zero, therefore the respective transitions are admissible. We have the following ground evolution functions for the Environment and Alice:

$$t_E(l_E, Act_A) = gc \mapsto gc+1 \wedge q_1 \mapsto qs_4 \wedge e_{123} \mapsto \bot \wedge e_{23} \mapsto qs_6, \text{ if } e_{123} = qc_3 \wedge Act_A = m\_q_{1\_}+\alpha;$$
$$t_E(l_E, Act_A) = gc \mapsto gc+1 \wedge q_1 \mapsto qs_5 \wedge e_{123} \mapsto \bot \wedge e_{23} \mapsto qs_7, \text{ if } e_{123} = qc_3 \wedge Act_A = m\_q_{1\_}-\alpha;$$
$$t_A(l_A, Act_A) = pc \mapsto pc+1 \wedge s_1 \mapsto 0 \wedge q_1 \mapsto 2, \text{ if } Act_A = m\_q_{1\_}+\alpha;$$
$$t_A(l_A, Act_A) = pc \mapsto pc+1 \wedge s_1 \mapsto 1 \wedge q_1 \mapsto 2, \text{ if } Act_A = m\_q_{1\_}-\alpha\,.$$

In the case that the measured qubit is in a state that coincides with one of the states of the measurement basis, there is only one possible outcome and we need to prevent reaching an impossible state. The translation of the transition function in case that a measurement outcome has zero probability requires modification of the evolution functions. We only show the case when measuring $|-\alpha\rangle$ is impossible. The evolution function of the Environment is given as:

$$t_E(l_E, Act_i) = gc \mapsto gc+1 \wedge q \mapsto qs_{+\alpha}, \text{ if } q = qc_{+\alpha} \wedge (Act_i = m\_q_{\text{-}}+\alpha \vee Act_i = m\_q_{\text{-}}-\alpha).$$

The Environment "signals" that the measurement of q in a quantum state qs_x has only one possible outcome. We introduce action env_x in Act_E and define the following protocol function: $P_E(l_E) = \{env_x\}$, if $q = qs_x$. The evolution function of agent i is then defined as:

$$t_i(l_i, Act_i, Act_E) = pc \mapsto pc+1 \wedge s' \mapsto 0 \wedge q \mapsto 2, \text{ if } Act_i = m\_q_{\text{-}}+\alpha \vee (Act_i = m\_q_{\text{-}}-\alpha \wedge Act_E = env_x),$$
$$t_i(l_i, Act_i, Act_E) = pc \mapsto pc+1 \wedge s' \mapsto 1 \wedge q \mapsto 2, \text{ if } Act_i = m\_q_{\text{-}}-\alpha \wedge Act_E \neq env_x.$$

3.3 Correctness Proof

We now show that the translation f defined in the previous section is sound, that is, f preserves the truth conditions of formulae defined in the language $\mathscr{L}$ introduced in Section 2.2 from the set of atomic propositions $AP = \{x = y, q_i = q_j\}$. In [7] the truth conditions for the atoms in AP in a configuration C of a network $\mathscr{N}$ are given as follows:

$$(C, \mathscr{N}) \models x = y \quad \text{iff} \quad \text{there are agents } i, j \text{ such that } \Gamma_i(x) = \Gamma_i(y);$$
$$(C, \mathscr{N}) \models q_i = q_j \quad \text{iff} \quad \text{the global quantum state } \sigma \text{ is such that } \sigma = q_i = q_j.$$

Intuitively, $x = y$ holds iff the bits denoted by x and y are equal. Also, $q_i = q_j$ holds iff the qubits denoted by q_i and q_j are equal. We can prove the following result on the translation f and the language $\mathscr{L}$.

Theorem 1. *For every formula $\phi \in \mathscr{L}$,*

$$(C, \mathscr{N}) \models \phi \quad \textit{iff} \quad (f(\mathscr{N}), f(C)) \models \phi$$

Proof. The proof is by induction on the length of ϕ. For reasons of space, we only provide a sketch of the proof. If ϕ is an atomic formula, then ϕ is of the form $\mathfrak{a} = \mathfrak{b}$, where $\mathfrak{a}$ and $\mathfrak{b}$ are both either bits or qubits. By the definition of $f(C)$ in Section 3.1 we can easily check that $(C, \mathscr{N}) \models \mathfrak{a} = \mathfrak{b}$ iff $(f(\mathscr{N}), f(C)) \models \mathfrak{a} = \mathfrak{b}$. Thus, the base case holds. The inductive case for propositional connectives $\neg$ and $\vee$ is straightforward.

If $\phi = EX\psi$, then by the translation of events in the event sequence $\mathscr{E}$ into actions in Act defined in Section 3.2, we can see that two configurations $C, C' \in \mathscr{N}$ are in the temporal relation induced by

$\mathscr{E}$, or $C \xrightarrow{\mathscr{N}} C'$, iff their translations $f(C), f(C') \in f(\mathscr{N})$ are in the temporal relation induced by Act, or $f(C) \xrightarrow{f(\mathscr{N})} f(C')$. The result then follows by the induction hypothesis. The inductive case for the other temporal operators is similar.

If $\phi = K_i\psi$, then by the definition of the local state l_i of an agent i in Section 3.1, we have that $l_i(f(C)) = l'_i(f(C'))$ iff $\Gamma_i = \Gamma'_i$ and $\mathscr{E}_i = \mathscr{E}'_i$, that is, $C \sim_i^{\mathscr{N}} C'$ iff $f(C) \sim_i^{f(\mathscr{N})} f(C')$. Also in this case the result follows by the induction hypothesis. This completes the sketch. □

Theorem 1 allows us to check whether a specification $\phi \in \mathscr{L}$ is satisfied in a network $\mathscr{N}$ by verifying ϕ in the corresponding interpreted system $f(\mathscr{N})$.

4 Implementation and Evaluation

In this section we present an implementation of the formal map above. DMC2ISPL[1] is a source-to-source compiler, written in C++ and using GNU Octave libraries for matrix operations. DMC2ISPL translates a protocol specified in a machine-readable DMC input format into an ISPL program. The code generated is then run by MCMAS, which in turn reports on the specification requirements of the protocol.

We modified DMC, so it can be read by the compiler. The adaptation closely follows the syntax of the original DMC, but also reflects some features of ISPL. A DMC file consists of five modules: a set of *agents*, a set of *qubits*, whose initial state is explicitly declared, a set of *groups* of agents that are used in formulae involving group modalities, a set of *formulae* to be verified, and a set of *macros* that allow agents to perform complex quantum operations in a single time step. The declaration of an agent consists of a set of input qubits, a set of *a priori* known qubits, a set of classical inputs, and a set of events the agent executes. For illustration, the DMC code snippet for QTP can be found in Listing 1.

```
1  -- AGENTS
2  Alice: {1,2},
3         {},
4         {},
5         {c!(Bob, s2), c!(Bob, s1), Me(2,0,-,-,s2), Me(1,0,-,-,s1), En(1, 2)};
6
7  Bob: {3},
8       {},
9       {},
10      {cX(3, x2), cZ(3, x1), c?(Alice, x2), c?(Alice, x1)};
11
12 -- QUBITS
13 1: ?;
14 2, 3: {(0.5, 0), (0.5, 0), (0.5, 0), (-0.5, 0)};
15
16 -- FORMULAE
17 AF {3 = init(1)};
18 !EF K (Alice, {3}) and !EF K (Bob, {3});
19 AF K(Bob, {3 = init(1)});
20 !EF K(Alice, {3 = init(1)});
```

Listing 1: QTP.dmc

DMC2ISPL has the architecture of a standard compiler. It consists of the three following components: a module for parsing and validating the DMC input file, a module for generating the reachable quantum

[1]The source code is available from http://www.doc.ic.ac.uk/ pg809/dmc2ispl.tar.gz

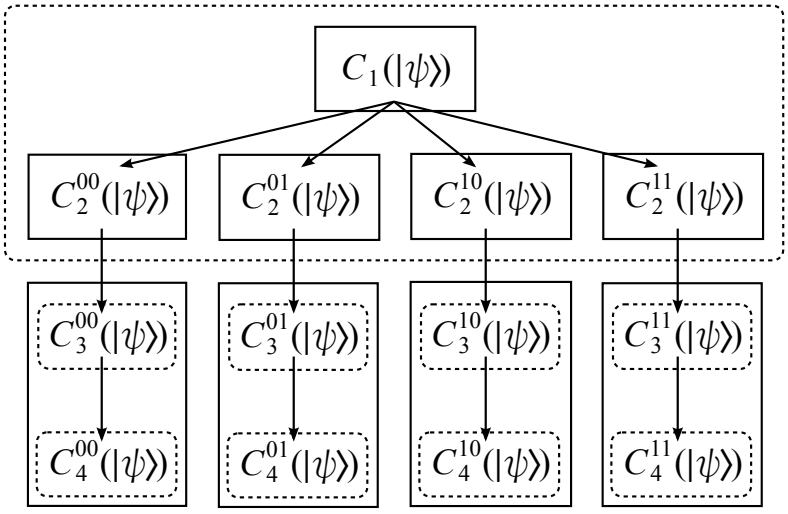


Figure 1: The epistemic accessibility relations of Alice and Bob in the QTP network.

state space, and a module for generating the ISPL output file. Essentially, since MCMAS does not support matrix arithmetic, the compiler is responsible for computation of the reachable quantum state space, enumeration of encountered quantum states, and generation of the evolution function of the global quantum system. Quantum states of a n-qubit system are represented as $2^n \times 1$ complex matrices and unitary operators and measurement projections as $2^n \times 2^n$ sparse complex matrices. After the elimination of the global phase, whenever two identical state matrices are encountered during the evolution of the quantum state of the n-qubit system, they have assigned the same name. MCMAS then works with these enumerations.

We used the compiler to verify QTP, as well as the Quantum Key Distribution (QKD) [12], and the Superdense Coding (SDC) [4] protocol against the properties from the reference papers [7, 10]. Table 3 summarises these properties. We discuss QTP in more detail.

The figure 1 gives a graphical representation of the possible configurations in the QTP network. Note that configurations are parametrised by measurement outcomes and the quantum input $|\psi\rangle$. The first formula in QTP section of Table 3 states that the $\mathscr{N}_{TP}$ network is correct, since the state of Bob's qubit q_3 will eventually be equal to the initial state of Alice's qubit q_1. The second formula states that neither agent knows the actual quantum state of the qubit q_3 at any point of the computation. The third formula states that Bob eventually knows that the state of his qubit q_3 is equal to the initial state of qubit q_1. The last formula states that Alice never knows this fact.

Interestingly, while [10] states that all four formulae are true in the model, MCMAS evaluated the last formula to false. The reason is that even though Alice cannot distinguish configuration $C_3^{00}(|\psi\rangle)$ from $C_4^{00}(|\psi\rangle)$, the atom $q_3 = init(q_1)$ holds in both configurations as Bob does not apply any correction for measurement outcomes $s_1 s_2 \mapsto 00$, and so the quantum state of the system is invariant along this path. This shows the importance of an automated algorithmic approach to verification as opposed to a hand-made inspection.

We conclude with some performance considerations. The tests were carried out on a 32-bit Fedora 12 Linux machine with a 2.26GHz Intel Core2 Duo processor and 2.9GiB RAM as follows: first, we translated the DMC specification into the corresponding ISPL code using the compiler, then we analysed the resulting code using MCMAS. Table 4 reports the results for the three protocols. It can be seen that

Protocol	Formula	Reading
QTP	$AF(q_3 = init(q_1))$	q_3 eventually equals to initial q_1
	$\neg EFK_{\mathbf{A}}(q_3 = \|\psi\rangle) \wedge \neg EFK_{\mathbf{B}}(q_3 = \|\psi\rangle)$	neither **A** nor **B** ever knows state of q_3
	$AFK_{\mathbf{B}}(q_3 = init(q_1))$	**B** eventually knows q_1 was teleported
	$\neg EFK_{\mathbf{A}}(q_3 = init(q_1))$	**A** never knows q_1 was teleported
QKD	$\alpha_A = \alpha_B \rightarrow AF(K_{\mathbf{A}}(s_1 = s_2) \wedge K_{\mathbf{B}}(s_1 = s_2))$	success if **A** & **B** used the same basis
	$\alpha_A \neq \alpha_B \rightarrow \neg EF(K_{\mathbf{A}}(s_1 = s_2) \vee K_{\mathbf{B}}(s_1 = s_2))$	failure if **A** & **B** used different bases
SDC	$AF(s_1 = y_1 \wedge s_2 = y_2)$	**B** eventually receives the inputs of **A**
	$AFK_{\mathbf{B}}(s_1 = y_1 \wedge s_2 = y_2)$	**B** eventually knows the inputs
	$\neg EFK_{\mathbf{A}}K_{\mathbf{B}}(s_1 = y_1 \wedge s_2 = y_2)$	**A** never knows the fact above

Table 3: Verified properties of QKD and SDC protocols.

Protocol	Reachable States		Memory (kB)		Time (s)	
	DMC2ISPL	MCMAS	DMC2ISPL	MCMAS	DMC2ISPL	MCMAS
QTP	40	108	7184	6068	0.015	0.066
QKD	53	348	7240	6119	0.016	0.014
SDC	4239	2192	8132	6279	0.112	0.407

Table 4: Verification results for QTP, QKD and SDC protocols.

all protocols were verified very quickly. This is due to their small state space and the limited number of entangled qubits involved.

However, the amount of required resources grows exponentially for a constant increase in the number of entangled qubits. Additionally, measuring a quantum system using many different measurement angles results in many unique quantum states, which in turn requires a large number of enumeration values and an extensive evolution function. This affects the verification of a quantum protocol by MCMAS. We analysed several experimental protocols to test the limits of the tool. The results showed that protocols with up to 10^7 reachable classical states and 20 entangled qubits can be realistically verified.

5 Conclusion

In this paper we presented a methodology for the automated verification of quantum distributed systems via model checking. We defined a translation from DMC to IS, so that MCMAS can be used to verify protocols specified in DMC, and we implemented it in a source-to-source compiler. The DMC formalism was adapted to be used as an input language for the compiler. Several quantum protocols were translated and their temporal epistemic properties were successfully checked with MCMAS. Given the universality of the underlying Measurement Calculus [8], the expressive power of DMC in terms of available quantum operations is complete. However, DMC does not support any control flow statement for the classical part of protocols. This is one of the two major limitations of the technique, although it can be solved by a suitable extension of the language. Another limitation results from the state space explosion and cannot be easily overcome since the quantum simulator requires exponential time and space on a classical computer.

References

[1] A. Baltag & S. Smets (2005): *Complete Axiomatizations for Quantum Actions*. *International Journal of Theoretical Physics* 44(12), doi:10.1007/s10773-005-8022-2.

[2] A. Baltag & S. Smets (2006): *LQP: the dynamic logic of quantum information*. Mathematical Structures in Computer Science 16(3), doi:10.1017/S0960129506005299.

[3] A. Baltag & S. Smets (2008): *A Dynamic-Logical Perspective on Quantum Behavior*. Studia Logica 89(2), doi:10.1007/s11225-008-9126-5.

[4] C. H. Bennett & S. J. Wiesner (1992): *Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states*. Physical Review Letters 69(20), doi:10.1103/PhysRevLett.69.2881.

[5] C. H. Bennett et al. (1993): *Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels*. Physical Review Letters 70(13), doi:10.1103/PhysRevLett.70.1895.

[6] E. M. Clarke, Jr., O. Grumberg & D. A. Peled (1999): *Model Checking*. MIT Press.

[7] V. Danos & E. D'Hondt (2008): *Classical Knowledge for Quantum Cryptographic Reasoning*. Electronic Notes in Theoretical Computer Science 192(3), doi:10.1016/j.entcs.2008.10.026.

[8] V. Danos, E. Kashefi & P. Panangaden (2007): *The Measurement Calculus*. J. ACM 54(2), doi:10.1145/1219092.1219096.

[9] V. Danos et al. (2007): *Distributed Measurement-based Quantum Computation*. Electronic Notes in Theoretical Computer Science 170, doi:10.1016/j.entcs.2006.12.012.

[10] E. D'Hondt & P. Panangaden (2005): *Reasoning about quantum knowledge*. In: Proceedings of FSTTCS '05, doi:10.1007/11590156_45.

[11] E. D'Hondt & M. Sadrzadeh (2011): *Classical Knowledge for Quantum Security*. Electronic Notes in Theoretical Computer Science 270(1), doi:10.1016/j.entcs.2011.01.014.

[12] A. K. Ekert (1991): *Quantum cryptography based on Bell's theorem*. Physical Review Letters 67(6), doi:10.1103/PhysRevLett.67.661.

[13] M. Elboukhari, M. Azizi & A. Azizi (2010): *Analysis of the Security of BB84 by Model Checking*. IJNSA 2(2), doi:10.5121/ijnsa.2010.2207.

[14] R. Fagin et al. (2003): *Reasoning About Knowledge*. MIT Press.

[15] S. J. Gay, R. Nagarajan & N. Papanikolaou (2008): *QMC: A Model Checker for Quantum Systems*. In: Proceedings of CAV 2008, doi:10.1007/978-3-540-70545-1_51.

[16] M. Kwiatkowska, G. Norman & D. Parker (2004): *Probabilistic symbolic model checking with PRISM: a hybrid approach*. Int. J. Softw. Tools Technol. Transf. 6(2), doi:10.1007/s10009-004-0140-2.

[17] A. Lomuscio, H. Qu & F. Raimondi (2009): *MCMAS: a model checker for the verification of multi-agent systems*. In: Proceedings of CAV '09, doi:10.1007/11691372_31.

[18] R. van der Meyden & M. Patra (2003): *Knowledge in Quantum Systems*. In: Proc. of TARK '03, doi:10.1145/846241.846257.

[19] M. A. Nielsen & I. L. Chuang (2000): *Quantum Computation and Quantum Information*. Cambridge University Press.

[20] M. Ying, Y. Li, N. Yu & Y. Feng (2010): *Model-Checking Linear-Time Properties of Quantum Systems*. `http://arxiv.org/abs/1101.0303`.

Mean field and fluid approaches to Markov chain analysis

Jeremy T. Bradley *
jb@doc.ic.ac.uk
Department of Computing, Imperial College London, UK

Representing the explicit state space of performance models has inherent difficulties. Just as the state-space explosion effects functional correctness evaluation, so it can also be easily a problem in performance models. In particular, classical Markov chain analysis of any variety requires exploration of the global state space and, even for a simple system, this quickly becomes computationally infeasible. Fluid and mean-field analysis techniques attempt to side-step the state-space explosion and provide a computationally cheap way of analysing certain features of Markov chains.

1 Introduction

In recent years, there has been a substantial interest in the *fluid* or *mean-field* approach to analysing stochastic problems in computing [1, 2, 3, 4, 5, 6, 7] and stochastic process algebra models in particular [8, 9, 10, 11, 12]. This style of analysis of the underlying Markov process offers an attractive computational alternative to traditional explicit state space analysis techniques. However, there are certain limitations which need to be kept in mind. Often fluid and mean-field analysis require the construction of an aggregate state space that means that specific model features are abstracted away. Additionally, in constructing a fluid model of a large Markov chain which incorporates synchronisation features, approximations may need to be made and it is important that a modeller is made aware of those approximations when analysing their model. If we are prepared to live with these aspects of the analysis, then the computational benefits are such that huge analysis tasks can be carried out in a relatively short period of time.

Quantitative analysis of Markov processes by means of the fluid or mean-field approach exploits substantial parallelism in the original model to construct a series of ordinary differential equations (ODEs) which capture the transient evolution of the system. Analysis of substantial Markov models using variety of fluid techniques [1, 13, 10, 8] is made possible and analyses of state spaces of 10^{100} states and beyond are not uncommon. Explicit state-space performance techniques which analyse the underlying continuous-time Markov chain directly (for example, [14, 15]) are typically limited to 10^{11} states for a steady-state style of analysis. It is perhaps, slightly unfair to make this comparison directly, as fluid and mean-field analyses do not scale with the number of global states in a model, but instead with the local state space of the constituent Markov processes.

In discrete time, mean-field techniques have been well documented by Benaïm and Le Boudec [2]. McCaig *et al.* [12] developed a discrete-time mean-field framework around the synchronous process algebra WSCCS, based on original work by Sumpter [16]. While Bakhshi *et al.* [4, 7] have developed some discrete-time model-specific analysis techniques for gossip protocols using the mean-field technique. In the continuous-time domain, Hillston developed *fluid-flow* analysis [1] to make first-order approximations of massively parallel PEPA models. Bortolussi [8] has presented a formulation for the stochastic constraint programming language, sCCP and Cardelli has a first-order fluid analysis translation to ordinary differential equations (ODEs) for the π-calculus [9].

*The author is funded in part by the EPSRC on the AMPS project (reference EP/G011737/1).

Preliminary Report. Final version to appear in:
QAPL 2012

2 Fluid passage-time analysis

The key impact of fluid and mean-field analysis techniques has been that it allows the modeller to trial many different system configurations and parameterisations easily whereas, as has been noted, a single system instance might take many hours or days to analyse using classical techniques. Latterly, fluid techniques have been used to extract useful passage-time measures [17, 18, 19] from Markov processes specified using stochastic process algebra. This allows a modeller to focus on the duration of key transactions in the higher-level system. With the rapid computation of fluid analysis, parameter sweeping can be efficiently performed to find which model variables (exponential rates or scale parameters) have greatest effect on a key passage-time measure. Often these passage times will take the form of requirements or service level objectives (SLOs) for the system, for example, 97.8% of search queries should be responded to within 0.5 seconds. Understanding how a key passage time is sensitive to changes in model parameters can help improve the design of the system. In particular how a passage time reacts to changes in the scale of component deployment within the system, so-called scalability analysis, is critical to the system engineering process.

References

[1] J. Hillston, "Fluid flow approximation of PEPA models," in *QEST'05, Proceedings of the 2nd International Conference on Quantitative Evaluation of Systems*, (Torino), pp. 33–42, IEEE Computer Society Press, September 2005.

[2] M. Benaïm and J.-Y. Le Boudec, "A class of mean field interaction models for computer and communication systems," *Performance Evaluation*, vol. 65, no. 11-12, pp. 823–838, 2008.

[3] L. Massoulié and M. Vojnovic, "Coupon replication systems," *IEEE/ACM Transactions on Networking*, vol. 16, pp. 603–616, June 2008.

[4] R. Bakhshi, L. Cloth, W. Fokkink, and B. Haverkort, "Mean-field analysis for the evaluation of gossip protocols," in *QEST'09, Proceedings of the 5th IEEE Conference on the Quantitative Evaluation of Systems*, pp. 247–256, IEEE Computer Society, September 2009.

[5] A. Ganesh, S. Lilienthal, D. Manjunath, A. Proutiere, and F. Simatos, "Load balancing via random local search in closed and open systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, p. 287, June 2010.

[6] S. Shakkottai and R. Johari, "Demand-aware content distribution on the Internet," *IEEE/ACM Transactions on Networking*, vol. 18, pp. 476–489, Apr. 2010.

[7] R. Bakhshi, L. Cloth, W. Fokkink, and B. R. Haverkort, "Mean-field framework for performance evaluation of push-pull gossip protocols," *Performance Evaluation*, vol. 68, no. 2, pp. 157–179, 2011.

[8] L. Bortolussi and A. Policriti, "Stochastic concurrent constraint programming and differential equations," in *QAPL'07, 5th Workshop on Quantitative Aspects of Programming Languages*, vol. 190 of *Electronic Notes in Theoretical Computer Science*, pp. 27–42, Elsevier, September 2007.

[9] L. Cardelli, "On process rate semantics," *Theoretical Computer Science*, vol. 391, pp. 190–215, February 2008.

[10] R. Hayden and J. T. Bradley, "A fluid analysis framework for a Markovian process algebra," *Theoretical Computer Science*, vol. 411, pp. 2260–2297, May 2010.

[11] M. Tribastone, S. T. Gilmore, and J. Hillston, "Scalable differential analysis of process algebra models," *IEEE Transactions on Software Engineering*, vol. 38, pp. 205–219, Jan–Feb 2012.

[12] C. McCaig, R. Norman, and C. Shankland, "From individuals to populations: A mean field semantics for process algebra," *Theoretical Computer Science*, vol. 412, no. 17, pp. 1557–1580, 2011.

[13] L. Cardelli, "On process rate semantics," *Theoretical Computer Science*, vol. 391, pp. 190–215, February 2008.

[14] W. J. Knottenbelt, P. G. Harrison, M. S. Mestern, and P. S. Kritzinger, "A probabilistic dynamic technique for the distributed generation of very large state spaces," *Performance Evaluation*, vol. 39, pp. 127–148, February 2000.

[15] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: Probabilistic symbolic model checker," in *TOOLS'02, Proceedings of the 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation* (A. J. Field et al., ed.), vol. 2324 of *Lecture Notes in Computer Science*, (London), pp. 200–204, Springer-Verlag, 2002.

[16] D. J. T. Sumpter and D. S. Broomhead, "Relating individual behaviour to population dynamics," *Proceedings of the Royal Society: Series B*, vol. 268, pp. 925–932, 2001.

[17] A. Clark, A. Duguid, S. T. Gilmore, and M. Tribastone, "Partial evaluation of PEPA models for fluid-flow analysis," in *Proceedings of the 5th European Performance Engineering Workshop on Computer Performance Engineering (EPEW)* (N. Thomas and C. Juiz, eds.), vol. 5261 of *Lecture Notes in Computer Science*, pp. 2–16, Springer Berlin Heidelberg, Aug. 2008.

[18] R. Hayden, A. Stefanek, and J. T. Bradley, "Fluid computation of passage time distributions in large Markov models," *Theoretical Computer Science*, vol. 413, pp. 106–141, January 2012.

[19] R. Hayden, J. T. Bradley, and A. Clark, "Performance specification and evaluation with Unified Stochastic Probes and fluid analysis," *IEEE Transactions on Software Engineering*, 2012. (In press).

Quantifying Side-Channels in RSA and AES

Boris Köpf
IMDEA Software Institute
Madrid, Spain
boris.koepf@imdea.org

Quantitative information-flow analysis (QIF) offers methods for reasoning about information-theoretic confidentiality properties of programs. The measures used by QIF are associated with operational security guarantees such as lower bounds for the effort required to determine a secret by exhaustive search. Moreover, they can be concisely expressed in terms of programming language semantics, which enables one to leverage existing program analysis techniques for their computation.

This talk reports on a line of work on techniques for the QIF analysis of cache and timing side-channels in implementations of cryptographic algorithms. Attacks exploiting these side-channels are highly effective [2, 3, 7], and most countermeasures against them are only heuristic (i.e. they defeat particular attacks, but are not backed up by a formal security guarantee). The talk will show how QIF techniques can be used for establishing upper bounds for the side-channel leakage of implementations of the RSA and AES cryptosystems, based on formal models of the underlying platforms.

For RSA, I will present work [4, 6] on the QIF analysis of input blinding, the state-of-the-art countermeasure against timing attacks. The analysis reveals that blinding offers strong guarantees whenever the range of possible timing measurements is small. Based on this insight, we propose the combination of blinding and discretization of execution times as the first countermeasure (beyond constant-time implementations) against RSA timing attacks that is backed up by a formal security guarantee. Our experiments on a 1024-bit RSA implementation demonstrate the cost-efficiency of this countermeasure.

For AES, I will report on ongoing work [5] on a method for the automatic QIF analysis of side-channels due to observable cache behavior. At the heart of this method is a novel technique for efficient counting of concretizations of abstract cache-states that enables connecting techniques for static cache-analysis and QIF. We implement this counting procedure on top of the AbsInt TimingExplorer [1], the most advanced engine for static cache-analysis and perform a study where we derive upper bounds on the cache leakage of a 128-bit AES executable. Our results demonstrate the feasibility of automating QIF analyses for cache side-channels of real systems.

References

[1] *AbsInt aiT Worst-Case Execution Time Analyzers*. http://www.absint.com/a3/.

[2] Daniel J. Bernstein (2005): *Cache-timing attacks on AES*. Technical Report.

[3] Paul Kocher: *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*. In: *Proc. CRYPTO 2006.*

[4] Boris Köpf & Markus Dürmuth: In: *Proc IEEE CSF 2009.*

[5] Boris Köpf, Laurent Mauborgne & Martin Ochoa: *Automatic Quantification of Cache Side-Channels*. Cryptology ePrint Archive, Report 2012/034. http://eprint.iacr.org/.

[6] Boris Köpf & Geoffrey Smith: In: *Proc. IEEE CSF 2010.*

[7] Dag Arne Osvik, Adi Shamir & Eran Tromer: *Cache Attacks and Countermeasures: the Case of AES*. In: *Proc. CT-RSA 2006.*

Submitted to:
QAPL 2012

A non-local method for robustness analysis of floating point programs *

Ivan Gazeau, Dale Miller and Catuscia Palamidessi
INRIA and LIX, Ecole Polytechnique

Robustness is a standard correctness property which intuitively means that if the input to the program changes less than a fixed small amount then the output changes only slightly. This notion is useful in the analysis of rounding error for floating point programs because it helps to establish bounds on output errors introduced by both measurement errors and by floating point computation. Compositional methods often do not work since key constructs—like the conditional and the while-loop—are not robust. We propose a method for proving the robustness of a while-loop. This method is non-local in the sense that instead of breaking the analysis down to single lines of code, it checks certain global properties of its structure. We show the applicability of our method on two standard algorithms: the CORDIC computation of the cosine and Dijkstra's shortest path algorithm.

Keywords: Program analysis, floating-point arithmetic, robustness to errors.

1 Introduction

Programs using floating point arithmetic are often used for critical applications and it is therefore fundamental to develop methods to establish the correctness of such programs. A central problem in dealing with floating point programs is the propagation of errors due to the digitization of analog quantities and the introduction of floating point errors during computation. As is well known, floating point arithmetic on these representations is quite different from real number arithmetic: for example, addition is neither commutative nor associative [5].

The developers of floating point programs would like to think in terms of real number semantics instead of the more ad hoc and complicated semantics given by some specific definition of floating point arithmetic, such as the IEEE standard 754 [8]. A central problem in trying to reason about floating point programs is that in dealing with non-continuous operators such as the conditional and the while-loop, floating point errors can result in what appears to be erratic behavior. The problem is that these constructs are *non-robust*: small variations in the data can cause large variations in the results.

When the program contains non-robust operators, traditional compositional methods do not work well. Decomposing the correctness of a looping program using Hoare triples, for example, usually requires either introducing abstractions (eg, approximations) which can then make conclusions too imprecise, or to undergo a very complex and intricate proof.

In this paper, we will take a different approach: we shall describe some programs where such erratic behavior is recognized and find a way to reason and bound all of that behavior. By moving away from the reasoning using Hoare's style emphasis on local and compositional analysis of a looping program, we are able to avoid reasoning about individual erratic behaviors: instead, we will treat such behaviors as an aggregate and try to bound the behavior of that aggregate.

*This work has been partially supported by the project ANR-09-BLAN-0345-02 CPP.

Preliminary Report. Final version to appear in:
QAPL 2012

To illustrate such a possibility in reasoning, consider Dijkstra's minimal path algorithm [3]. This greedy algorithm moves from a source node to its neighbors, always picking the node with the least accumulated path from the source. If one makes small changes to the distances labeling edges, then the least path distance will change also by a small amount: that is, this algorithm is continuous. However, the actual behavior of the loop and the marking of subsequent nodes can vary greatly with small changes to edge lengths. Our approach to reasoning will allow us to view all of these apparently erratic choices of intermediate paths as an aggregate on which we are able to establish the robustness of the entire algorithm.

Plan of the paper In the next section we introduce the concept of robustness and we relate it to the notions of continuity and k-Lipschitz. Section 3 contains our main contribution: a schema for reasoning about robustness in programs and its correctness. We then show the applicability of our proposal in two main examples: The CORDIC algorithm for computing cosine, presented in Section 4, and Dijkstra's shortest-path algorithm, presented in Section 5. In Section 6 we discuss some related work. Section 7 concludes and discusses some future lines of research.

2 Robustness of floating-point programs

Robustness is a standard concept from control theory [12, 11]. In the case of programming languages, there are two definitions of robustness that have been considered. One definition used by Chaudhuri et al [1] considered robustness to be based on continuity. Later Chaudhuri et al [2] considered a stronger notion of robustness, namely the k-Lipschitz property: that is, changes to the input to a program lead to only proportionally bounded changes to the output. Another approach was used by Majumdar et al in [9, 10] where robustness is formulated as "if the input of the program changes by an amount less than ε, where ε is a *fixed* constant, then the output changes only slightly." In our paper, we propose a more flexible and general notion of robustness that generalizes both of these concepts. We now motivate and explain our notion of robustness in more detail.

The notions of robustness considered in [1, 2] are mainly useful for *exact semantics*, namely when we do not take into account the errors introduced by the representation and/or the computation. In this case, the only deviation comes from the error of the input. The continuity property, that for a function f on reals is defined as:

$$\forall \varepsilon > 0\ \exists \delta\ \forall i, i' \in \mathbb{R}\ \ |i - i'| < \delta \Rightarrow |f(i) - f(i')| < \varepsilon$$

ensures that the correct output can be approximated when we can approximate the input closely enough. This notion of robustness, however, is too weak in many settings, because a small variation in the input can cause an unbounded change in the output. The k-Lipschitz property, defined as

$$\forall i, i' \in \mathbb{R}\ \ |f(i) - f(i')| \leq k|i - i'|$$

amends this problem because it bounds the variation in the output linearly by the variation in the input.

In our setting, however, the k-Lipschitz property is too strong. This is due to the following reasons

1. If we consider a *finite precision semantics*, like floating point implementations, the constant factor k can become much bigger than the one optimal for the exact semantics. For instance, assume that the available representations are the numbers in the set $\{k2^{-32} | k \in \mathbb{Z}\}$ and rounding is done by taking the lower value, and observe that a function like $f : x \mapsto 2^{-4}x$, which is 2^{-4}-Lipschitz in

the exact semantics, is only 1-Lipschitz in this approximate semantics. Indeed, there exists two values that differ by just 2^{-32} and return a result that differ by 2^{-32}. For example, take 1 and $1-2^{-32}$: we have that $f(1) = 2^{-4}$ and $f(1-2^{-32}) = 2^{-4} - 2^{-36}$, but the second result will be rounded down to $2^{-4} - 2^{-32}$.

2. There are algorithms that have a desired precision e as a parameter and are considered correct as long as the result differs by at most e from the results of the mathematical function they are meant to implement. A program of this kind may be discontinuous (and therefore not k-Lipschitz) even if it is considered to be a correct implementation of a k-Lipschitz function. The phenomenon is illustrated by the following program f which is meant to compute the inverse of a strictly increasing function $g : \mathbb{R}^+ \to \mathbb{R}^+$ whose inverse is k-Lipschitz for some k.

```
f(i){ y=0;
      while(g(y) < i){
            y= y+e; }
      return y; }
```

The program f approximates g^{-1} with precision e in the sense that

$$\forall x \in \mathbb{R}^+ \; f(x) - e \leq g^{-1}(x) \leq f(x)$$

Given the above inequality, we would like to consider the program f as robust, even though the function it computes is discontinuous (and hence not k-Lipschitz, for any k).

These two observations lead us to define another property, $P^1_{k,\varepsilon}$, to capture robustness:

$$\forall i, i' \in \mathbb{R}, |f(i) - f(i')| \leq k|i - i'| + \varepsilon$$

This property amends the two previous problems by setting ε to 2^{-32} in the first example and to e in the second example. It also extends the usual definition of the k-Lipschitz property, which can be expressed as $P^1_{k,0}$.

Now, we want to extend this definition to allow for several variables and for other metric spaces besides $\mathbb{R}$: e.g., probability distributions, intervals arithmetic etc. Thus, we consider, instead, two metric spaces: one for input (I, d_I) and the other for the return value (R,d_R). Hence, our robustness property $P^2_{k,\varepsilon}$ becomes

$$\forall i, i' \in I, d_R(f(i), f(i')) \leq k d_I(i, i') + \varepsilon$$

Finally, since we are studying small deviation, it is not useful to get this property for any i and i' in I but rather when they are close: ie, $d_I(i,i') \leq \delta$, for suitable values $\delta \in \mathbb{R}^+$. In convex spaces, this property can be easily extended to pairs of inputs having distance more than δ by using intermediate values. So, finally, in this paper we propose the property $P_{k,\varepsilon,\delta}$, described in the following definition.

Definition 2.1. *Let I and R metric spaces with distance d_I and d_R respectively, $f : I \to R$ a function, $k, \varepsilon \in \mathbb{R}^+$, and let $\delta \in \mathbb{R}^+ \cup \{+\infty\}$. We define the property $P_{k,\varepsilon,\delta}$ for the function f as follows:*

$$\forall i, i' \in I, \quad d_I(i,i') \leq \delta \implies d_R(f(i), f(i')) \leq k d_I(i,i') + \varepsilon$$

3 A schema and its correctness

The main characteristic of our schema is to subdivide the code into several parts instead of analyzing it line by line.

Our template, which we show in a moment, divides the data structures in an algorithm into two parts, called A and B. Here, A is the witness to the progress of the algorithm: in particular, the stopping condition will only depend on A (and the input). The structure B is used to accumulate results that provide the answer when the stopping condition is satisfied.

3.1 The schema structure definition

Instead of presenting a formal definition of program schema and matching of code, we illustrate these with the schema in Figure 1.

```
foo(i){
      a = a0;
      b = b0;
      while(S(i,a)){
            c = O(a,b,c,i);
            a = M(a,c);
            b = N(i,b,c);
      }
      return b; }
```

Figure 1: The main template

Here, the schema variables `a`, `b`, `c`, etc, denote tuples of program variables such that no program variable occurs twice among these schema variables. Program expressions such as

```
c = O(a,b,c,i);
```

denotes a program phrase that computes new values for the variables denoted by `c` from values of variables in the tuples `a`, `b`, `c`, and `i`. The actual computation here will be denoted by `O`. This looping program initializes the variables in `a` and `b` with the values in the tuples `a0` and `b0`, respectively. The stopping condition for the loop is given by the boolean valued expression `S(i,a)` and the result of the program is the tuple of values denoted by the variables in `b`.

We shall assume that all program variables are typed in the usual way: variables may range over the values in their associated type. Our analysis of the metric properties of a looping program will, however, consider that tuples of variables, for example, `a` and `b` in Figure 1, range over some *metric space* on the Cartesian product of the variables in the tuple.

3.2 A sufficient condition for robustness

We shall now prove that a program having the generic structure of *foo* given in Figure 1 has, under certain conditions, the property $P_{k,\varepsilon,\delta}$ for some k, ε, δ.

The aim of our method is to postpone the analysis of the exact semantics of commands as far as possible. In order to begin the analysis without specific knowledge of this semantics, we need to manipulate other programs made from the functions O, M, and N that have been identified. For example, the program *listFoo* in Figure 2 will be used to extracts the list of values of c obtained for a particular execution of *foo* with input i. The new lines added to *listFoo* will assume the usual semantics for natural numbers.

```
ListFoo(i){
      a = a0;
      b = b0;
      j = 0;
      while(! S(i,a)){
            c = O(a,b,c,i);
            j = j+1;
            l[j] = c;
            a = M(a,c);
            b = N(i,b,c); }
      return l; }
```

Figure 2: Collecting `c` values in a list

We now define two new programs. The first is the foo_b program given below: it has the same shape as *foo* but instead of setting `c` by the computation of `O(a,b,c,i)`, it sets `c` with the values of a list given in input. Naturally, the stop condition for the loop is now that all element of the list have been accessed. Note that since a was just used in the computation of O, the commands affecting a are now useless and can be removed.

```
foo_b(l,i){
//    a = a0;
      b = b0;
      for(int j = 0; j < l.length; j++ ){
            c = l[j];
//          a = M(a,c);
            b = N(i,b,c); }
      return b; }
```

We have used Java-style instructions such as $l.length$ for the length of the list l and $l[j]$ for the j^{th} element of the list l. (The // syntax is used to form a comment.) We define the new function $foo_B(i,i') = foo_b(listFoo(i),i')$. Notice that $foo_B(i,i) = foo(i)$.

The second program $foo_a(l)$ is the same program as foo_b except that a is returned instead of b. In this program, the lines where b is set are now useless.

```
foo_a(l){
      a = a0;
//    b = b0;
      for(int j = 0; j < l.length; j++ ){
            c = l[j];
            a = M(a,c);
//          b = N(i,b,c);
      }
      return a; }
```

Finally, we define $foo_A(i) = foo_a(listFoo(i))$. The two function foo_A and foo_B and relations between them will be used to indirectly analyze the program *foo*.

In what follows, we use the following conventions: the domain of the variables a, b, c, and i are A, B, C and I, respectively, and $a0$ and $b0$ are some determined constants of type A and B respectively. For every type X, the expression X^* denote the type of lists of type X.

We now introduce four conditions that need to hold to prove that the *foo* program satisfies $P_{k,\varepsilon,\delta}$ for appropriate values of k, ε, and δ. The condition $C1$ expresses the property $P_{k_{N^*},\varepsilon_{N^*},\delta}$ for the transformed program foo_B, $C2$ expresses the fact that there is a relationship between the values stored in A and the values stored in B, and $C3$ and $C4$ address the stability of the stop condition $S(i,a)$.

Condition 3.1 (C1). $\forall l \in C^*.P_{k_{N^*},\varepsilon_{N^*},\delta}(\lambda z.foo_b(l,z))$.

The next condition states that whenever two inputs i and i' are within a δ of each other then it is the case that if their images in A (under foo_a) are close, then their images in B (under foo_b) are close.

Condition 3.2 (C2).

$$\forall i_1, i \in I, d_I(i,i_1) \leq \delta \implies d_B(foo_B(i,i), foo_B(i_1,i)) \leq k_A d_A(foo_A(i_1)), foo_A(i)) + \varepsilon_2$$

The stopping condition S should satisfy the following two conditions. The first expresses that the boundary of the region $\{a \mid S(i,a)\}$ cannot vary too much.

Condition 3.3 (C3)**.**

$$\forall a \in A, \forall i, i' \in I, d_I(i,i') \leq \delta \wedge S(i',a) \implies \exists a' \in I, d_A(a,a') \leq k_s d_I(i',i) + \varepsilon_s \wedge S(i,a')$$

The following condition on S states that the diameter of the region $\{a \mid S(i,a)\}$ is as small as the desired precision.

Condition 3.4 (C4)**.** $\exists \varepsilon_t, \forall a, a' \in A, \forall i \in I, S(i,a) \wedge S(i,a') \implies d_A(a,a') \leq \varepsilon_t$

Finally, our main theorem is the following.

Theorem 3.1. *If the program foo terminates and the conditions C1, C2, C3, and C4 hold, then* $P_{k_0,\varepsilon_0,\delta}$ *holds for the function computed by foo with* $k_0 = k_{N^*} + k_A k_s$ *and* $\varepsilon_0 = \varepsilon_{N^*} + k_A(\varepsilon_s + \varepsilon_t) + \varepsilon_2$.

Proof In the proof, we will use these two observations:

1. Since $listFoo(i)$ is obtained from the computation of $foo(i)$, and since $foo_B(i,i')$ replaces the result of O by this list, if we compute $foo_B(i,i)$ we are replacing each value for c by itself. Therefore we have that $foo(i) = foo_B(i,i)$.
2. In the execution of $foo(i)$, the final value of a that satisfies the stopping condition $S(i,a)$ is $foo_A(i)$.

By the observation 1, proving the theorem is equivalent to proving

$$\forall i, i0 \in I, d_I(i,i0) \leq \delta \implies d_B(foo_B(i,i), foo_B(i0,i0)) \leq k_0 d_I(i,i0) + \varepsilon_0.$$

By condition C1, choosing $l = listFoo(i0)$, we have

$$\forall i, i0 \in I, d_I(i,i_0) \leq \delta \implies d_B(foo_b(listFoo(i0),i0), foo_b(listFoo(i0),i)) \leq k_{N^*} d_I(i,i0) + \varepsilon_{N^*}.$$

By definition of foo_B, we have

$$\forall i, i0 \in I, d_I(i,i_0) \leq \delta \implies d_B(foo_B(i0,i0), foo_B(i0,i)) \leq k_{N^*} d_I(i,i0) + \varepsilon_{N^*}. \tag{1}$$

From observation 2, $S(i0, foo_A(i0))$ holds. By condition C3 (instantiating i' with $i0$) we derive that:

$$\forall i, i0 \in I, d_I(i,i_0) \leq \delta \implies \exists a' \in A, d_A(foo_A(i0),a') \leq k_s d_I(i,i0) + \varepsilon_s \wedge S(i,a'). \tag{2}$$

Hence, by observations 2 and 1, $S(i, foo_A(i))$ also holds. From inequality (2) and condition C4, we derive

$$d_A(a', foo_A(i)) \leq \varepsilon_t. \tag{3}$$

From the last inequality and from inequality (2), we derive, using the triangle inequality

$$d_A(foo_A(i0), foo_A(i)) \leq k_s d_I(i,i0) + \varepsilon_s + \varepsilon_t. \tag{4}$$

From condition C2 and inequality (4), we have

$$\forall i, i0 \in I, d_I(i,i0) \leq \delta \implies d_B(foo_B(i0,i), foo_B(i,i)) \leq k_A(k_s d_I(i,i0) + \varepsilon_s + \varepsilon_t) + \varepsilon_2. \tag{5}$$

From inequalities (1) and (5), using the triangle inequality, we derive

$$\begin{gathered} \forall i, i0 \in I, d_I(i,i0) \leq \delta \\ \implies \\ d_B(foo_B(i,i), foo_B(i0,i0)) \leq k_{N^*} d_I(i,i0) + \varepsilon_{N^*} + k_A(k_s d_I(i,i0) + \varepsilon_s + \varepsilon_t) + \varepsilon_2. \end{gathered}$$

Finally, we define $\varepsilon_0 = \varepsilon_{N^*} + k_A(\varepsilon_s + \varepsilon_t) + \varepsilon_2$ and $k_0 = k_{N^*} + k_A k_s$. □

4 Example: the CORDIC algorithm for computing cosine

In this section we apply our method to a program implementing the CORDIC algorithm [13], and we prove that it is $P_{k,\varepsilon,\infty}$.

CORDIC (COordinate Rotation DIgital Computer) is a class of simple and efficient algorithms to compute hyperbolic and trigonometric functions using only basic arithmetic (addition, subtraction and shifts), plus table lookup. The notions behind this computing machinery were motivated by the need to calculate the trigonometric functions and their inverses in real time navigation systems. Still now-a-days, since the CORDIC algorithms require only simple integer math, CORDIC is the preferred implementation of math functions on small hand calculators.

CORDIC is a successive approximation algorithm: A sequence of successively smaller rotations based on binary decisions hone in on the value we want to find. The CORDIC version illustrated in the program below computes the cosine of any angle in $[0, \pi/2]$.

```
double cos(double beta)
{
    double x = 1, y = 0, x_new, theta = 0, sigma, e = 1E-10;
    int Pow2=1;
    while(|theta - beta| > e) {
         Pow2 *= 2;
         if(beta > theta)
           sigma=1;
         else
           sigma=-1;
         sigma=sigma/Pow2;
         fact= cos(atan(sigma)); // Value stored
         x_new = x + y*sigma;
         y = fact (y - x*sigma);
         theta += atan(sigma); // Value stored
         x = fact * x_new; }
    return x; }
```

Note that this program makes call to trigonometric functions like cosine itself. But in the actual implementation, as it is explained in the comments, these calls (that are done on values divided by successive powers of two) are stored in a database so that no computation of these functions is actually done.

4.1 Scheme instantiation

To apply our method, we have first of all to instantiate the schema variables A, B, C (cf. Section 3.1) with a suitable partition of the variables of the program. The variables I are determined: they must be instantiated with the variables which represent the input.

In this example the partition for the variables will be the following.

```
A := double theta;
B := double x,y;
C := double sigma;
I := double beta;
```

We now must define a suitable metric on the types of the variables in A and B. We choose the following:

- d_A is the usual distance on $\mathbb{R}$.
- d_B is the L_2 norm on $\mathbb{R}^2$.

Now we need to identify the stopping condition $S(i,a)$. This is given by:

```
S(beta,theta) := | theta - beta | <= e
```

Finally, we need to instantiate the functions $M(a,c)$, $N(i,b,c)$, $O(a,b,c,i)$ of the schema with suitable regions of code. We choose these as follows:

```
O(theta,<x,y>,sigma,beta) {
    Pow2 *= 2;
    if(beta > theta)
      sigma=1;
    else
      sigma=-1;
    sigma=sigma/Pow2;
    return sigma; }

M(theta,sigma) {
    theta += atan(sigma);
    return theta; }

N(beta,<x,y>,sigma) {
    fact = cos(atan(sigma));
    x_new = x + y*sigma;
    y = fact * (y - x*Pow2);
    x = fact * x_new;
    return <x,y>; }
```

Finally, we need to prove that the conditions C1, C2, C3, and C4 (cf. Section 3.2) are satisfied.

4.2 Proof of C1

C1 can be proved by classical analysis of the following program.

```
double cos(double beta, int[] listFoo)
{
    double x = 1, y = 0, x_new, theta = 0, sigma = 0,e = 1E-10;
    int Pow2=1;
    for(int j=0;j<listFoo.length;j++) {
         sigma=listFoo[j];
         fact = cos(sigma);
         x_new = x + y*sigma;
         y = fact * (y - x*sigma);
         x = fact * x_new;
    }
    return x*K;
}
```

4.3 Proof of C2

This part of the proof is rather technical. The interested reader can find it in the appendix. The proof of C2 is the most difficult part of this example. We have proved it "by hand", and we do not claim that there is an easy way to automate it. However, this proof points out that we can prove the intended property without considering the whole semantics of the program, but just the relevant properties.

4.4 Proof of C3

Once we instantiate $S(i,a)$, C3 is given by the condition:

$$\forall a \in A, \forall i, i' \in I, |i-a| \leq e, \exists a' \in I, |a-a'| \leq k_s|i-i'| + \varepsilon_s \wedge |i'-a'| \leq e$$

We can satisfy this property by setting $a' = a + i' - i$, $k_s = 1$, and $\varepsilon_s = 0$.

4.5 Proof of C4

C4 can be rewritten, once we instantiate $S(i,a)$ to

$$\exists \varepsilon_t, \forall a, a' \in A, \forall i \in I, |i-a| \leq e \wedge |i-a'| \leq e \implies |a-a'| \leq \varepsilon_t$$

Which is true for $\varepsilon_t = 2e$.

5 Example: Dijkstra's shortest path algorithm

In this section we apply our method to Dijkstra's shortest path algorithm. This is an algorithm that, given a graph, computes the shortest path between a source and any vertex of the graph. We will prove, by instantiating our schema, that the following program implementing the Dijkstra's algorithm can be proved $P_{1,0,0}$ in the semantic of real numbers using our theorem.

In the following program we use some conventions: the number of vertices is fixed to w, all vertices are connected, and the maximum value for a path 999 (some stand-in of infinity).

```
int[] dijkstra( int graph[w][w]){
     int pathestimate[w],mark[w];
     int source,i,j,u,predecessor[w],count=0;
     int minimum(int a[],int m[],int k);

     for(j=1;j<=w;j++){
          mark[j]=0;
          pathestimate[j]=999;
          predecessor[j]=0;
     }
     source=0;
     pathestimate[source]=0;
     while(count<w){
          u=minimum(pathestimate,mark,w);
          mark[u]=1;
          count=count+1;
          for(i=1;i<=w;i++){
               if(pathestimate[i]>pathestimate[u]+graph[u][i]){
                    pathestimate[i]=pathestimate[u]+graph[u][i];
                    predecessor[i]=u;
               }
          }
     }
     return pathestimate;
}

int minimum(int a[],int m[],int k){
     int mi=999;
     int i,t;
     for(i=1;i<=k;i++){
          if(m[i]!=1){
               if(mi>=a[i]){
                    mi=a[i];
                    t=i;
               }
          }
     }
     return t;
}
```

5.1 Scheme instantiation

To apply our theorem, we have to instantiate the scheme variables A, B, C with some variables of the program. The variables of I are instantiated with the variables that represent the input. We chose the following instantiation: A contains the variables *count* and *mark*, B the array of double *pathestimate* and C the variable u which identify the current vertex to propagate.

```
A := int count;int mark[w];
B := pathestimate[w];
C := int u;
I := graph[w][w];
```

We now have to choose a suitable metric on the types of the variables, and we choose the following: d_I is the L_1 norm on an array of real numbers, d_B is the L_∞ norm on array of real numbers and d_A is the identity metric: that is, the distance between two elements of A is 0 if they are the same elements and it is ∞ otherwise.

Next, we identify the stopping condition:

```
S(graph,<count,mark>) := count >= w
```

Finally, we identify the functions $M(a,c)$, $N(i,b,c)$, $O(a,b,c,i)$ with the following regions of code:

```
O (count, mark, pathestimate, u, graph) {
   u=minimum(pathestimate,mark,w);
   int minimum(int a[],int m[],int k){
        int mi=999;
        int i,t;
        for(i=1;i<=k;i++){
             if(m[i]!=1){
                  if(mi>=a[i]){
                       mi=a[i];
                       t=i;
                  }
             }
        }
       return t;
   }
   return u;
}

M (<mark, count>, u) {
   mark[u]=1;
   count=count+1;
   return <mark,count>;
}

N (graph, pathestimate, u) {
   for(i=1;i<=w;i++){
        if(pathestimate[i]>pathestimate[u]+graph[u][i]){
             pathestimate[i]=pathestimate[u]+graph[u][i];
        }
   }
   return pathestimate;
}
```

We now have to prove that the conditions C1, C2, C3 and C4 hold for the given instantiations.

5.2 Proof of C1

For all $i0 \in I$, $foo_a(i0,i)$ is k-Lipschitz and k does not depend on $i0$. The proof proceeds by a standard analysis of the following program.

```
int[] dijkstra( int graph[w][w], int[] listFoo)
{
    int pathestimate[w],mark[w];
    int source,i,j,u,predecessor[w],count=0;
    int minimum(int a[],int m[],int k);
    for(j=1;j<=w;j++){
        mark[j]=0;
        pathestimate[j]=999;
        predecessor[j]=0;
    }
    source=0;
    pathestimate[source]=0;
    for(j=0;j<listFoo.length;j++){
        u=listFoo[j];
        for(i=1;i<=w;i++) {
            if(pathestimate[i]>pathestimate[u]+graph[u][i]){
                pathestimate[i]=pathestimate[u]+graph[u][i];
                predecessor[i]=u;
            }
        }
    }
    return pathestimate;
}
```

In an exact semantics (with real numbers), this program is 1-Lipschitz as any element of *pathestimate* is the sum of some element of *graph*. If the analysis is done with an exact semantics (with real numbers), we are able to prove that this program is 1-Lipschitz.

5.3 Proof of C2

The proof for C2 is rather technical. The basic idea is however quite simple. Indeed, the A structure is a set in a discrete space on which elements are added. So we prove that whatever the order of the element is B is constant. This is done by showing that local transpositions do not change the result. So the principle should apply in other algorithms with the same A structure. The complete proof can be found in the appendix.

5.4 Proof of C3

We have to prove

$$\forall a \in A, \forall i, i' \in I, S(i,a), \exists a' \in I, d_A(a,a') \leq k_s d_I(i,i') + \varepsilon_s \wedge S(i',a')$$

Since the stopping condition $S(i,a)$ does not depend on i in this case, we take $a' = a$. Thus, we can take $k_s = 1$ and $\varepsilon_s = 0$. □

5.5 Proof of C4

We have to prove

$$\exists \varepsilon_t \in \mathbb{R}, \forall a, a' \in A, \forall i \in I, S(i,a) \wedge S(i,a') \implies d_A(a,a') \leq \varepsilon_t.$$

Since $\{a | S(i,a)\}$ is a singleton for every i, the property holds for $\varepsilon_t = 0$. □

6 Related Work

Static analysis via abstract interpretation can be an effective method for deriving precise bounds on deviations [6, 7]. Since such static analysis is generally limited to analyzing code line-by-line, significant over approximations might be necessary. For example, when encountering an "if" instruction (or a looping construct), a static analyzer will have to assume that either the control flow is not perturbed by the finite-precision errors (often unrealistic) or the results from the two branches of the conditional must be merged (often causing significant over-approximation). In our examples here, control flow can be perturbed a great deal by precision errors and merging both branches is not a solution as the program is not locally continuous. Our method is useful for solving this problem since it avoids narrowly analyzing the semantics of the conditional.

In the two papers [2, 1], robustness analysis is done for the Dijkstra's algorithm. The authors split their analysis into two parts: first they prove the continuity of the algorithm and second they prove it is piecewise robust. The problem of discontinuity that can occur at some point of the execution is solved through an abstract language syntax for loops. Like in our theorem, this syntax need additional conditions (mainly the commutativity for two observable equivalent commands). However, their abstract language is more specific than our theorem: CORDIC is not in the scope of these papers which also means their conditions are simpler and their proofs are more directed than ours. The other distinction is in the semantics of the language. Their paper aims at furnishing the whole semantics which is an exact one and computational errors are treated qualitatively with the argument that a robust program is not sensitive to small variations. With our analysis, we give a quantitative definition of what small enough means. The last difference is our design for analyzing non-local-robustness. We prefer to consider non-local behaviors as happening and solving them by a program transformation using pattern than to rewrite the program in a syntax that hide the non-local behavior.

7 Future work and conclusion

We have presented a theorem that allow us to prove the robustness of some floating point programs. This theorem is abstract enough to be applicable in a number of rather different programs: here, we illustrate its use with programs to compute cosine using the CORDIC method and to compute the shortest path in a graph.

For future work, we would like to address a key possible weakness of our method: it is currently tied to a particular template. Although that template is presented abstractly, there should certainly be ways to improve the generality beyond the matching of a template. Also, since the property $P_{k,\varepsilon,\delta}$ (Definition 2.1) is more general than both k-Lipschitz and the other definitions of robustness [9, 10], we would like to explore applications of this property to cases where neither of the other definitions work.

Condition C2 is, at least in the examples considered in this paper, the most difficult condition to verify. This suggests that we might consider more restrictive conditions that would entail C2.

Acknowledgments: We would like to thank Eric Goubault and Jean Goubault-Larrecq for many useful discussions on the topic of this paper.

References

[1] Swarat Chaudhuri, Sumit Gulwani & Roberto Lublinerman (2010): *Continuity analysis of programs*. In Manuel V. Hermenegildo & Jens Palsberg, editors: *POPL*, ACM, pp. 57–70. Available at http://doi.acm.org/10.1145/1706299.1706308.

[2] Swarat Chaudhuri, Sumit Gulwani, Roberto Lublinerman & Sara NavidPour (2011): *Proving programs robust*. In Tibor Gyimóthy & Andreas Zeller, editors: *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13rd European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*, ACM, pp. 102–112. Available at http://doi.acm.org/10.1145/2025113.2025131.

[3] E. W. Dijkstra (1959): *A Note on Two Problems in Connexion with Graphs*. *Numer. Math.* 1, pp. 269–271.

[4] Ivan Gazeau, Dale Miller & Catuscia Palamidessi (2012): *A non-local method for robustness analysis of floating point programs*. Technical Report, INRIA, Tallinn, Estonie. Available at http://hal.inria.fr/hal-00665995.

[5] D. Goldberg (1991): *What every computer scientist should know about floating-point arithmetic*. *ACM Computing Surveys* 23(1), pp. 5–47.

[6] Eric Goubault (2001): *Static Analyses of the Precision of Floating-Point Operations*. In Patrick Cousot, editor: *Static Analysis, 8th International Symposiumsium, Lecture Notes in Computer Science* 2126, Springer Verlag, pp. 234–259.

[7] Eric Goubault & Sylvie Putot (2011): *Static Analysis of Finite Precision Computations*. In Ranjit Jhala & David A. Schmidt, editors: *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings, Lecture Notes in Computer Science* 6538, Springer, pp. 232–247. Available at http://dx.doi.org/10.1007/978-3-642-18275-4.

[8] IEEE Task P754 (2008): *IEEE 754-2008, Standard for Floating-Point Arithmetic*. IEEE, pub-IEEE-STD:adr, doi:http://dx.doi.org/10.1109/IEEESTD.2008.4610935. Available at http://en.wikipedia.org/wiki/IEEE_754-2008;http://ieeexplore.ieee.org/servlet/opac?punumber=4610933.

[9] Rupak Majumdar & Indranil Saha (2009): *Symbolic Robustness Analysis*. In Theodore P. Baker, editor: *IEEE Real-Time Systems Symposium*, IEEE Computer Society, pp. 355–363. Available at http://doi.ieeecomputersociety.org/10.1109/RTSS.2009.17.

[10] Rupak Majumdar, Indranil Saha & Zilong Wang (2010): *Systematic testing for control applications*. In: *MEMOCODE*, pp. 1–10.

[11] *The Parsec benchmark suite*. Available at http://parsec.cs.princeton.edu/.

[12] Stefan Pettersson & Bengt Lennartson (1996): *Stability And Robustness For Hybrid Systems*. In: *Proceedings of the 35th edition of Decision and Control*, pp. 1202–1207.

[13] Jack E. Volder (1959): *The CORDIC Trigonometric Computing Technique*. *IRE Transactions on Electronic Computers* EC-8, pp. 330–334.

A Appendix

Please consult the technical report version of this paper [4].

Quantitative Information Flow as Safety and Liveness Hyperproperties*

Hirotoshi Yasuoka
Tohoku University
Sendai, Japan
yasuoka@kb.ecei.tohoku.ac.jp

Tachio Terauchi
Nagoya University
Nagoya, Japan
terauchi@is.nagoya-u.ac.jp

We employ Clarkson and Schneider's "hyperproperties" to classify various verification problems of quantitative information flow. The results of this paper unify and extend the previous results on the hardness of checking and inferring quantitative information flow. In particular, we identify a subclass of liveness hyperproperties, which we call "k-observable hyperproperties", that can be checked relative to a reachability oracle via self composition.

1 Introduction

We consider programs containing high security inputs and low security outputs. Informally, the quantitative information flow problem concerns the amount of information that an attacker can learn about the high security input by executing the program and observing the low security output. The problem is motivated by applications in information security. We refer to the classic by Denning [14] for an overview.

In essence, quantitative information flow measures *how* secure, or insecure, a program (or a part of a program –e.g., a variable–) is. Thus, unlike non-interference [12, 16], that only tells whether a program is completely secure or not completely secure, a definition of quantitative information flow must be able to distinguish two programs that are both interfering but have different levels of security.

For example, consider the programs $M_1 \equiv \texttt{if}\ H = g\ \texttt{then}\ O := 0\ \texttt{else}\ O := 1$ and $M_2 \equiv O := H$. In both programs, H is a high security input and O is a low security output. Viewing H as a password, M_1 is a prototypical login program that checks if the guess g matches the password. By executing M_1, an attacker only learns whether H is equal to g, whereas she would be able to learn the entire content of H by executing M_2. Hence, a reasonable definition of quantitative information flow should assign a higher quantity to M_2 than to M_1, whereas non-interference would merely say that M_1 and M_2 are both interfering, assuming that there are more than one possible value of H.

Researchers have attempted to formalize the definition of quantitative information flow by appealing to information theory. This has resulted in definitions based on the Shannon entropy [14, 9, 22], the min entropy [29], and the guessing entropy [18, 4]. All of these definitions map a program (or a part of a program) onto a non-negative real number, that is, they define a function $\mathscr{X}$ such that given a program M, $\mathscr{X}(M)$ is a non-negative real number. (Concretely, $\mathscr{X}$ is $SE[\mu]$ for the Shannon-entropy-based definition with the distribution μ, $ME[\mu]$ for the min-entropy-based definition with the distribution μ, and $GE[\mu]$ for the guessing-entropy-based definition with the distribution μ.)

In a previous work [33, 32], we have proved a number of hardness results on checking and inferring quantitative information flow (QIF) according to these definitions. A key concept used to connect the hardness results to QIF verification problems was the notion of k-safety, which is an instance in a

*This work was supported by MEXT KAKENHI 23700026, 22300005, 23220001, and Global COE Program "CERIES."

Preliminary Report. Final version to appear in:
QAPL 2012

	$SE[U]$	$ME[U]$	$GE[U]$
LBP	Liveness	Liveness	Liveness
UBP	Safety	Safety	Safety
LBP constant bound	Liveness	k-observable	k-observable
UBP constant bound	Safety	k-safety [32]	k-safety [32]

Table 1: A summary of hyperproperty classifications

collection of the class of program properties called *hyperproperties* [11]. In this paper, we make the connection explicit by providing a fine-grained classification of QIF problems, utilizing the full range of hyperproperties. This has a number of benefits, summarized below.

1.) A unified view on the hardness results of QIF problems.

2.) New insights into hyperproperties themselves.

3.) A straightforward derivation of some complexity theoretic results.

Regarding 1.), we focus on two types of QIF problems, an upper-bounding problem that checks if QIF of a program is bounded above by the given number, and a lower-bounding problem that checks if QIF is bounded below by the given number. Then, for each QIF definitions *SE*, *GE*, *ME*, we classify whether or not they are safety hyperproperty, k-safety hyperproperty, liveness hyperproperty, or k-observable hyperproperty (and give a bound on k for k-safe/k-observable). Safety hyperproperty, k-safety hyperproperty, liveness hyperproperty, and observable hyperproperty are classes of hyperproperties defined by Clarkson and Schneider [11]. In this paper, we identify new classes of hyperproperties, k-observable hyperproperty, that is useful for classifying QIF problems. k-observable hyperproperty is a subclass of observable hyperproperties, and observable hyperproperty is a subclass of liveness hyperproperties.[1] We focus on the case the input distribution is uniform, that is, $\mu = U$, as showing the hardness for a specific case amounts to showing the hardness for the general case. Also, checking and inferring QIF under the uniformly distributed inputs has received much attention [17, 4, 19, 8, 22, 9], and so, the hardness for the uniform case is itself of research interest.[2]

Regarding 2.), we show that the k-observable subset of the observable hyperproperties is amenable to verification via self composition [5, 13, 30, 26, 31], much like k-safety hyperproperties, and identify which QIF problems belong to that family. We also show that the hardest of the QIF problems (but nevertheless one of the most popular) can only be classified as a general liveness hyperproperty, suggesting that liveness hyperproperty is a quite permissive class of hyperproperties.

Regarding 3.), we show that many complexity theoretic results for QIF problems of loop-free boolean programs can be derived from their hyperproperties classifications [33, 32]. We also prove new complexity theoretic results, including the (implicit state) complexity results for loop-ful boolean programs, complementing the recently proved explicit state complexity results [7].

Table 1 and Table 2 summarize the hyperproperties classifications and computational complexities of upper/lower-bounding problems. We abbreviate lower-bounding problem, upper-bounding problem, and boolean programs to LBP, UBP, and BP, respectively. The "constant bound" rows correspond to bounding problems with a constant bound (whereas the plain bounding problems take the bound as an input).

The proofs omitted from the paper appear in the extended report [35].

[1] Technically, only non-empty observable hyperproperties are liveness hyperproperties.

[2] In fact, computing QIF under other input distributions can sometimes be reduced to this case [3]. See also Section 5.3.

	$SE[U]$	$ME[U]$	$GE[U]$
LBP for BP	*PSPACE*-hard	*PSPACE*-complete	*PSPACE*-complete
UBP for BP	*PSPACE*-hard	*PSPACE*-complete	*PSPACE*-complete
LBP for loop-free BP	*PP*-hard	*PP*-hard	*PP*-hard
UBP for loop-free BP	*PP*-hard [32]	*PP*-hard [32]	*PP*-hard [32]
LBP for loop-free BP, constant bound	Unknown	*NP*-complete	*NP*-complete
UBP for loop-free BP, constant bound	Unknown	*coNP*-complete	*coNP*-complete

Table 2: A summary of computational complexities

2 Preliminaries

2.1 Quantitative Information Flow

We introduce the information theoretic definitions of QIF that have been proposed in literature. First, we review the notion of the *Shannon entropy* [28], $\mathscr{H}[\mu](X)$, which is the average of the information content, and intuitively, denotes the uncertainty of the random variable X. And, we review the notion of the *conditional entropy*, $\mathscr{H}[\mu](Y|Z)$, which denotes the uncertainty of Y after knowing Z.

Definition 2.1 (Shannon Entropy and Conditional Entropy) *Let X be a random variable with sample space $\mathbb{X}$ and μ be a probability distribution associated with X (we write μ explicitly for clarity). The Shannon entropy of X is defined as*

$$\mathscr{H}[\mu](X) = \sum_{x \in \mathbb{X}} \mu(X = x) \log \frac{1}{\mu(X = x)}$$

Let Y and Z be random variables with sample space Y and Z, respectively, and μ' be a probability distribution associated with Y and Z. Then, the conditional entropy of Y given Z is defined as

$$\mathscr{H}[\mu](Y|Z) = \sum_{z \in \mathbb{Z}} \mu(Z = z) \mathscr{H}[\mu](Y|Z = z)$$

where

$$\mathscr{H}[\mu](Y|Z = z) = \textstyle\sum_{y \in \mathbb{Y}} \mu(Y = y|Z = Z) \log \frac{1}{\mu(Y=y|Z=z)}$$
$$\mu(Y = y|Z = z) = \frac{\mu(Y=y,Z=z)}{\mu(Z=z)}$$

(The logarithm is in base 2.)

Let M be a program that takes a high security input H, and gives the low security output trace O. For simplicity, we restrict to programs with just one variable of each kind, but it is trivial to extend the formalism to multiple variables (e.g., by letting the variables range over tuples or lists). Also, for the purpose of the paper, unobservable (i.e., high security) output traces are irrelevant, and so we assume that the only program output is the low security output trace. Let μ be a probability distribution over the values of H. Then, the semantics of M can be defined by the following probability equation. (We restrict to deterministic programs in this paper.)

$$\mu(O = o) = \sum_{\substack{h \in \mathbb{H} \\ M(h) = o}} \mu(H = h)$$

Here, $M(h)$ denotes the infinite low security output trace of the program M given a input h, and $M(h) = o$ denotes the output trace of M given h that is equivalent to o. In this paper, we adopt the termination-insensitive security observation model, and let the outputs o and o' be equivalent iff $\forall i \in \omega.o_i = \bot \vee o'_i = \bot \vee o_i = o'_i$ where o and o_i denotes the ith element of o, and $\bot$ is the special symbol denoting termination.[3]

In this paper, programs are represented by sets of traces, and traces are represented by lists of stores of programs. More formally,

$$M(h) \text{ is equal to } o \quad \text{iff} \quad \sigma_0;\sigma_1;\ldots;\sigma_i;\ldots \in M$$
$$\text{where } \sigma_0(H) = h \text{ and } \forall i \geq 1.\sigma_i(O) = o_i \ (o_i \text{ denotes the ith element of } o)$$

Here, σ denotes a store that maps variables to values. Because we restrict all programs to deterministic programs, every program M satisfies the following condition: For any trace $\overrightarrow{\sigma}, \overrightarrow{\sigma}' \in M$, we have $\sigma_0(H) = \sigma'_0(H) \Rightarrow \overrightarrow{\sigma} = \overrightarrow{\sigma}'$ where σ_0 and σ'_0 denote the first elements of $\overrightarrow{\sigma}$ and $\overrightarrow{\sigma}'$, respectively. Now, we are ready to define Shannon-entropy-based quantitative information flow.

Definition 2.2 (Shannon-Entropy-based QIF [14, 9, 22]) *Let M be a program with a high security input H, and a low security output trace O. Let μ be a distribution over H. Then, the Shannon-entropy-based quantitative information flow is defined*

$$SE[\mu](M) = \mathscr{H}[\mu](H) - \mathscr{H}[\mu](H|O)$$

Intuitively, $\mathscr{H}[\mu](H)$ denotes the initial uncertainty and $\mathscr{H}[\mu](H|O)$ denotes the remaining uncertainty after knowing the low security output trace. (For space, the paper focuses on the low-security-input free definitions of QIF.)

As an example, consider the programs M_1 and M_2 from Section 1. For concreteness, assume that g is the value 01 and H ranges over the space $\{00,01,10,11\}$. Let U be the uniform distribution over $\{00,01,10,11\}$, that is, $U(h) = 1/4$ for all $h \in \{00,01,10,11\}$. The results are as follows.

$$SE[U](M_1) = \mathscr{H}[U](H) - \mathscr{H}[U](H|O) = \log 4 - \tfrac{3}{4}\log 3 \approx .81128$$

$$SE[U](M_2) = \mathscr{H}[U](H) - \mathscr{H}[U](H|O) = \log 4 - \log 1 = 2$$

Consequently, we have that $SE[U](M_1) \leq SE[U](M_2)$, but $SE[U](M_2) \not\leq SE[U](M_1)$. That is, M_1 is more secure than M_2 (according to the Shannon-entropy based definition with uniformly distributed inputs), which agrees with our intuition.

Next, we introduce the *min entropy*, which has recently been suggested as an alternative measure for quantitative information flow [29].

Definition 2.3 (Min Entropy) *Let X and Y be random variables, and μ be an associated probability distribution. Then, the min entropy of X is defined*

$$\mathscr{H}_\infty[\mu](X) = \log \frac{1}{\mathscr{V}[\mu](X)}$$

and the conditional min entropy of X given Y is defined

$$\mathscr{H}_\infty[\mu](X|Y) = \log \frac{1}{\mathscr{V}[\mu](X|Y)}$$

[3] Here, we adopt the trace based QIF formalization of [23].

where

$$\begin{array}{rcl} \mathscr{V}[\mu](X) & = & \max_{x\in\mathbb{X}} \mu(X=x) \\ \mathscr{V}[\mu](X|Y=y) & = & \max_{x\in\mathbb{X}} \mu(X=x|Y=y) \\ \mathscr{V}[\mu](X|Y) & = & \sum_{y\in\mathbb{Y}} \mu(Y=y)\mathscr{V}[\mu](X|Y=y) \end{array}$$

Intuitively, $\mathscr{V}[\mu](X)$ represents the highest probability that an attacker guesses X in a single try. We now define the min-entropy-based definition of QIF.

Definition 2.4 (Min-Entropy-based QIF [29]) *Let M be a program with a high security input H, and a low security output trace O. Let μ be a distribution over H. Then, the min-entropy-based quantitative information flow is defined*

$$ME[\mu](M) = \mathscr{H}_\infty[\mu](H) - \mathscr{H}_\infty[\mu](H|O)$$

Computing the min-entropy based quantitative information flow for our running example programs M_1 and M_2 from Section 1 with the uniform distribution, we obtain,

$$ME[U](M_1) = \mathscr{H}_\infty[U](H) - \mathscr{H}_\infty[U](H|O) = \log 4 - \log 2 = 1$$

$$ME[U](M_2) = \mathscr{H}_\infty[U](H) - \mathscr{H}_\infty[U](H|O) = \log 4 - \log 1 = 2$$

Again, we have that $ME[U](M_1) \leq ME[U](M_2)$ and $ME[U](M_2) \not\leq ME[U](M_1)$, and so M_2 is deemed less secure than M_1.

The third definition of quantitative information flow treated in this paper is the one based on the guessing entropy [24], that has also recently been proposed in literature [18, 4].

Definition 2.5 (Guessing Entropy) *Let X and Y be random variables, and μ be an associated probability distribution. Then, the guessing entropy of X is defined*

$$\mathscr{G}[\mu](X) = \sum_{1\leq i\leq m} i \times \mu(X=x_i)$$

where $\{x_1, x_2, \ldots, x_m\} = \mathbb{X}$ and $\forall i, j. i \leq j \Rightarrow \mu(X=x_i) \geq \mu(X=x_j)$.

The conditional guessing entropy of X given Y is defined

$$\mathscr{G}[\mu](X|Y) = \sum_{y\in\mathbb{Y}} \mu(Y=y) \sum_{1\leq i\leq m} i \times \mu(X=x_i|Y=y)$$

where $\{x_1, x_2, \ldots, x_m\} = \mathbb{X}$ and $\forall i, j. i \leq j \Rightarrow \mu(X=x_i|Y=y) \geq \mu(X=x_j|Y=y)$.

Intuitively, $\mathscr{G}[\mu](X)$ represents the average number of times required for the attacker to guess the value of X. We now define the guessing-entropy-based quantitative information flow.

Definition 2.6 (Guessing-Entropy-based QIF [18, 4]) *Let M be a program with a high security input H, and a low security output trace O. Let μ be a distribution over H. Then, the guessing-entropy-based quantitative information flow is defined*

$$GE[\mu](M) = \mathscr{G}[\mu](H) - \mathscr{G}[\mu](H|O)$$

We test GE on the running example from Section 1 by calculating the quantities for the programs M_1 and M_2 with the uniform distribution.

$$GE[U](M_1) = \mathscr{G}[U](H) - \mathscr{G}[U](H|O) = \tfrac{5}{2} - \tfrac{7}{4} = 0.75$$

$$GE[U](M_2) = \mathscr{G}[U](H) - \mathscr{G}[U](H|O) = \tfrac{5}{2} - 1 = 1.5$$

Therefore, we again have that $GE[U](M_1) \leq GE[U](M_2)$ and $GE[U](M_2) \not\leq GE[U](M_1)$, and so M_2 is considered less secure than M_1, even with the guessing-entropy based definition with the uniform distribution.

2.2 Bounding Problems

We introduce the bounding problems of quantitative information flow that we classify. First, we define the QIF upper-bounding problems. Upper-bounding problems are defined as follows: Given a program M and a rational number q, decide if the information flow of M is less than or equal to q.

$$\mathscr{U}_{SE} = \{(M,q) \mid SE[U](M) \leq q\}$$
$$\mathscr{U}_{ME} = \{(M,q) \mid ME[U](M) \leq q\}$$
$$\mathscr{U}_{GE} = \{(M,q) \mid GE[U](M) \leq q\}$$

Recall that U denotes the uniform distribution.

Next, we define lower-bounding problems. Lower-bounding problems are defined as follows: Given a program M and a rational number q, decide if the information flow of M is greater than q.

$$\mathscr{L}_{SE} = \{(M,q) \mid SE[U](M) > q\}$$
$$\mathscr{L}_{ME} = \{(M,q) \mid ME[U](M) > q\}$$
$$\mathscr{L}_{GE} = \{(M,q) \mid GE[U](M) > q\}$$

2.3 Non Interference

We recall the notion of non-interference, which, intuitively, says that the program leaks no information.

Definition 2.7 (Non-intereference [12, 16]) *A program M is said to be non-interfering iff for any $h,h' \in \mathbb{H}$, $M(h) = M(h')$.*

Non-interference is known to be a special case of bounding problems that tests against 0.

Theorem 2.8 ([8, 32]) *1.) M is non-interfering iff $(M,0) \in \mathscr{U}_{SE}$. 2.) M is non-interfering iff $(M,0) \in \mathscr{U}_{ME}$. 3.) M is non-interfering iff $(M,0) \in \mathscr{U}_{GE}$.*

3 Liveness Hyperproperties

Clarkson and Schneider have proposed the notion of hyperproperties [11].

Definition 3.1 (Hyperproperties [11]) *We say that P is a hyperproperty if $P \subseteq \mathscr{P}(\Psi_{\mathtt{inf}})$ where $\Psi_{\mathtt{inf}}$ is the set of all infinite traces, and $\mathscr{P}(X)$ denote the powerset of X.*

Note that hyperproperties are sets of trace sets. As such, they are more suitable for classifying information properties than the classical trace properties which are sets of traces. For example, non-interference is not a trace property but a hyperproperty.

Clarkson and Schneider have identified a subclass of hyperproperties, called liveness hyperproperties, as a generalization of liveness properties. Intuitively, a liveness hyperproperty is a property that can not be refuted by a finite set of finite traces. That is, if P is a liveness hyperproperty, then for any finite set of finite traces T, there exists a set of traces that contains T and satisfies P. Formally, let *Obs* be the set of finite sets of finite traces, and *Prop* be the set of sets of infinite traces (i.e., hyperproperties), that is,

$$\begin{array}{rcl} \mathit{Obs} & = & \mathscr{P}^{\mathtt{fin}}(\Psi_{\mathtt{fin}}) \\ \mathit{Prop} & = & \mathscr{P}(\Psi_{\mathtt{inf}}) \end{array}$$

(Here, $\mathscr{P}^{\mathtt{fin}}(X)$ denotes the finite subsets of X, $\Psi_{\mathtt{fin}}$ denotes the set of finite traces.) Let $\leq$ be the relation over $\mathit{Obs} \times \mathit{Prop}$ such that

$$S \leq T \quad \text{iff} \quad \forall t \in S. \exists t'. t \circ t' \in T$$

where $t \circ t'$ is the sequential composition of t and t'. Then,

Definition 3.2 (Liveness Hyperproperties [11]) *We say that a hyperproperty P is a liveness hyperproperty if for any set of traces $S \in Obs$, there exists a set of traces $S' \in Prop$ such that $S \leq S'$ and $S' \in P$.*

Now, we state the first main result of the paper: the lower-bounding problems are liveness hyperproperties.[4]

Theorem 3.3 *$\mathscr{L}_{SE}$, $\mathscr{L}_{ME}$, and $\mathscr{L}_{GE}$ are liveness hyperproperties.*[5]

The proof follows from the fact that, for any program M, there exists a program M' containing all the observations of M and has an arbitrary large information flow.[6]

We show that the upper-bounding problem for Shannon-entropy based quantitative information flow is also a liveness hyperproperty.

Theorem 3.4 *$\mathscr{U}_{SE}$ is a liveness hyperproperty.*

The theorem follows from the fact that we can lower the amount of the information flow by adding traces that have the same output trace. Therefore, for any program M, there exists M' having more observation than M such that $SE[U](M') \leq q$.

3.1 Observable Hyperproperties

Clarkson and Schneider [11] have identified a class of hyperproperties, called *observable hyperproperties*, to generalize the notion of observable properties [2] to sets of traces.[7]

Definition 3.5 (Observable Hyperproperties [11]) *We say that P is a observable hyperproperty if for any set of traces $S \in P$, there exists a set of traces $T \in Obs$ such that $T \leq S$, and for any set of traces $S' \in Prop$, $T \leq S' \Rightarrow S' \in P$.*

We call T in the above definition an *evidence*.

Intuitively, observable hyperproperty is a property that can be verified by observing a finite set of finite traces. We prove a relationship between observable hyperproperties and liveness hyperproperties.

Theorem 3.6 *Every non-empty observable hyperproperty is a liveness hyperproperty.*

Proof: Let P be a non-empty observable hyperproperty. It must be the case that there exists a set of traces $M \in P$. Then, there exists $T \in Obs$ such that $T \leq M$ and $\forall M' \in Prop. T \leq M' \Rightarrow M' \in P$. For any set of traces $S \in Obs$, there exists $M' \in Prop$ such that $S \leq M'$. Then, we have $M \cup M' \in P$, because $T \leq M \cup M'$. Therefore, P is a liveness hyperproperty. □

We note that the empty set is not a liveness hyperproperty but an observable hyperproperty.

We show that lower-bounding problems for min-entropy and guessing-entropy are observable hyperproperties.

Theorem 3.7 *$\mathscr{L}_{ME}$ is an observable hyperproperty.*

Theorem 3.8 *$\mathscr{L}_{GE}$ is an observable hyperproperty.*

[4] We implicitly extend the notion of hyperproperties to classify hyperproperties that take programs and rational numbers. See [32].

[5] More precisely, we prove that they are liveness hyperproperties for deterministic systems [11], because we restrict all programs to deterministic programs. For sake of simplicity, we omit such annotations.

[6] Here, we assume that the input domains are not bounded. Therefore, we can construct a program that leaks more high-security inputs by enlarging the input domain. Hyperproperty classifications of bounding problems with bounded domains appear in Section 5.1.

[7] Roughly, an observable property is a set of traces having a finite evidence prefix such that any trace having the prefix is also in the set.

Theorem 3.7 follows from the fact that, if $(M,q) \in \mathscr{L}_{ME}$, then M contains an evidence of $\mathscr{L}_{ME}$. This follows from the fact that when a program M' contains at least as much observation as M, $ME[U](M) \leq ME[U](M')$ (cf. Lemma 3.15). Theorem 3.8 is proven in a similar manner.

We show that neither of the bounding problems for Shannon-entropy are observable hyperproperties.

Theorem 3.9 *Neither $\mathscr{U}_{SE}$ nor $\mathscr{L}_{SE}$ is an observable hyperproperty.*

We give the intuition of the proof for $\mathscr{U}_{SE}$. Suppose $SE[U](M) \leq q$. M does not provide an evidence of $SE[U](M) \leq q$, because for any potential evidence, we can raise the amount of the information flow by adding traces that have disjoint output traces. The result for $\mathscr{L}_{SE}$ is shown in a similar manner.

It is interesting to note that the bounding problems of *SE* can only be classified as general liveness hyperproperties (cf. Theorem 3.3 and 3.4) even though *SE* is often the preferred definition of QIF in practice [14, 9, 22]. This suggests that approximation techniques may be necessary for checking and inferring Shannon-entropy-based QIF.

3.2 K-Observable Hyperproperties

We define k-observable hyperproperty that refines the notion of observable hyperproperties. Informally, a k-observable hyperproperty is a hyperproperty that can be verified by observing k finite traces.

Definition 3.10 (K-Observable Hyperproperties) *We say that a hyperproperty P is a k-observable hyperproperty if for any set of traces $S \in P$, there exists $T \in Obs$ such that $T \leq S$, $|T| \leq k$, and for any set of traces $S' \in Prop$, $T \leq S' \Rightarrow S' \in P$.*

Clearly, any k-observable hyperproperty is an observable hyperproperty.

We note that k-observable hyperproperties can be reduced to 1-observable hyperproperties by a simple program transformation called *self composition* [5, 13].

Definition 3.11 (Parallel Self Composition [11]) *Parallel self composition of S is defined as follows.*

$$S \times S = \{(s[0],s'[0]);(s[1],s'[1]);(s[2],s'[2]);\cdots \mid s,s' \in S\}$$

where $s[i]$ denotes the ith element of s.

Then, a k-product parallel self composition (simply self composition henceforth) is defined as S^k.

Theorem 3.12 *Every k-observable hyperproperty can be reduced to a 1-observable hyperproperty via a k-product self composition.*

As an example, consider the following hyperproperty. The hyperproperty is the set of programs that return 1 and 2 for some inputs. Intuitively, the hyperproperty expresses two good things happen (programs return 1 and 2) for programs.

$$\{M \mid \exists h,h'.M(h) = 1 \wedge M(h') = 2\}$$

This is a 2-observable hyperproperty as any program containing two traces, one having 1 as the output and the other having 2 as the output, satisfies it.

We can check the above property by self composition. (Here, || denotes a parallel composition.)

$$M'(H,H') \quad \equiv \quad O := M(H) \,||\, O' := M(H') \,||\, \mathsf{assert}(\neg(\mathsf{O} = 1 \wedge \mathsf{O}' = 2))$$

Clearly, M satisfies the property iff the assertion failure is reachable in the above program, that is, iff the predicate $O = 1 \wedge O' = 2$ holds for some inputs H,H'. (Note that, for convenience, we take an assertion failure to be a "good thing".)

We show that neither the lower-bounding problem for min-entropy nor the lower-bounding problem for guessing-entropy is a k-observable hyperproperty for any k.

Theorem 3.13 *Neither $\mathscr{L}_{ME}$ nor $\mathscr{L}_{GE}$ is a k-observable property for any k.*

However, if we let q be a constant, then we obtain different results. First, we show that the lower-bounding problem for min-entropy-based quantitative information flow under a constant bound q, is a $\lfloor 2^q \rfloor + 1$-observable hyperproperty.

Theorem 3.14 *Let q be a constant. Then, $\mathscr{L}_{ME}$ is a $\lfloor 2^q \rfloor + 1$-observable hyperproperty.*

The theorem follows from Lemma 3.15 below which states that min-entropy based quantitative information flow under the uniform distribution coincides with the logarithm of the number of output traces. That is, $(M,q) \in \mathscr{L}_{ME}$ iff there is an evidence in M containing $\lfloor 2^q \rfloor + 1$ disjoint outputs.

Lemma 3.15 ([29]) $ME[U](M) = \log |\{o \mid \exists h.M(h) = o\}|$

Next, we show that the lower-bounding problem for guessing-entropy-based quantitative information flow under a constant bound q is a $\lfloor \frac{(\lfloor q \rfloor + 1)^2}{\lfloor q \rfloor + 1 - q} \rfloor + 1$-observable hyperproperty.

Theorem 3.16 *Let q be a constant. Then, $\mathscr{L}_{GE}$ is a $\lfloor \frac{(\lfloor q \rfloor + 1)^2}{\lfloor q \rfloor + 1 - q} \rfloor + 1$-observable hyperproperty.*

The proof of the theorem is similar to that of Theorem 3.14, in that the size of the evidence set can be computed from the bound q.

3.3 Computational Complexities

We prove computational complexities of $\mathscr{L}_{ME}$ and $\mathscr{L}_{GE}$ by utilizing their hyperproperty classifications. Following previous work [33, 32, 7], we focus on boolean programs.

First, we introduce the syntax of boolean programs. The semantics of boolean programs is standard. We call boolean programs without while statements *loop-free* boolean programs.

$$\begin{array}{rcl} M & ::= & x := \psi \mid M_0; M_1 \mid \mathsf{if}\ \psi\ \mathsf{then}\ M_0\ \mathsf{else}\ M_1 \mid \mathsf{while}\ \psi\ \mathsf{do}\ M \mid \texttt{skip} \\ \phi, \psi & ::= & \mathsf{true} \mid x \mid \phi \wedge \psi \mid \neg \phi \end{array}$$

Figure 1: The syntax of boolean programs

In this paper, we are interested in the computational complexity with respect to the syntactic size of the input program (i.e., "implicit state complexity", as opposed to [7] which studies complexity over programs represented as explicit states).

We show that the lower-bounding problems for min-entropy and guessing-entropy are *PP*-hard.

Theorem 3.17 *$\mathscr{L}_{ME}$ and $\mathscr{L}_{GE}$ for loop-free boolean programs are PP-hard.*

The theorem is proven by a reduction from MAJSAT, which is a *PP*-hard problem. *PP* is the set of decision problems solvable by a polynomial-time nondeterministic Turing machine which accepts the input iff more than half of the computation paths accept. MAJSAT is the problem of deciding, given a boolean formula ϕ over variables $\vec{x}$, if there are more than $2^{|\vec{x}|-1}$ satisfying assignments to ϕ.

Next, we show that if q be a constant, the upper-bounding problems for min-entropy and guessing-entropy become *NP*-complete.

Theorem 3.18 *Let q be a constant. Then, $\mathscr{L}_{ME}$ and $\mathscr{L}_{GE}$ are NP-complete for loop-free boolean programs.*

NP-hardness is proven by a reduction from SAT, which is a *NP*-complete problem. The proof that $\mathscr{L}_{ME}$ and $\mathscr{L}_{GE}$ for a constant q are in *NP* follows from the fact that $\mathscr{L}_{ME}$ and $\mathscr{L}_{GE}$ are k-observable hyperproperties for some k. We give the proof intuition for $\mathscr{L}_{ME}$. Recall that k-observable hyperproperties can be reduced to 1-observable hyperproperties via self composition. Consequently, it is possible to decide if the information flow of a given program M is greater than q by checking if the predicate of the assert statement is violated for some inputs in the following program.

$$\begin{array}{l} M'(H_1, H_2, \ldots, H_n) \equiv \\ \quad O_1 := M(H_1); O_2 := M(H_2); \ldots; O_n := M(H_n); \\ \quad \mathsf{assert}(\bigvee_{i,j \in \{1,\ldots,n\}} (O_i = O_j \wedge i \neq j)) \end{array}$$

where $n = \lfloor 2^q \rfloor + 1$. Let ϕ be the weakest precondition of $O_1 := M(H_1); O_2 := M(H_2); \ldots; O_n := M(H_n)$ with respect to the post condition $\bigvee_{i,j \in \{1,\ldots,n\}} (O_i = O_j \wedge i \neq j)$. Then, $ME[U](M) > q$ iff $\neg\phi$ is satisfiable. Because a weakest precondition of a loop-free boolean program is a polynomial size boolean formula over the boolean variables representing the inputs[8], deciding $ME[U](M) > q$ is reducible to SAT.

For boolean programs (with loops), $\mathscr{L}_{ME}$ and $\mathscr{L}_{GE}$ are *PSPACE*-complete, and $\mathscr{L}_{SE}$ is *PSPACE*-hard (the tight upper-bound is open for $\mathscr{L}_{SE}$).

Theorem 3.19 *$\mathscr{L}_{ME}$ and $\mathscr{L}_{GE}$ are PSPACE-complete for boolean programs.*

Theorem 3.20 *$\mathscr{L}_{SE}$ is PSPACE-hard for boolean programs.*

4 Safety Hyperproperties

Clarkson and Schneider [11] have proposed safety hyperproperties, a subclass of hyperproperties, as a generalization of safety properties. Intuitively, a safety hyperproperty is a hyperproperty that can be refuted by observing a finite set of finite traces.

Definition 4.1 (Safety Hyperproperties [11]) *We say that a hyperproperty P is a safety hyperproperty if for any set of traces $S \notin P$, there exists a set of traces $T \in Obs$ such that $T \leq S$, and $\forall S' \in Prop. T \leq S' \Rightarrow S' \notin P$.*

We classify some upper-bounding problems as safety hyperproperties.

Theorem 4.2 *U_{ME} and U_{GE} are safety hyperproperties.*

Next, we review the definition of k-safety hyperproperties [11], which refines the notion of safety hyperproperties. Informally, a k-safety hyperproperty is a hyperproperty which can be refuted by observing k number of finite traces.

Definition 4.3 (K-Safety Hyperproperties [11]) *We say that a hyperproperty P is a k-safety property if for any set of traces $S \notin P$, there exists a set of traces $T \in Obs$ such that $T \leq S$, $|T| \leq k$, and $\forall S' \in Prop. T \leq S' \Rightarrow S' \notin P$.*

Note that 1-safety hyperproperty is just the standard safety property, that is, a property that can be refuted by observing a finite execution trace. The notion of k-safety hyperproperties first came into limelight when it was noticed that non-interference is a 2-safety hyperproperty, but not a 1-safety hyperproperty [30].

A k-safety hyperproperty can be reduced to a 1-safety hyperproperty by self composition [5, 13].

[8] For loop-free boolean programs, a weakest precondition can be constructed in polynomial time [15, 21].

Theorem 4.4 ([11]) *k-safety hyperproperty can be reduced to* 1*-safety hyperproperty by self composition.*

We have shown in our previous work that $\mathscr{U}_{ME}$ and $\mathscr{U}_{GE}$ are k-safety hyperproperties when the bound q is fixed to a constant.

Theorem 4.5 ([32]) *Let q be a constant. $\mathscr{U}_{ME}$ is a $\lfloor 2^q \rfloor + 1$-safety property.*

Theorem 4.6 ([32]) *Let q be a constant. $\mathscr{U}_{GE}$ is a $\lfloor \frac{(\lfloor q \rfloor + 1)^2}{\lfloor q \rfloor + 1 - q} \rfloor + 1$-safety property.*

The only hyperproperty that is both a safety hyperproperty and a liveness hyperproperty is $\mathscr{P}(\Psi_{\mathtt{inf}})$, that is, the set of all traces [11]. Consequently, neither $\mathscr{U}_{ME}$ nor $\mathscr{U}_{GE}$ is a liveness hyperproperty.

We have also shown in the previous work that the upper-bounding problem for Shannon-entropy based quantitative information flow is not a k-safety hyperproperty, even when q is a constant.

Theorem 4.7 ([32]) *Let q be a constant. $\mathscr{U}_{SE}$ is not a k-safety property for any $k > 0$.*

4.1 Computational Complexities

We prove computational complexities of upper-bounding problems by utilizing their hyperproperty classifications. As in Section 3.3, we focus on boolean programs.

First, we show that when q is a constant, U_{ME} and U_{GE} are *coNP*-complete.

Theorem 4.8 *Let q be a constant. Then, $\mathscr{U}_{ME}$ and $\mathscr{U}_{GE}$ are coNP-complete for loop-free boolean programs.*

coNP-hardness follows from the fact that non-interference is *coNP*-hard [32]. The *coNP* part of the proof is similar to the *NP* part of Theorem 3.18, and uses the fact that $\mathscr{U}_{ME}$ is k-safety for a fixed q and uses self composition. By self composition, the upper-bounding problem can be reduced to a reachability problem (i.e., an assertion failure is unreachable for any input). To decide if $ME[U](M) \leq q$, we construct the following self-composed program M' from the given program M.

$$
\begin{array}{l}
M'(H_1, H_2, \ldots, H_n) \equiv \\
\quad O_1 := M(H_1); O_2 := M(H_2); \ldots; O_n := M(H_n); \\
\quad \mathsf{assert}(\bigvee_{i,j \in \{1,\ldots,n\}} (O_i = O_j \wedge i \neq j))
\end{array}
$$

where $n = \lfloor 2^q \rfloor + 1$. Then, the weakest precondition of $O_1 := M(H_1); O_2 := M(H_2); \ldots; O_n := M(H_n)$ with respect to the post condition $\bigvee_{i,j \in \{1,\ldots,n\}} (O_i = O_j \wedge i \neq j)$ is valid iff $ME[U](M) \leq q$. Because a weakest precondition of a loop-free boolean program is a polynomial size boolean formula, and the problem of deciding a given boolean formula is valid is a *coNP*-complete problem, $\mathscr{U}_{ME}$ is in *coNP*.

Like the lower-bounding problems $\mathscr{U}_{ME}$ and $\mathscr{U}_{GE}$ for boolean programs (with loops) are *PSPACE*-complete, and $\mathscr{U}_{SE}$ is *PSPACE*-hard.

Theorem 4.9 *$\mathscr{U}_{ME}$ and $\mathscr{U}_{GE}$ are PSPACE-complete for boolean programs.*

Theorem 4.10 *$\mathscr{U}_{SE}$ is PSPACE-hard for boolean programs.*

5 Discussion

5.1 Bounding Domains

The notion of hyperproperty is defined over all programs regardless of their size. (For example, non-interference is a 2-safety property for all programs and reachability is a safety property for all programs.) But, it is easy to show that the lower bounding problems would become "k-observable" hyperproperties if we constrained and bounded the input domains because then the size of the semantics (i.e., the number of traces) of such programs would be bounded by $|\mathbb{H}|$ (and upper bounding problems would become "k-safety" hyperproperties [32]). In this case, the problems are trivially $|\mathbb{H}|$-observable hyperproperties. However, these bounds are high for all but very small domains, and are unlikely to lead to a practical verification method.

5.2 Observable Hyperproperties and Observable Properties

As remarked in [11], observable hyperproperties generalize the notion of observable properties [2]. It can be shown that there exists a non-empty observable property that is not a liveness property (e.g., the set of all traces that starts with σ). In contrast, Theorem 3.6 states that every non-empty observable hyperproperty is also a liveness hyperproperty. Intuitively, this follows because the hyperproperty extension relation $\leq$ allows the right-hand side to contain traces that does not appear in the left-hand side. Therefore, for any $T \in \mathit{Obs}$, there exists $T' \in \mathit{Prop}$ that contains T and an evidence of the observable hyperproperty.

5.3 Maximum of QIF over Distribution

Researchers have studied the maximum of QIF over the distribution. For example, *channel capacity* [25, 23, 27] is the maximum of the Shannon-entropy based quantitative information flow over the distribution (i.e., $\max_\mu SE[\mu]$). Smith [29] showed that for any program without low-security inputs, the channel capacity is equal to the min-entropy-based quantitative information flow, that is, $\max_\mu SE[\mu] = ME[U]$. Therefore, we obtain the same hyperproperty classifications and complexity results for channel capacity as $ME[U]$.

Min-entropy channel capacity and *guessing-entropy channel capacity* are respectively the maximums of min-entropy based and guessing-entropy based QIF over distributions (i.e., $\max_\mu ME[\mu]$ and $\max_\mu GE[\mu]$). It has been shown that $\max_\mu ME[\mu] = ME[U]$ [6, 20] and $\max_\mu GE[\mu] = GE[U]$ [34], that is, they attain their maximums when the distributions are uniform. Therefore, they have the same hyperproperty classifications and complexities as $ME[U]$ and $GE[U]$, which we have already analyzed in this paper.

6 Related Work

Černý et al. [7] have investigated the computational complexity of Shannon-entropy based QIF. Formally, they have defined a Shannon-entropy based QIF for interactive boolean programs, and showed that the explicit-state computational complexity of their lower-bounding problem is *PSPACE*-complete. In contrast, this paper's complexity results are "implicit" complexity results of bounding problems of boolean programs (i.e., complexity relative to the syntactic size of the input) some of which are obtained by utilizing their hyperproperties classifications.

Clarkson and Schneider [11] have classified quantitative information flow problems via hyperproperties. Namely, they have shown that the problem of deciding if the channel capacity of a given program is q, is a liveness hyperproperty. And, they have shown that an upper-bounding problem for the *belief*-based QIF [10] is a safety hyperproperty. (It is possible to refine their result to show that their problem for deterministic programs is actually equivalent to non-interference, and therefore, is a 2-safety hyperproperty [34].)

7 Conclusion

We have related the upper and lower bounding problems of quantitative information flow, for various information theoretic definitions proposed in literature, to Clarkson and Schneider's hyperproperties. Hyperproperties generalize the classical trace properties, and are thought to be more suitable for classifying information flow properties as they are relations over sets of program traces. Our results confirm this by giving a fine-grained classification and showing that it gives insights into the complexity of the QIF bounding problems. One of the contributions is a new class of hyperproperties: *k-observable* hyperproperty. We have shown that k-observable hyperproperties are amenable to verification via self composition.

References

[1] (2010): *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*. IEEE Computer Society.

[2] Samson Abramsky (1991): *Domain Theory in Logical Form*. *Ann. Pure Appl. Logic* 51(1-2), pp. 1–77. Available at `http://dx.doi.org/10.1016/0168-0072(91)90065-T`.

[3] Michael Backes, Matthias Berg & Boris Köpf (2011): *Non-uniform distributions in quantitative information-flow*. In: *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, ACM, New York, NY, USA, pp. 367–375. Available at `http://doi.acm.org/10.1145/1966913.1966960`.

[4] Michael Backes, Boris Köpf & Andrey Rybalchenko (2009): *Automatic Discovery and Quantification of Information Leaks*. In: *IEEE Symposium on Security and Privacy*, IEEE Computer Society, pp. 141–153. Available at `http://dx.doi.org/10.1109/SP.2009.18`.

[5] Gilles Barthe, Pedro R. D'Argenio & Tamara Rezk (2004): *Secure Information Flow by Self-Composition*. In: *CSFW*, IEEE Computer Society, pp. 100–114. Available at `http://doi.ieeecomputersociety.org/10.1109/CSFW.2004.17`.

[6] Christelle Braun, Konstantinos Chatzikokolakis & Catuscia Palamidessi (2009): *Quantitative Notions of Leakage for One-try Attacks*. *Electr. Notes Theor. Comput. Sci.* 249, pp. 75–91. Available at `http://dx.doi.org/10.1016/j.entcs.2009.07.085`.

[7] Pavol Černý, Krishnendu Chatterjee & Thomas A. Henzinger (2011): *The Complexity of Quantitative Information Flow Problems*. In: *CSF*, IEEE Computer Society, pp. 205–217. Available at `http://doi.ieeecomputersociety.org/10.1109/CSF.2011.21`.

[8] David Clark, Sebastian Hunt & Pasquale Malacaria (2005): *Quantified Interference for a While Language*. *Electr. Notes Theor. Comput. Sci.* 112, pp. 149–166. Available at `http://dx.doi.org/10.1016/j.entcs.2004.01.018`.

[9] David Clark, Sebastian Hunt & Pasquale Malacaria (2007): *A static analysis for quantifying information flow in a simple imperative language*. *J. Comput. Secur.* 15, pp. 321–371. Available at `http://dl.acm.org/citation.cfm?id=1370628.1370629`.

[10] Michael R. Clarkson, Andrew C. Myers & Fred B. Schneider (2005): *Belief in Information Flow*. In: *CSFW*, IEEE Computer Society, pp. 31–45. Available at http://dx.doi.org/10.1109/CSFW.2005.10.

[11] Michael R. Clarkson & Fred B. Schneider (2010): *Hyperproperties*. *Journal of Computer Security* 18(6), pp. 1157–1210. Available at http://dx.doi.org/10.3233/JCS-2009-0393.

[12] Ellis S. Cohen (1977): *Information Transmission in Computational Systems*. In: *SOSP*, pp. 133–139. Available at http://doi.acm.org/10.1145/800214.806556.

[13] Ádám Darvas, Reiner Hähnle & David Sands (2005): *A Theorem Proving Approach to Analysis of Secure Information Flow*. In: Dieter Hutter & Markus Ullmann, editors: *SPC*, *Lecture Notes in Computer Science* 3450, Springer, pp. 193–209. Available at http://dx.doi.org/10.1007/978-3-540-32004-3_20.

[14] Dorothy Elizabeth Robling Denning (1982): *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[15] Cormac Flanagan & James B. Saxe (2001): *Avoiding exponential explosion: generating compact verification conditions*. In: *POPL*, pp. 193–205. Available at http://doi.acm.org/10.1145/360204.360220.

[16] Joseph A. Goguen & José Meseguer (1982): *Security Policies and Security Models*. In: *IEEE Symposium on Security and Privacy*, pp. 11–20.

[17] Jonathan Heusser & Pasquale Malacaria (2009): *Applied Quantitative Information Flow and Statistical Databases*. In: Pierpaolo Degano & Joshua D. Guttman, editors: *Formal Aspects in Security and Trust*, *Lecture Notes in Computer Science* 5983, Springer, pp. 96–110. Available at http://dx.doi.org/10.1007/978-3-642-12459-4_8.

[18] Boris Köpf & David A. Basin (2007): *An information-theoretic model for adaptive side-channel attacks*. In: Peng Ning, Sabrina De Capitani di Vimercati & Paul F. Syverson, editors: *ACM Conference on Computer and Communications Security*, ACM, pp. 286–296. Available at http://doi.acm.org/10.1145/1315245.1315282.

[19] Boris Köpf & Andrey Rybalchenko (2010): *Approximation and Randomization for Quantitative Information-Flow Analysis*. In *CSF* [1], pp. 3–14. Available at http://doi.ieeecomputersociety.org/10.1109/CSF.2010.8.

[20] Boris Köpf & Geoffrey Smith (2010): *Vulnerability Bounds and Leakage Resilience of Blinded Cryptography under Timing Attacks*. In *CSF* [1], pp. 44–56. Available at http://doi.ieeecomputersociety.org/10.1109/CSF.2010.11.

[21] K. Rustan M. Leino (2005): *Efficient weakest preconditions*. *Inf. Process. Lett.* 93(6), pp. 281–288. Available at http://dx.doi.org/10.1016/j.ipl.2004.10.015.

[22] Pasquale Malacaria (2007): *Assessing security threats of looping constructs*. In: Martin Hofmann & Matthias Felleisen, editors: *POPL*, ACM, pp. 225–235. Available at http://doi.acm.org/10.1145/1190216.1190251.

[23] Pasquale Malacaria & Han Chen (2008): *Lagrange multipliers and maximum information leakage in different observational models*. In: Úlfar Erlingsson & Marco Pistoia, editors: *PLAS*, ACM, pp. 135–146. Available at http://doi.acm.org/10.1145/1375696.1375713.

[24] James L. Massey (1994): *Guessing and Entropy*. In: *ISIT '94: Proceedings of the 1994 IEEE International Symposium on Information Theory*, p. 204. Available at http://dx.doi.org/10.1109/ISIT.1994.394764.

[25] Stephen McCamant & Michael D. Ernst (2008): *Quantitative information flow as network flow capacity*. In: Rajiv Gupta & Saman P. Amarasinghe, editors: *PLDI*, ACM, pp. 193–205. Available at http://doi.acm.org/10.1145/1375581.1375606.

[26] David A. Naumann (2006): *From Coupling Relations to Mated Invariants for Checking Information Flow*. In: Dieter Gollmann, Jan Meier & Andrei Sabelfeld, editors: *ESORICS*, *Lecture Notes in Computer Science* 4189, Springer, pp. 279–296. Available at http://dx.doi.org/10.1007/11863908_18.

[27] James Newsome, Stephen McCamant & Dawn Song (2009): *Measuring channel capacity to distinguish undue influence*. In: Stephen Chong & David A. Naumann, editors: *PLAS*, ACM, pp. 73–85. Available at http://doi.acm.org/10.1145/1554339.1554349.

[28] Claude Shannon (1948): *A Mathematical Theory of Communication*. *Bell System Technical Journal* 27, pp. 379–423, 623–656. Available at http://doi.acm.org/10.1145/584091.584093.

[29] Geoffrey Smith (2009): *On the Foundations of Quantitative Information Flow*. In: Luca de Alfaro, editor: *FOSSACS*, *Lecture Notes in Computer Science* 5504, Springer, pp. 288–302. Available at http://dx.doi.org/10.1007/978-3-642-00596-1_21.

[30] Tachio Terauchi & Alexander Aiken (2005): *Secure Information Flow as a Safety Problem*. In: Chris Hankin & Igor Siveroni, editors: *SAS*, *Lecture Notes in Computer Science* 3672, Springer, pp. 352–367. Available at http://dx.doi.org/10.1007/11547662_24.

[31] Hiroshi Unno, Naoki Kobayashi & Akinori Yonezawa (2006): *Combining type-based analysis and model checking for finding counterexamples against non-interference*. In: Vugranam C. Sreedhar & Steve Zdancewic, editors: *PLAS*, ACM, pp. 17–26. Available at http://doi.acm.org/10.1145/1134744.1134750.

[32] Hirotoshi Yasuoka & Tachio Terauchi (2010): *On Bounding Problems of Quantitative Information Flow*. In: Dimitris Gritzalis, Bart Preneel & Marianthi Theoharidou, editors: *ESORICS*, *Lecture Notes in Computer Science* 6345, Springer, pp. 357–372. Available at http://dx.doi.org/10.1007/978-3-642-15497-3_22.

[33] Hirotoshi Yasuoka & Tachio Terauchi (2010): *Quantitative Information Flow - Verification Hardness and Possibilities*. In *CSF* [1], pp. 15–27. Available at http://doi.ieeecomputersociety.org/10.1109/CSF.2010.9.

[34] Hirotoshi Yasuoka & Tachio Terauchi (2011): *On Bounding Problems of Quantitative Information Flow (Extended version)*. *Journal of Computer Security* 19(6), pp. 1029–1082. Available at http://dx.doi.org/10.3233/JCS-2011-0437.

[35] Hirotoshi Yasuoka & Tachio Terauchi (2011). *Quantitative Information Flow as Safety and Liveness Hyperproperties*. Available at http://www.kb.ecei.tohoku.ac.jp/~yasuoka.

Differential privacy for relational algebra: improving the sensitivity bounds via constraint systems*

Catuscia Palamidessi
INRIA and LIX, Ecole Polytechnique, France
catuscia@lix.polytechnique.fr

Marco Stronati
Università di Pisa, Italy
marco.stronati@gmail.com

Abstract. Differential privacy is a modern approach in privacy-preserving data analysis to control the amount of information that can be inferred about an individual by querying a database. The most common techniques are based on the introduction of probabilistic noise, often defined as a Laplacian parametric on the sensitivity of the query. In order to maximize the utility of the query, it is crucial to estimate the sensitivity as precisely as possible.
In this paper we consider relational algebra, the classical language for queries in relational databases, and we propose a method for computing a bound on the sensitivity of queries in an intuitive and compositional way. We use constraint-based techniques to accumulate the information on the possible values for attributes provided by the various components of the query, thus making it possible to compute tight bounds on the sensitivity.

1 Introduction

Differential privacy [6, 7, 8, 9] is a recent approach addressing the privacy of individuals in data analysis on statistical databases. In general, statistical databases are designed to collect global information in some domain of interest, while the information about the particular entries is supposed to be kept confidential. Unfortunately, querying a database might leak information about an individual, because the presence of her record may induce the query to return a different result.
To illustrate the problem, consider for instance a database of people affected by a certain disease, containing data such as age, height, etc. Usually the identity of the people present in the database is supposed to be secret, but if we are allowed to query the database for the number of records which are contained in it, and for – say – the average value of the data (height, age, etc.), then one can infer the precise data of the last person entry in the database, which poses a serious threat to the disclosure of her identity as well.
To avoid this problem, one of the most commonly used methods consists in introducing some *noise* on the answer. In other words, instead of giving the *exact* answer the curator gives an *approximated* answer, chosen randomly according to some probability distribution.
Differential privacy measures the level of privacy provided by such a randomized mechanism by a parameter ε: a mechanism $\mathcal{K}$ is ε-differentially private if for every pair of *adjacent* databases R and R' (i.e. databases which which differ for only one entry), and for every property $\mathcal{P}$, the probabilities that $\mathcal{K}(R)$ and $\mathcal{K}(R')$ satisfy $\mathcal{P}$ differ at most by the multiplicative constant e^{ε}.
The amount of noise that the mechanism must introduce in order to achieve ε differential privacy depends on the so-called *sensitivity* of the query, namely the maximum distance between the answers on two adjacent databases. For instance, one of the most commonly used mechanisms, the *Laplacian*, adds noise

*This work has been partially supported by the project ANR-09-BLAN-0169-01 PANDA

Preliminary Report. Final version to appear in:
QAPL 2012

to the correct answer y by reporting an approximated answer z according to the following probability density function:

$$P_y(z) = c\, e^{-\frac{|y-z|}{\Delta f}\varepsilon}$$

where Δf is the sensitivity of the query f, and c is a normalization factor. Clearly, the higher is the sensitivity, the greater the noise, in the sense that the above function is more "flat", i.e. we get a higher probability of reporting an answer very different from the exact one.

Of course, there is a trade off between the privacy and the *utility* of a mechanism: the more noise a mechanism adds, the less precise the reported answer, which usually means that the result of querying the database becomes less useful – whatever the purpose.

For this reason, it is important to avoid adding excessive noise: one should add only the noise strictly necessary to achieve the desired level of differential privacy. This means that the sensitivity of the query should be computed as precisely as possible. At the same time, for the sake of efficiency it is desirable that the computation of the sensitivity is done *statically*. Usually this implies that we cannot compute the precise sensitivity, but only approximate it from above. The goal of this paper is to explore a *constraint-based methodology* in order to compute strict upper bounds on the sensitivity.

The language we chose to conduct our analysis is *relational algebra* [4, 5], a formal and well defined model for relational databases, that is the basis for the popular Structured Query Language (SQL, [2]). It consists in a collection of few operators that take relations as input and return relations as output, manipulating rows or columns and computing aggregation of values.

Sensitivity on aggregations often depends on attribute ranges, and these restrictions can be exploited to provide better bounds. To this purpose, we extend mechanisms already in place in modern database systems: In RDBMS (Relational Data Bases Management Systems) implementations, during the creation of a relation, it is possible to define a set of constraints over the attributes of the relation, to further restrict the type information. For instance:

```
Persons{(Name, String)(Age, Integer)} {Age > 0 ∧ Age < 120)}
```

refines the type integer used to express the age of a person in the database, by establishing that it must be a positive value smaller than 120.

Constraints in RDBMS can be defined on single attributes (*column constraints*), or on several attributes (*table constraints*), and help define the structure of the relation, for example by stating whether an attribute is a primary key or a reference to an external key. In addition, so called *check constraints* can be defined, to verify the insertion of correct values. In the example above, for instance, the constraint would avoid inserting an age of, say, 200. Check constraints are particularly useful for our purposes because they restrict the possible values of the attributes, thus allowing a finer analysis of the sensitivity.

Contribution Our contribution is twofold:

1. we propose a method to compute a bound on the sensitivity of a query in relational algebra in a compositional way, and
2. we propose the use of constraints and constraint solvers to refine the method and obtain strict bounds on queries which have aggregation functions at the top level.

Plan of the paper Next section recalls some preliminary notions about relational databases and differential privacy. Section 3 introduces a constraint system and the idea of carrying along the information provided by the constraints as we analyze the query. Section 4 proposes a generalization of differential

privacy and sensitivity to generic metric spaces. This generalization will be useful in order to compute the sensitivity of a query in a compositional way. Sections 6, 7 and 8 analyze the sensitivity and the propagation of constraints for the various operators of relational algebra. Finally Section 9 proposes a method to compute a sensitivity bound on the global query, and shows its correctness and the improvement provided by the use of constraints. Section 10 discusses some related work, and Section 11 concludes. Due to space limitations, in this version we have omitted several proof. The interested reader can find them in the full online version of the paper [3].

2 Preliminaries

We recall here some basic notions about relational databases and relational algebra, differential privacy, and sensitivity.

2.1 Relational Databases and Relational Algebra

Relational algebra [4, 5] can be considered as the theoretic foundation of database query languages and in particular of SQL [2]. It is based on the concept of relation, which is the mathematical essence of a (relational) database, and of certain operators on relations like union, intersection, projections, filters, etc.. Here we recall the basic terminology used for relational databases, while the operators will be illustrated in detail in the technical body of the paper.
A relation (or database) based on a certain schema is a collection of tuples (or records) of values. The schema defines the types (domain) and the names (attributes) of these values.

Definition 1 (Relation Schema). *A relation schema $r(a_1 : D_1, a_2 : D_2, \ldots, a_n : D_n)$ is composed of the relation name r and a set of attributes $a_1, a_2, \ldots, a_n$ associated with the domains $D_1, D_2, \ldots, D_n$, respectively. We use the notation $dom(a_i)$ to refer to D_i.*

Definition 2 (Relation). *A relation R on a relation schema $r(a_1 : D_1, a_2 : D_2, \ldots, a_n : D_n)$ is a subset of the Cartesian product $D_1 \times D_2 \times \ldots \times D_n$.*

A relation is thus composed by a set of n-tuples, where each n-tuple τ has the form $(d_1, d_2, \ldots, d_n)$ with $d_i \in D_i$. Note that τ can also be seen as a partial function from attributes to atomic values, i.e. $\tau(a_i) = d_i$. Given a schema, we will denote the universe of possible tuples by $\mathcal{T}$, and the set of all possible relations by $\mathcal{R} = 2^{\mathcal{T}}$.
Relational algebra is a language that operates from relations to relations. Differentially private queries, however, can only return a value, and for this reason they must end with an aggregation (operator γ). Nevertheless it is possible to show that the full power of relational algebra aggregation can be retrieved.

2.2 Differential Privacy

Differential privacy is a property meant to guarantee that the participation in a database does not constitute a threat for the privacy of an individual. More precisely, the idea is that a (randomized) query satisfies differential privacy if two relations that differ only for the addition of one record are almost indistinguishable with respect to the results of the query.
Two relations $R, R' \in \mathcal{R}$ that differ only for the addition of one record are called *adjacent*, denoted by $R \sim R'$. Formally, $R \sim R'$ iff $R \setminus R' = \{\tau\}$ or viceversa $R' \setminus R = \{\tau\}$, where τ is a tuple.

Definition 3 (Differential privacy [6]). *A randomized function $\mathcal{K} : \mathcal{R} \to Z$ satisfies ε-differential privacy if for all pairs $R, R' \in \mathcal{R}$, with $R \sim R'$, and all $Y \subseteq Z$, we have that:*

$$Pr[\mathcal{K}(R) \in Y] \leq Pr[\mathcal{K}(R') \in Y] \cdot e^{\varepsilon}$$

where $Pr[E]$ represents the probability of the event E.

Differentially private mechanisms are usually obtained by adding some random noise to the result of the query. The best results are obtained by calibrating the noise distribution according to the so-called sensitivity of the query. When the answers to the query are real numbers ($\mathbb{R}$), its sensitivity is defined as follows. (We represent a query as a function from databases to the domain of answers.)

Definition 4 (Sensitivity [6]). *Given a query $Q : \mathcal{R} \to \mathbb{R}$, the sensitivity of Q, denoted by Δ_Q, is defined as:*

$$\Delta_Q = \sup_{R \sim R'} |\, Q(R) - Q(R') \,|.$$

The above definition can be extended to queries with answers on generic domains, provided that they are equipped with a notion of distance.

3 Databases with constraints

As explained in the introduction, one of the contributions of our paper is to provide strict bounds on the sensitivity of queries by using constraints. For an introduction to the notions of constraint, constraint solver, and constraint system we refer to [1].
In this section we define the constraint system that we will use, and we extend the notion of database schema so to accommodate the additional information provided by the constrains during the analysis of a query.

Definition 5 (Constraint system). *Our constraint system is defined as follows:*

- *Terms are constructed from:*
  - *variables, ranging over the attribute names of the schemas,*
  - *constants, ranging over the domains of the schemas,*
  - *applications of n-ary functions (e.g. $+, \times$) to n terms.*
- *Atoms are applications of n-ary predicates to n terms. Possible predicates are $\geq, \leq, =, \in$.*
- *Constraints are constructed from:*
  - *atoms, and*
  - *applications of logical operators ($\neg, \wedge, \vee, \equiv$) to constraints.*

We denote the composition of constraint by $\otimes$. The solutions of a set of constraints C is the set of tuples that satisfy C, denoted $Sol(C)$. The relations that can be build from $sol(C)$ are denoted by $\mathcal{R}(C) = \mathcal{P}(sol(C))$. The solutions with respect to an attribute a is denoted $sol(C,a)$. Namely, $sol(C,a)$ is the projection on a of $sol(C)$. When the domain is equipped with an ordering relation, we also use $inf(C,a)$ and $sup(C,a)$ to denote the infimum and the supremum values, respectively, of $sol(C,a)$. Typically the solutions and the *inf* and *sup* values can be computed automatically using constraint solvers. Finally we define the diameter of a constraint C as the maximum distance between the solutions of C.

Definition 6 (Diameter). *The diameter of a constraint C, denoted $diam(C)$, is the graph diameter of the adjacency graph $(\mathcal{R}(C), \sim)$ of all possible relations composed by tuples that satisfy C.*

We now extend the classical definition of schema to contain also the set of constraints.

Definition 7 (Constrained schema). *A constrained schema $r(A,C)$ is composed of the relation name r, a set of attributes A, and a set of constraints C. A relation on a constrained schema is a subset of $sol(C)$. We will use $schema(R)$ to represent the constrained relation schema of a relation R.*

The above definition extends the notion of relation schema (Definition 1): In fact here each a_i can be seen as associated with $sol(C,a_i)$. Definition 1 can then be retrieved by imposing as only constraints those of the form $a_i \in D_i$.

Example 1. *Consider the constrained schema* $\texttt{Items}(A,C)$*, where* $A = \{\texttt{Item},\texttt{Price},\texttt{Cost}\}$*, and* $C = \{(\texttt{Cost} \leq \texttt{Price} \leq 1000, 0 < \texttt{Cost} \leq 1000)\}$*. The following R is a possible relation over this schema. R:*

			Item	Price	Cost
Items	{Item, Price, Cost}		Oil	100	10
	{(Cost ≤ Price ≤ 1000, 0<Cost≤1000)}		Salt	50	11
	Items(A,C)			R	

4 Differential privacy on arbitrary metrics

The classic notions of differential privacy and sensitivity are meant for queries defined on $\mathcal{R}$, the set of all relations on a given schema. The adjacency relation induces a graph structure (where the arcs correspond to the adjacency relation), and a metric structure (where the distance is defined as the distance on the graph).
In order to compute the sensitivity bounds in a compositional way, we need to cope with different structures at the intermediate steps, and with different notions of distance. Consequently, we need to extend the notions of differential privacy and sensitivity to general metric domains.
We start by defining the notions of distance that we will need.

Definition 8 (Hamming distance d_H). *The distance between two relations $R,R' \in \mathcal{R}$ is the Hamming distance $d_H(R,R') = |R \ominus R'|$, the cardinality of the symmetric difference between R and R'. The symmetric difference is defined as $R \ominus R' = (R \setminus R') \cup (R' \setminus R)$.*

Note that d_H coincides with the graph-theoretic distance on the graph induced by the adjacency relation $\sim$, and that $d_H(R,R') = 1 \Leftrightarrow R \sim R'$. We now extend the Hamming distance to tuples of relations, to deal with n-ary operators.

Definition 9 (Distance d_{nH}). *The distance d_{nH} between two tuples of n relations $(R_1,\ldots,R_n),(R'_1,\ldots,R'_n) \in \mathcal{R}^n$ is defined as: $d_{nH}((R_1,\ldots,R_n),(R'_1,\ldots,R'_n)) = max(d_H(R_1,R'_1),\ldots,d_H(R_n,R'_n))$*

Note that d_{nH} coincides with the Hamming distance for $n = 1$. We chose this maximum metric instead of other distances because it allows us to compute the sensitivity compositionally, while this is not the case for other notions of distance. We can show counterexamples, for instance, for both the Euclidian and the Manhattan distances.

Definition 10 (Distance d_E). *The distance between two real numbers $x,x' \in \mathbb{R}$ is the usual euclidean distance $d_E(x,x') = |x-x'|$.*

In summary, we have two metric spaces over which the relational algebra operators work, namely $(\mathcal{R}^n, d_{nH})$, and $(\mathbb{R}, d_E)$.

Example 2. *Consider a relation R and two tuples τ, π such that $\tau \notin R$ and $\pi \in R$. We define its neighbors R^+ and $R^\pm$, obtained by adding one record, and by changing one record, respectively:*

$$R^+ = R \cup \{\tau\} \qquad R^\pm = R \cup \{\tau\} \setminus \{\pi\}$$

Their distance from R is : $d_H(R,R^+) = |R \ominus R^+| = 1$, and $d_H(R,R^\pm) = |R \ominus R^\pm| = 2$. Note also that $R \sim R^+$.

Notation 1. *In the following, we will use the notation R^+ to denote $R \cup \{\tau\}$ for a generic tuple τ, with the assumption (unless otherwise specified) that $\tau \notin R$.*

We now adapt the definition of differential privacy to arbitrary metric spaces (X,d) (where X is the support set and d the distance function).

Definition 11 (Differential privacy extended). *A randomized mechanism $\mathcal{K} : \mathcal{X} \to \mathcal{Z}$ on a generic metric space (X,d) provides ε-differential privacy if for any $x,x' \in X$, and any set of possible outputs $Y \subseteq Z$,*

$$Pr[\mathcal{K}(x) \in Y] \;\leq\; Pr[\mathcal{K}(x') \in Y] \cdot e^{\varepsilon \cdot d(x,x')}$$

It can easily be shown that Definitions 11 and 3 are equivalent if $d = d_H$.
We now define the sensitivity of a function on a generic metric space.

Definition 12 (Sensitivity extended). *Let (X,d_X) and (Y,d_Y) be metric spaces. The sensitivity Δ_f of a function $f : (X,d_X) \to (Y,d_Y)$ is defined as*

$$\Delta_f \;=\; \sup_{\substack{x,x' \in X \\ x \neq x'}} \frac{d_Y(f(x),f(x'))}{d_X(x,x')}$$

Again, we can show that Definitions 12 and 4 are equivalent if $d_X = d_{nH}$ (proof in full version [3]). This more general definition makes clear that the sensitivity of a function is a measure of how much it increases distances from its inputs to its outputs.
As a refinement of the definition of sensitivity, we may notice that this attribute does not depend on the function alone, but also on the domain, where the choice of x,x' ranges to compute the supremum. In our framework this is particularly useful because we have a very precise description of the restrictions on the domain of an operator, thanks to its input constrained schema (Def 7).

Definition 13 (Sensitivity constrained). *Given a function $f : (X,d_X) \to (Y,d_Y)$, and a set of constraints C on X, the sensitivity of f with respect to C is defined as*

$$\Delta_f(C) \;=\; \sup_{\substack{x,x' \in sol(C) \\ x \neq x'}} \frac{d_Y(f(x),f(x'))}{d_X(x,x')}$$

The introduction of constraints, in addition to an improved precision, allows us to define conveniently function composition. It should be noted that when combining two functions $f \circ g$, where $g : (Y,d_Y) \to (Z,d_Z)$, the domain of g actually depends on the restrictions introduced by f and we can take this into account maximizing over $y,y' \in sol(C \otimes C_f)$, that is the domain obtained combining the initial constraint C and the constraint introduced by f.

5 Operators

We now proceed to compute a bound on the sensitivity of each relational algebra operator through a static analysis that depends only on the relation schema the operator is applied to, and not on its particular instances.
From a static point of view each operator will be considered as a transformation from schema to schema (instead of a transformation from relations to relations): they may add or remove attributes, and modify constraints.
The following analysis is split in operators $\mathtt{op} : (\mathcal{R}^n, d_{nH}) \to (\mathcal{R}, d_H)$, with n equals 1 or 2, and aggregation $\gamma_f : (\mathcal{R}, d_H) \to (\mathbb{R}, d_E)$. In the sensitivity analysis of the formers, given they work only on Hamming metrics, we are only interested in their effect on the number of rows. In our particular case, these relational algebra operators treats all rows equally, without considering their content. This simplification grants us the following property:

Proposition 1. *If* $\mathtt{op} : (\mathcal{R}, d_H) \to (\mathcal{R}, d_H)$ *and C is an arbitrary set of constraints*

$$\Delta_{\mathtt{op}}(C) = \sup_{\substack{R,R' \in \mathcal{R}(C) \\ R \neq R'}} \frac{d_H(\mathtt{op}(R), \mathtt{op}(R'))}{d_H(R,R')} = \min\left(\Delta_{\mathtt{op}}(\emptyset), diam(C \otimes C_{\mathtt{op}})\right)$$

(The proposition holds analogously for the binary case). This property, that does not hold for general functions, allows us in the case of relational algebra to decouple the computation of sensitivity from the constraint system, and solve them separately. $\Delta_{\mathtt{op}}(\emptyset)$ (from now on just $\Delta_{\mathtt{op}}$) can be seen as the sensitivity intrinsic to each operator, the maximum value of sensitivity the operator can cause, when the constraints are loose enough [1] to be omitted. While $diam(C \otimes C_{\mathtt{op}})$, the diameter of the co-domain of the operator, limits the maximum distance the operator can produce, that is the numerator in the distances ratio.

6 Row operators

In this section we consider a first group of operators of type $(\mathcal{R}^n, d_{nH}) \to (\mathcal{R}, d_H)$ with $n = 1, 2$, which are characterized by the fact that they can only add or remove tuples, not modify their attributes. Indeed the header of the resulting relation maintains the same set of attributes and only the relative constraints may be modified.

6.1 Union $\cup$

The union of two relation is the set theoretic union of two set of tuples with the same attributes. The example below illustrates this operation:

Name	Age	Height
John	30	180
Tim	10	100

$\cup$

Name	Age	Height
Alice	45	160
Tim	10	100

=

Name	Age	Height
John	30	180
Tim	10	100
Alice	45	160

The union of two relations may reduce their distance to zero or leave it unchanged.

Proposition 2. *The union has sensitivity* 2*:* $\Delta_\cup = 2$.

[1] for all possible domains the sensitivity can't be greater.

Proof. If $d_{2H}((R_1,R_2),(R_3,R_4)) = 1$ then we have two cases

a) $R_3 = R_1^+, R_4 = R_2$ or $R_3 = R_1, R_4 = R_2^+$. For the symmetry of distance only one case needs to be considered:

$$|(R_1 \cup R_2) \ominus (R_1^+ \cup R_2)| = \begin{cases} 0 & \tau \in R_2 \\ 1 & o.w. \end{cases}$$

The only difference is the tuple τ. If $\tau \in R_2$ then τ would be in both results, leading to identical relations, thus reducing the distance to zero. If $\tau \notin R_2$ then τ will again be the only difference between the results, thus resulting into distance 1.

b) $R_3 = R_1^+, R_4 = R_2^+$

$$|(R_1 \cup R_2) \ominus (R_1^+ \cup R_2^+)| = \begin{cases} 0 & \tau_1 \in R_2 \wedge \tau_2 \in R_1 \\ 1 & \tau_1 \in R_2 \vee \tau_2 \in R_1 \\ 2 & \tau_1 \notin R_2 \wedge \tau_2 \notin R_1 \end{cases}$$

In this case we have two records differing, τ_1 and τ_2, and in the worst case they may remain different in the results, giving a final sensitivity of 2 for the operator. □

Definition 14 (Constraints for union). *Let schema*$(R_1) = (A, C_1)$ *and schema*$(R_2) = (A, C_2)$. *Then schema*$(R_1 \cup R_2) = (A, C_1 \vee C_2)$.

6.2 Intersection $\cap$

The intersection of two relation is the set theoretic intersection of two set of tuples with the same attributes.
As for the union, the difference applied to argument at distance 1 may have an effect on the only tuple in which the arguments differ, thus resulting into a distance 0 or 1.

Proposition 3. *The intersection has sensitivity* 2*:* $\Delta_\cap = 2$.

Proof. Similar to the case of Proposition 2. □

Definition 15 (Constraints for intersection). *Let schema*$(R_1) = (A, C_1)$ *and schema*$(R_2) = (A, C_2)$. *Then schema*$(R_1 \cap R_2) = (a, C_1 \wedge C_2)$.

6.3 Difference $\setminus$

The difference of two relation is the set theoretic difference of two set of tuples with the same attributes.
As in the case of the union, the difference applied to argument at distance 1 may have an effect on the only tuple in which the arguments differ, thus resulting into a distance 0 or 1.

Proposition 4. *The set difference has sensitivity* 2*:* $\Delta_\setminus = 2$.

Proof. Similar to the case of Proposition 2. □

Definition 16 (Constraints for set difference). *Let schema*$(R_1) = (A, C_1)$ *and schema*$(R_2) = (A, C_2)$. *Then schema*$(R_1 \setminus R_2) = (A, C_1 \wedge (\neg C_2))$.

6.4 Restriction σ

The restriction operator $\sigma_\varphi(R)$ removes all rows not satisfying the condition φ (typically constructed using the predicates $=, \neq, <, >$ and the logical connectives $\vee, \wedge, \neg$), over a subset of R attributes.
As an example, consider the following SQL program that removes all people whose age is smaller than 20 or whose height is greater than 180. The table illustrates an example of application of the corresponding restriction $\sigma_{Age \geq 20 \wedge Height < 180}$.

```
SELECT *
FROM   R
WHERE  Age>=20 AND Height<=180
```

$\sigma_{Age \geq 20 \wedge Height < 180}$

Name	Age	Height
John	30	180
Tim	10	100
Alice	45	160
Natalie	20	175

=

Name	Age	Height
Alice	45	160
Natalie	20	175

The restriction can be expressed in terms of set difference: $\sigma_\varphi(R) = R \setminus \{\tau \mid \neg\varphi(\tau)\}$. However the sensitivity is different because the operator is unary, the second argument is fixed by the condition φ

Proposition 5. *The restriction has sensitivity* 2*:* $\Delta_{\sigma_\varphi} = 1$.

Definition 17 (Constraints for restriction). *Let schema*$(R) = (A, C)$ *and* $A' \subseteq A$*. Then define schema*$(\sigma_{\varphi(A')}(R)) = (A, C \wedge \varphi(A'))$.

7 Attribute operators

The following set of operators, unlike those analyzed so far, can affect the number of tuples of a relation, as well as its attributes.

7.1 Projection π

The projection operator $\pi_{a_1,\ldots,a_n}(R)$ eliminates the columns of R with attributes other than $a_1, \ldots, a_n$, and then deletes possible duplicates, thus reducing distances or leaving them unchanged. It is the opposite of the restricted Cartesian product $\times_1$ which will be presented later.
The following example illustrates the use of the projection. Here, the attribute to preserve are `Name` and `Age`.

```
SELECT Name,Age
FROM   R
```

$\pi_{Name,Age}$

Name	Age	Car
John	30	Ford
John	30	Renault
Alice	45	Fiat

=

Name	Age
John	30
Alice	45

Proposition 6. *The projection has sensitivity* 1*:* $\Delta_\pi = 1$.

Proof. $|\pi_{a_1,\ldots,a_n}(R) \ominus \pi_{a_1,\ldots,a_n}(R^+)| = \begin{cases} 0 & \exists \rho \in R.\, \forall i \in \{1,\ldots,n\}\ \rho(a_i) = \tau(a_i) \\ 1 & o.w. \end{cases}$ □

Definition 18 (Constraints for projection). *Let schema*$(R) = (A, C)$ *and* $A' \subseteq A$*. Then schema*$(\pi_{A'}(R)) = (A', C)$.

7.2 Cartesian product

The Cartesian product of two relation is the set theoretic Cartesian product of two set of tuples with different attributes, with the exception that in relations the order of attributes does not count, thus making the operation commutative. The following example illustrate this operation.

Name	Age	Height
John	30	180
Alice	45	160

$\times$

Car	Owner
Fiat	Alice
Ford	Alice

$=$

Name	Age	Height	Car	Owner
John	30	180	Fiat	Alice
John	30	180	Ford	Alice
Alice	45	160	Fiat	Alice
Alice	45	160	Ford	Alice

This operator may seem odd in the context of a query language, but it is in fact the base of the *join*, the operator to merge the information of two relations.

$$R \underset{R.a_i=T.a_i}{\bowtie} T = \sigma_{R.a_i=T.a_i}(R \times T)$$

We analyze now the sensitivity of the Cartesian product.

One record $\times_1$ We first consider a restricted version $\times_1$, where on one side we have a single tuple.

Proposition 7. *The operator $\times_1$ has sensitivity 1: $\Delta_{\times_1} = 1$.*

N records $\times$ We consider now the full Cartesian product operator. It is immediate to see that a difference of a single row can be expanded to an arbitrary number of records, thus causing and unbounded sensitivity.

Proposition 8. *The (unrestricted) Cartesian product has unbounded sensitivity.*

We now define how constraints propagate through Cartesian product:

Definition 19 (Constraints for product). *Let schema$(R_1) = (A_1, C_1)$ and schema$(R_2) = (A_2, C_2)$. Then schema$(R_1 \times R_2) = (A_1 \cup A_2, C_1 \wedge C_2)$.*

7.3 Restricted $\times$

The effect of Cartesian product is to expand each record with a block of records, a behavior clearly against our objective of distance-preserving computations. However we propose some restricted versions of the operator in order to maintain its functionality to a certain extent:

- $\times_n$: product with blocks of a fixed n size, to obtain n sensitivity. In this case n representative elements can be chosen from the relation, the definition of policies to pick these elements is left to future developments.
- $\times_\gamma$: a new single record is built as an aggregation of the relation, through the operator $\emptyset\gamma f$ (presented later), thus falling in the case of $\times_1$ sensitivity.
- a mix the two approaches could be considered, building n aggregations, possibly using the operator ${}_{\{a_i\}}\gamma_f$ (presented later).

In both approaches the rest of the query can help to select the right records from the block, for example an external restriction could be anticipated.

8 Aggregation γ

The classical relational algebra operator for aggregation ${}_{\{a_1,\ldots,a_m\}}\gamma_{\{f_1,\ldots,f_k\}}(R)$ performs the following steps:

- it partitions R, so that each group has all the tuples with the same values for each a_i,
- it computes all f_i for each group,
- it returns a single tuple for each group, with the values of a_i and of f_i.

The most common function founds on RDBMS are `count,max,min,avg,sum` and we will restrict our analysis to these ones. The following example illustrates how we can use an aggregation operator to know, for each type of `Car`, how many people own it and what is their average height.

```
SELECT  Car, Count(*), Avg(Height)
FROM    R
GROUPBY Car
```

$${}_{\{Car\}}\gamma_{\{Count,Avg(Height)\}}\left(\begin{array}{llll} \text{Name} & \text{Age} & \text{Height} & \text{Car} \\ \hline \text{Alice} & 45 & 160 & \text{Ford} \\ \text{John} & 30 & 180 & \text{Fiat} \\ \text{Frank} & 45 & 165 & \text{Renault} \\ \text{Natalie} & 20 & 170 & \text{Ford} \end{array}\right) = \begin{array}{lll} \text{Car} & \text{Count} & \text{Avg(Height)} \\ \hline \text{Ford} & 2 & 165 \\ \text{Fiat} & 1 & 180 \\ \text{Renault} & 1 & 165 \end{array}$$

In the domain of differential privacy special care must be taken when dealing with this operator as it is in fact the point of the query in which our analysis of sensitivity ends and the noise must be added to the result of the function application.
A differentially private query should return a single value, in our case in $\mathbb{R}$, and the only queries that statically guarantee this property are those ending with the operator ${}_{\emptyset}\gamma_f : (\mathcal{R}, d_H) \to (\mathbb{R}, d_E)$ (from here on abbreviated γ_f), that apply only one function f to the whole relation without grouping. For this reason we will ignore grouping for now, and focus on queries of the form ${}_{\emptyset}\gamma_f(Q)$ where Q is a sub-query without aggregations. It is however possible to recover the original ${}_A\gamma_F$ behavior and use it in sub-queries.

8.1 Functions

In this section we analyze the sensitivity of the common mathematical functions `count,sum, max,min` and `avg`. The application of functions coincide with the change of domain, in fact they take as input a relation in $(\mathcal{R}, d_H)$ and return a single number in $(\mathbb{R}, d_E)$, (not to be confused with a relation with a single tuple, which also contains a single value).
Extending standard results [6], we can prove that, when $f = \texttt{count,sum,max,min,avg}$ then $\Delta_f(C)$ can be computed as follows:

Proposition 9.

$$\begin{aligned} \Delta_{\texttt{count}}(C) &= 1 \\ \Delta_{\texttt{sum}_{a_i}}(C) &= \max\{|\sup(C,a_i)|, |\inf(C,a_i)|\} \end{aligned}$$

$$\begin{aligned} \Delta_{\texttt{max}_{a_i}}(C) &= |\sup(C,a_i) - \inf(C,a_i)| \\ \Delta_{\texttt{min}_{a_i}}(C) &= |\sup(C,a_i) - \inf(C,a_i)| \\ \Delta_{\texttt{avg}_{a_i}}(C) &= \frac{|\sup(C,a_i) - \inf(C,a_i)|}{2} \end{aligned}$$

8.2 Exploiting the constraint system

The sensitivity of aggregation functions, as shown above, depends on the range of the values of an attribute, so clearly it is important to compute the range as accurately as possible.

The usual approach is to consider the bounds given by the domain of each attribute. In terms of constraint system, this corresponds to consider the solutions of the constraint $C_I = a_1 \in D_1 \wedge a_2 \in D_2 \wedge \ldots \wedge a_n \in D_n$. I.e. the standard approach computes the sensitivity of aggregation functions for an attribute a on the basis of $\sup(C_I, a)$ and $\inf(C_I, a)$.
In our proposal we also use C_I: for us it is the initial constraint, at the beginning of the analysis of the query. The difference is that our approach updates this constraints with information provided by the various components of the query, and then exploits this information to compute more accurate ranges for each attribute. The following example illustrates the idea.

Example 3. *Assume that* $schema(R) = (\{Weight, Height\}, C_I)$*, and that the domain for* $Weight$ *is* $[0,150]$ *and for* $Height$ *is* $[0,200]$*. The following query asks the average weight of all the individuals whose weight is below the height minus* 100*.*

$$\gamma_{avg(Weight)}(\sigma_{Weight \leq Height - 100}(R))$$

Below we show the initial constraint C_I *and the constraints* C_Q *computed by taking into account the condition of* σ*. Compare the sensitivity computed using* C_I *with the one computed using* C_Q*: They differ because in* C_Q *the max value of* $Weight$ *is* 100*, while in* C_I *is* 150*.*

$$C_I = \{W \in [0,150] \wedge H \in [0,200]\} \qquad \Delta(C_I, \gamma_{avg(W)}) = \frac{|max(C_I,W) - min(C_I,W)|}{2} = 75$$
$$C_Q = \{W \in [0,150] \wedge H \in [0,200] \wedge W \leq H - 100\} \qquad \Delta(C_Q, \gamma_{avg(W)}) = \frac{|max(C_Q,W) - min(C_Q,W)|}{2} = 50$$

Hence exploiting the constraints generated by the query can lead to a significant reduction of the sensitivity.

8.3 Constraints generated by the functions

We now define how to add new constraints for the newly created attributes computed by the functions.

Definition 20 (Constraints for functions). *Let* $schema(R) = (A, C)$*,* $A' \subseteq A$ *and* $F = \{f_1(a_1), \ldots, f_n(a_n)\}$*, where* $a_1, \ldots, a_n \in A$*. Then* $schema({}_{A'}\gamma_F(R)) = (A' \cup \{a_{f_1} \ldots a_{f_n}\},\ C \wedge c_{f_1} \wedge \ldots \wedge c_{f_n})$*, where:*

$$c_{f_i} = \begin{cases} min(C,a_i) \leq \ a_{f_i} \ \leq sup(C,a_i) & \text{if } f_i = max/min/avg \\ 0 \leq \ a_{f_i} & \text{if } f_i = sum/count \end{cases}$$

9 Global sensitivity

We have concluded the analysis for all operators of relational algebra, and we now define the sensitivity of the whole query in a compositional way.
For the computation of the sensitivity, we need to take into account the constraint generated by it. We start by showing how to compute this constraint, in the obvious (compositional) way. Remember that we have already defined the constraints generated by each relational algebra operator in Sections 6, 7 and 8.

Definition 21 (Constraint generated by an intermediate query). *The global constraint generated by an intermediate query* Q *on relations with relational schema* $r(a_1 : D_1, a_2, D_2, \ldots, a_n : D_n)$ *is defined statically as:*

$$C_Q = schema(Q(R))$$

where R *is any relation such that* $schema(R) = (\{a_1, a_2, \ldots, a_n\}, C_I)$*, with* $C_I = a_1 \in D_1 \wedge a_2 \in D_2 \wedge \ldots \wedge a_n \in D_n$*.*

We assume, the top-level operator in a query is an aggregation γ_f, followed by a query composed freely using the other operators. We now show how to compute the sensitivity of the latter. Since it is a recursive definition, for the sake of elegance we will assume an identity query Id.

Definition 22 (Intermediate query sensitivity). *Assume* $\mathtt{op} : (\mathcal{R}^n, d_{nH}) \to (\mathcal{R}, d_H)$ *and* $C_{\mathtt{op}}$ *the constraint obtained* after *the application of* op*:*

$$\begin{array}{lcll} S(Id) & = & \min(1, diam(C_{Id})) & \textit{base case} \\ S(\mathtt{op} \circ Q) & = & \min\left(\Delta_{\mathtt{op}} \cdot S(Q), diam(C_{\mathtt{op} \circ Q})\right) & \textit{if } n = 1 \\ S(\mathtt{op} \circ (Q_1, Q_2)) & = & \min\left(\Delta_{\mathtt{op}} \cdot \max(S(Q_1), S(Q_2)), diam(C_{\mathtt{op} \circ (Q_1, Q_2)})\right) & \textit{if } n = 2 \end{array}$$

where op *can be any of* $\cup, \cap, \setminus, \sigma, \pi, \times, \times_1$ *and the (classic)* ${}_A\gamma_F$.

We are now ready to define the global sensitivity of the query:

Definition 23 (global sensitivity). *The global sensitivity GS of a query* $\gamma_f(Q)$ *is defined as:*

$$GS(\gamma_f(Q)) \quad = \quad \begin{cases} \Delta_f(C_Q) \cdot S(Q) & \textit{if } f = \mathtt{count}, \mathtt{sum}, \mathtt{avg} \\ \Delta_f(C_Q) & \textit{if } f = \mathtt{max}, \mathtt{min} \end{cases}$$

The following theorem, (proof in full version [3]), expresses the soundness and the strictness of the bound computed with our method.

Theorem 1 (Soundness and strictness). *The sensitivity bound computed by* $GS(\cdot)$ *is sound and strict. Namely:*

$$GS(\gamma_f(Q)) \; = \; \Delta_{\gamma_f(Q)}$$

10 Related Work

The field of privacy in statistical databases has often been characterized by ad-hoc solutions or algorithms to solve specific cases [7]. In recent years however there have been several efforts to develop a general framework to define differentially private mechanisms. In the work [12] the authors have proposed a functional query language equipped with a type system that guarantees differential privacy. Their approach is very elegant, and based on deep logical principles. However, it may be a bit far from the practices of the database community, addressing which is the aim of our paper.
The work that is closest to ours, is the PINQ framework [11], where McSherry extends the LINQ language, with differential privacy functionalities developed by himself, Dwork and others in [10].
Despite this existing implementation we felt the need for a more universal language to explore our ideas, and the mathematically-based framework of relational algebra seemed a natural choice. Furthermore the use of a constraint system to increase the precision of the sensitivity bound was, to out knowledge, never explored before.

11 Conclusions and future work

We showed how a classical language like relational algebra can be a suitable framework for differential privacy and how technology already in place, like check constraints, can be exploited to improve the precision of our sensitivity bounds.

Our analysis showed how the most common operation on databases, the join $\bowtie$, poses great privacy problems and in future we hope to develop solutions to this issue, possibly along the lines already presented in Section 7.3.
In this paper we have considered only the sensitivity, that is the effect on distances of operators, while another interesting aspect would be to compute the effect on the ε exponent as explored in [11], and possibly propose convenient strategies to query as much as possible over disjoint data sets.

References

[1] Krzysztof R. Apt (2003): *Principles of constraint programming*. Cambridge University Press.

[2] Alan Beaulieu (2009): *Learning SQL - Master SQL Fundamentals (2. ed.)*. O'Reilly.

[3] Marco Stronati Catuscia Palamidessi (2012): *Differential Privacy for Relational Algebra: improving the sensitivity bound via constraints systems.* Available at `http://www.lix.polytechnique.fr/~stronati/papers/qapl12.pdf`. Full Version.

[4] E. F. Codd (1970): *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM 13(6), pp. 377–387.

[5] E. F. Codd (1972): *Relational Completeness of Database Sublanguages*. In R. Rustin, editor: Data Base Systems, Prentice-Hall, New Jersey.

[6] Cynthia Dwork (2006): *Differential Privacy*. In: Proc. of ICALP, LNCS, 4052, Springer, pp. 1–12.

[7] Cynthia Dwork (2008): *Differential Privacy: A Survey of Results.* In: TAMC, pp. 1–19.

[8] Cynthia Dwork (2011): *A Firm Foundation for Private Data Analysis*. Communications of the ACM 54(1), pp. 86–96.

[9] Cynthia Dwork & Jing Lei (2009): *Differential Privacy and Robust Statistics*. In: Proc. of STOC, ACM, pp. 371–380.

[10] Cynthia Dwork, Frank McSherry, Kobbi Nissim & Adam Smith (2006): *Calibrating Noise to Sensitivity in Private Data Analysis.* In: TCC, pp. 265–284.

[11] Frank McSherry (2009): *Privacy integrated queries: an extensible platform for privacy-preserving data analysis.* In: SIGMOD Conference, pp. 19–30.

[12] Jason Reed & Benjamin C. Pierce (2010): *Distance makes the types grow stronger: a calculus for differential privacy.* In: ICFP, pp. 157–168.

Hybrid performance modelling of opportunistic networks

Luca Bortolussi[1] Vashti Galpin[2] Jane Hillston[2]
[1] Department of Mathematics and Computer Science, University of Trieste
[2] Laboratory for Foundations of Computer Science, School of Informatics, University of Edinburgh
luca@dmi.units.it, Vashti.Galpin@ed.ac.uk, Jane.Hillston@ed.ac.uk

We demonstrate the modelling of opportunistic networks using the process algebra stochastic HYPE. Network traffic is modelled as continuous flows, contact between nodes in the network is modelled stochastically, and instantaneous decisions are modelled as discrete events. Our model describes a network of stationary video sensors with a mobile ferry which collects data from the sensors and delivers it to the base station. We consider different mobility models and different buffer sizes for the ferries. This case study illustrates the flexibility and expressive power of stochastic HYPE. We also discuss the software that enables us to describe stochastic HYPE models and simulate them.

1 Introduction

Hybrid behaviour can arise in widely varying different contexts, both engineered and natural, pure and abstracted. Such systems have elements which are subject to continuous change, interleaved with discrete events which may change the elements themselves as well as their mode of evolution. The continuous aspect of the behaviour may be *pure* in that it is a physical entity which has continuous values, such as temperature or pressure, or may be *abstracted* as an approximation of a discrete quantity such as concentrations of biochemical species within a cell. Thus examples of hybrid systems include thermostatically controlled heating systems and genetic regulatory networks, such as the repressilator [13, 18].

In this paper we consider an engineered system with abstract continuity: an *opportunistic* network [22, 32]. In such a network, nodes experience periods of disconnectedness, during which they nevertheless may accumulate traffic in the form of packets. Sporadic connectivity is provided by occasional proximity of other nodes. Such connectivity is then exploited to further the progress of packets towards their destination (hence the term *opportunistic*). There are many interesting questions about performance and capacity planning for such networks, but a detailed discrete state representation in which all packets are treated individually can rapidly exceed feasible analysis and can also be expensive in terms of time. Instead, here, we abstract the traffic to be a fluid quantity rather than discrete packets and model the system as a stochastic hybrid system.

Process algebras have a long-established history of use for compositional modelling and analysis of systems with concurrent behaviour. Moreover, when extended with stochastic variables to represent duration and relative probability of events, they have been successfully applied to performance modelling and other forms of quantified analysis. HYPE is one of several recently defined process algebras which extend this capability to the modelling of hybrid systems [28, 17]. In HYPE the focus is on a fine-grained compositionality, in which all the *influences* or *flows* which impact the continuous variables in the system are modelled explicitly. Importantly, addressing the modelling in this style removes the need to include ordinary differential equations (ODEs) governing the continuous evolution of such variables in the syntax of the model. Instead the dynamic behaviour emerges, via the semantics of the language, when the components are composed. Moreover, this fine-grained approach gives the language more expressiveness, than say, hybrid automata, as recently demonstrated in [16]. Furthermore, the use of

Preliminary Report. Final version to appear in:
QAPL 2012

flows as the basic elements of model construction has advantages such as ease and simplification of modelling. This approach assists the modeller by allowing them to identify smaller or local descriptions of the model and then to combine these descriptions to obtain the larger system.

In the original definition of HYPE, discrete actions were termed *events* and these were always considered to be *instantaneous* although they could be subject to an activation condition determining just when the instantaneous jump would occur. A distinction was made between *urgent* and *non-urgent* events. Most activation conditions are expressed in terms of conditions on the evolving values of continuous variables and urgent events are triggered immediately when such conditions become true. In contrast, non-urgent events were not tied to the continuous state of the system (denoted by the undefined activation condition $\perp$) and could occur randomly at some unspecified time in the future. A recent extension of HYPE [7] refined this notion of non-urgent events, by introducing *stochastic events*. These events have an activation condition that is a random variable, capturing the probability distribution of the time until the event occurs. Thus these events still occur non-deterministically, but they are now quantified and so the models admit quantitative analysis. In order to carried out this type of analysis, a novel software tool has been developed which can simulate stochastic HYPE models.

This extended HYPE is ideal for modelling opportunistic networks in which we wish to study the emergent properties when nodes establish contact intermittently, but according to some probability distribution. In other words, we have some expectation of the frequency with which connections are formed, rather than admitting the possibility that this can be indefinitely postponed, as would be the case with non-urgent events. This is more realistic since the intermittent connectivity is usually provided by nodes embedded in vehicles which make regular visits.

The case study presented in this paper consists of stationary nodes that record multimedia data, specifically video, and a mobile node on a vehicle that collects the data and delivers it to a stationary base station. Since a characteristic of multimedia is very high data volume, this scenario is particularly appropriate for the fluid approach that we take here. We are specifically interested in two parameters of the model: how much data storage is required for the ferry and how often should it interact with the video sensors.

The rest of this paper is organised as follows. In Section 2 we briefly recall the basic notions of stochastic HYPE by means of a running example. This includes an account of the novel software tool which has been developed to simulate stochastic HYPE models. Next, in Section 2.1 we give a more detailed description of opportunistic networks, and the particular system we are considering. Section 4 presents a general framework for describing opportunistic networks, presents the stochastic HYPE model of the case study and the results of its analysis. Finally, Sections 5 and 6 discuss related work, future research and draw final conclusions.

2 Stochastic HYPE

In this section we present the definition of stochastic HYPE [7] and introduce a small example to illustrate the definition. More details about the language can be found in [7, 17, 16]. We consider a basic model of a network node with a buffer, which can receive packets from an input channel and send packets to an output channel. We assume that the number of packets that travel through the node and that are stored in the buffer is large, hence we describe them as a fluid quantity. Received packets are stored in the buffer, waiting to be sent. We allow reception and sending of packets to happen concurrently, but it is equally simple to enforce a mutually exclusive send/receive policy. We also assume that uplinks and downlinks are not always working, but they are activated and deactivated depending on the availability

$$Buffer \stackrel{def}{=} Sys \underset{M}{\bowtie} \underline{\text{init}}.Con \quad \text{with} \quad M = \{\underline{\text{init}}, \overline{\text{on}}_{in}, \overline{\overline{\text{off}}}_{in}, \overline{\text{on}}_{out}, \overline{\overline{\text{off}}}_{out}, \underline{\text{empty}}, \underline{\text{full}}\}.$$

$$Sys \stackrel{def}{=} Input \underset{\{\underline{\text{init}}\}}{\bowtie} Output$$

$$\begin{aligned} Input &\stackrel{def}{=} \overline{\text{on}}_{in}{:}(in, r_{in}, const).Input + \overline{\overline{\text{off}}}_{in}{:}(in, 0, const).Input + \\ &\quad \underline{\text{full}}{:}(in, 0, const).Input + \underline{\text{init}}{:}(in, 0, const).Input \\ Output &\stackrel{def}{=} \overline{\text{on}}_{out}{:}(out, -r_{out}, const).Output + \overline{\overline{\text{off}}}_{out}{:}(out, 0, const).Output + \\ &\quad \underline{\text{empty}}{:}(out, 0, const).Output + \underline{\text{init}}{:}(out, 0, const).Output \end{aligned}$$

$$\begin{aligned} Con &\stackrel{def}{=} Con_{in} \underset{\emptyset}{\bowtie} Con_{out} \\ Con_{in} &\stackrel{def}{=} \overline{\text{on}}_{in}.Con'_{in} \quad Con'_{in} \stackrel{def}{=} \overline{\overline{\text{off}}}_{in}.Con_{in} + \underline{\text{full}}.Con_{in} \\ Con_{out} &\stackrel{def}{=} \overline{\text{on}}_{out}.Con'_{out} \quad Con'_{out} \stackrel{def}{=} \overline{\overline{\text{off}}}_{out}.Con_{out} + \underline{\text{empty}}.Con_{out} \end{aligned}$$

$$iv(in) = B \qquad iv(out) = B$$

$$\begin{aligned} ec(\underline{\text{init}}) &= (true, B' = b_0) \\ ec(\overline{\text{on}}_{in}) &= (k^{on}_{in}, true) & ec(\overline{\overline{\text{off}}}_{in}) &= (k^{off}_{in}, true) \\ ec(\overline{\text{on}}_{out}) &= (k^{on}_{out}, true) & ec(\overline{\overline{\text{off}}}_{out}) &= (k^{off}_{out}, true) \\ ec(\underline{\text{full}}) &= (B = max_B, true) & ec(\underline{\text{empty}}) &= (B = 0, true) \end{aligned}$$

Figure 1: Simple network node model in stochastic HYPE.

of a connection. These events are described as stochastic, with firing times governed by exponential distributions. Finally, incoming traffic has to be stopped if the buffer becomes full and outgoing traffic has to be stopped when the buffer is empty.

HYPE modelling is centred around the notion of *flow*, which is intended here as some sort of influence continuously modifying one variable. Both the strength and form of a flow can be changed by *events*. In our example, there are two flows modifying the buffer level, modelled by the continuous variable B, namely reception and sending of packets. Flows are described by the *uncontrolled system*, a composition of several sequential subcomponents, each modelling how a specific flow is changed by events. For instance, in Figure 1, the subcomponent *Input* describes the inflow of packets in the buffer. This subcomponent reacts to four events: $\overline{\text{on}}_{in}$ and $\overline{\overline{\text{off}}}_{in}$, modelling the activation and deactivation of the uplink; $\underline{\text{full}}$, modelling the suspension of incoming traffic due to the buffer becoming full; and $\underline{\text{init}}$, the first event that sets the initial value of the influence. The tuple $(in, r_{in}, const)$ following event $\overline{\text{on}}_{in}$, is called an *activity* or an *influence* and describes how the input affects the buffer level when it is in effect: in is the name of the influence, which provides a link to the target variable of the flow (B in our example), r_{in} is the strength of the influence and $const$ is the influence type, identifying the functional form of the flow (which is specified separately by the interpretation $[\![const]\!] = 1$). When the input is switched off, the influence $(in, r_{in}, const)$ is replaced by $(in, 0, const)$ i.e. the influence strength of the input becomes zero. The other subcomponent affecting buffer level is the output component, modelling the sending of packets. States of a HYPE model are collections of influences, one for each influence name, defining a set of ordinary differential equations describing the continuous evolution of the system. For instance, $(in, r_{in}, const)$ contributes to the ODE of B with the summand $r_{in}[\![const]\!] = r_{in}$.

The controller *Con*, instead, is used to impose causality on events, reflecting natural constraints or design choices. For instance, Con_{in} models the fact that the reception of packets can be turned off only after being turned on. Furthemore, it describes termination of the input if the buffer becomes full, but only if the uplink is active.

Events in stochastic HYPE are of two kinds, either *stochastic* or *deterministic*. Deterministic events $\underline{\mathrm{a}} \in \mathscr{E}_d$ happen when certain conditions are met by the system. These *event conditions* are specified by a function *ec*, assigning to each event a *guard* or *activation condition* (a boolean predicate depending on system variable, stating when a transition can fire) and a *reset* (specifying how variables are modified by the event). For example, $ec(\underline{\mathrm{full}}) = (B = max_B, true)$ states that the uplink is shut down when the buffer reaches its maximum capacity max_B, and no variable is modified. If we wanted to model a policy throwing away a fraction ρ of the packets when the buffer becomes full, then we could have defined $ec(\underline{\mathrm{full}}) = (B = max_B, B' = (1-\rho)B)$. Deterministic events in HYPE are *urgent*, meaning that they fire as soon as their guard becomes true.

Stochastic events $\overline{\mathrm{a}} \in \mathscr{E}_s$ have an event condition composed of a stochastic rate (replacing the guard of deterministic events) and a reset. For instance, $ec(\overline{\mathrm{on_{in}}}) = (k_{in}^{on}, true)$ states that the reception of packets is a stochastic event happening at times exponentially distributed with constant rate k_{in}^{on}. In general, rates define exponential distributions and can be functions of the variables of the system.

For completeness, we provide the formal definition of the syntax of stochastic HYPE.

Definition 1 *A* stochastic HYPE model *is a tuple* $(ConSys, \mathscr{V}, IN, IT, \mathscr{E}_d, \mathscr{E}_s, \mathscr{A}, ec, iv, EC, ID)$ *where*

- *ConSys is a controlled system as defined below.*
- $\mathscr{V}$ *is a finite set of variables.*
- *IN is a set of influence names and IT is a set of influence type names.*
- $\mathscr{E}_d$ *is the set of instantaneous events of the form* $\underline{\mathrm{a}}$ *and* $\underline{\mathrm{a}}_i$*.*
- $\mathscr{E}_s$ *is the set of stochastic events of the form* $\overline{\mathrm{a}}$ *and* $\overline{\mathrm{a}}_i$*.*
- $\mathscr{A}$ *is a set of activities of the form* $\alpha(\mathscr{W}) = (\iota, r, I(\mathscr{W})) \in (IN \times \mathbb{R} \times IT)$ *where* $\mathscr{W} \subseteq \mathscr{V}$*.*
- $ec : \mathscr{E} \to EC$ *maps events to event conditions. Event conditions are pairs of activation conditions and resets. Resets are formulae with free variables in* $\mathscr{V} \cup \mathscr{V}'$*. Activation conditions for instantaneous events* $\mathscr{E}_d$ *are formulas with free variables in* $\mathscr{V}$ *and the second, while for stochastic events of* $\mathscr{E}_s$*, they are functions* $f : \mathbb{R}^{|\mathscr{V}|} \to \mathbb{R}^+$*.*
- $iv : IN \to \mathscr{V}$ *maps influence names to variable names.*
- *EC is a set of event conditions.*
- *ID is a collection of definitions consisting of a real-valued function for each influence type name* $[\![I(\mathscr{W})]\!] = f(\mathscr{W})$ *where the variables in* $\mathscr{W}$ *are from* $\mathscr{V}$*.*
- $\mathscr{E}$*,* $\mathscr{A}$*, IN and IT are pairwise disjoint.*

Definition 2 *A* controlled system *is constructed as follows.*

- Subcomponents *are defined by* $C_s(\mathscr{W}) = S$*, where* C_s *is the* subcomponent name *and S satisfies the grammar* $S' ::= \mathrm{a} : \alpha.C_s \mid S' + S'$ *(*$\mathrm{a} \in \mathscr{E} = \mathscr{E}_d \cup \mathscr{E}_s$*,* $\alpha \in \mathscr{A}$*), with the free variables of S in* $\mathscr{W}$*.*
- Components *are defined by* $C(\mathscr{W}) = P$*, where C is the* component name *and P satisfies the grammar* $P' ::= C_s(\mathscr{W}) \mid C(\mathscr{W}) \mid P' \underset{L}{\bowtie} P'$*, with the free variables of P in* $\mathscr{W}$ *and* $L \subseteq \mathscr{E}$*.*

- *An* uncontrolled system Σ *is defined according to the grammar* $\Sigma' ::= C_s(\mathscr{W}) \mid C(\mathscr{W}) \mid \Sigma' \underset{L}{\bowtie} \Sigma'$, *where* $L \subseteq \mathscr{E}$ *and* $\mathscr{W} \subseteq \mathscr{V}$.
- Controllers *only have events:* $M ::= \underline{a}.M \mid 0 \mid M + M$ *with* $\underline{a} \in \mathscr{E}$ *and* $L \subseteq \mathscr{E}$ *and* $Con ::= M \mid Con \underset{L}{\bowtie} Con$.
- *A* controlled system *is* $ConSys ::= \Sigma \underset{L}{\bowtie} \underline{\text{init}}.Con$ *where* $L \subseteq \mathscr{E}$. *The set of controlled systems is* $\mathscr{C}_{Sys}$.

The semantics of HYPE has been defined in [17, 16], where a mapping to Hybrid Automata [20] is also discussed. The semantics of stochastic HYPE [7], instead, is given in terms of TDSHA (Transition-Driven Stochastic Hybrid Automata, [8]), which are a high level representation of PDMPs (Piecewise Deterministic Markov Processes, [11]). PDMPs are stochastic hybrid processes which interleave a deterministic evolution, described by a set of differential equations (depending on the current discrete mode of the system), with discrete jumps, which can be of two types: *spontaneous*, happening at exponentially distributed random times, and *forced*, happening when specific conditions on system variables are met. Both kind of events can reset the state of the system according to a specified reset policy. Intuitively, the dynamics of a stochastic HYPE model is as follows: the system variables will evolve following the solution of a set of ODE, defined by the influences active in the system, one for each influence name. The events that can happen, instead, are determined by the current state of the controller. Active stochastic events happen at random times, while deterministic events happen when their guard becomes true. In both cases, the reset policy of the event is applied. Moreover, the state of the controller and the set of active influences is updated according to model structure. In particular, all influences preceded in subcomponents by the event that occurred will replace the ones with the corresponding name, so that the continuous dynamics can have different modes of operation.

All HYPE models that will be considered in the paper comply with the definition of well-defined HYPE models, given in [16]. Essentially, each subcomponent must be a self-looping agent of the form $S = \sum_{i=1}^{k} \mathrm{a}_i{:}\alpha_i.S + \underline{\text{init}}{:}\alpha.S$, with each α_i of the form (i_S, r_i, I_i), where i_S is an influence name appearing only in subcomponent S. Furthermore, synchronization must involve all shared events.

2.1 Simulation software

In this section, we provide some details about the implementation of stochastic HYPE that we used to analyse the model of an opportunistic network presented in the paper. This software tool supports an automatic extraction of basic statistics from a set of stochastic runs, and has plotting facilities (including 3D plots and distribution histograms) and data export facilities. Furthermore, it supports automatic exploration of the parameter space. The user interface is command-line-based, and uses a simple script language to instruct the software.

The basic idea behind the implementation, done in Java and available on request from the authors, is to flatten a HYPE model into a representation in which additional *discrete* variables (i.e., variables that can take only a finite set of values) are introduced to keep track of the current active influences and the current states of the controller. The number of variables required for this encoding is easily seen to be linear in the size of the system, as it requires an additional number of variables equal to the number of different subcomponents plus the number of different states of the controller. Furthermore, this approach has the advantage of avoiding an explicit construction of all the modes of the (stochastic) hybrid automaton associated with a HYPE model. This is possible since modes of such an automaton are uniquely identified by the values of the discrete variables introduced.

```
hype model network_node

#definitions
var B = 0;              //buffer size
param maxB = 100;       //buffer capacity
param r_in = 1;         //input rate
param r_out = -2;       //output rate
param kon_in = 0.5;     //uplink activation rate
param koff_in = 0.05;   //uplink deactivation rate
param kon_out = 0.02;   //downlink activation rate
param koff_out = 0.01;  //downlink deactivation rate

function const() = 1;        //constant function
guard above(X,K) = X >= K;   //X >= K
guard below(X,K) = X <= K;   //X <= K

#mappings
infl in :-> B;       //input influence
infl out :-> B;      //output influence
event on_in   =  :-> @ kon_in;         //input activation
event off_in  =  :-> @ koff_in;        //input deactivation
event on_out  =  :-> @ kon_out;        //ouput activation
event off_out =  :-> @ koff_out;       //output deactivation
event full    =  above(B,maxB) :-> ;   //buffer full
event empty   =  below(B,0)    :-> ;   //buffer empty

#subcomponents
//template to define a switch between two states
switch(on,off,block,r) := off,block,init:[0,const()] + on:[r,const()];

#components
input  := switch(on_in,off_in,full,r_in):in;       //input component
output := switch(on_out,off_out,empty,r_out):out;  //output component
sys := input <*> output;                           //uncontrolled system

#controller
con_in := on_in.con_in_1; con_in_1 := off_in.con_in + full.con_in;         //input controller
con_out := on_out.con_out_1; con_out_1 := off_out.con_out + empty.con_out; //output controller
con := con_in || con_out;                                                  //system controller

#system
sys <*> con;     //system
```

Figure 2: Code for the example given in Figure 1.

To illustrate how the encoding works in terms of the generated ODEs, consider the *Input* component of Section 2. To model which influence is active between $\alpha_1 = (in, r_{in}, const)$ and $\alpha_0 = (in, 0, const)$, we need a new variable, I_{Input}, taking values in $\{0,1\}$. If I_{Input} equals zero, the active influence is α_0, otherwise the active influence is α_1. This means that the component of the ODEs associated with variable B generated by *Input* is of the form $r_{in}[\![const]\!]\langle I_{Input} = 1\rangle + 0[\![const]\!]\langle I_{Input} = 0\rangle = r_{in}\langle I_{Input} = 1\rangle$, where $\langle\cdot\rangle$ denotes the logical value of a boolean predicate expressed as 0 or 1. Resets and guards of events are modified in order to correctly update the discrete variables introduced. For instance, the reset of event full resets I_{Input} to 0.

The tool provides simulation of stochastic and non-stochastic HYPE models, and uses the Java mathematical library MathCommons [1] to numerically integrate the ODEs, exploiting its embedded event detection system to manage the firing of events. In particular, stochastic simulation is dealt with in the following way [40, 8]. Consider a stochastic event with rate $\lambda(t)$, depending on time via the continuously evolving variables of the system. We compute its cumulative rate $\Lambda(t_0,t) = \int_{t_0}^{t} \lambda(s)ds$ by coupling

with the ODE system, the following equation for Λ: $\frac{d\Lambda(t_0,t)}{dt} = \lambda(t)$, with $\Lambda(t_0,t_0) = 0$. Then, we fire the stochastic transition as soon as $\Lambda(t_0,t) = -\ln(U)$, where $U \sim Unif(0,1)$ is a uniformly distributed random number in $[0,1]$, sampled using the pseudo-number generator of MathCommons library. Notice that if the rate $\lambda(t) = \lambda$ is constant, then the firing time is $-\frac{1}{\lambda}\ln(U)$, and hence we have the standard Monte Carlo inversion method to simulate exponentially distributed random variables [40].

We conclude this section with some details of the language supported by the tool to model with HYPE. Each HYPE model consists of 6 sections. The `#definitions` section contains the definition of system variables, parameters, expressions shorthands, user-defined functions (which replace influence types in the tool) and boolean predicates. The `#mappings` section is devoted to the definition of influences (mapping them to variables) and events (specifying their name, guard, reset, and, for stochastic events, rate). In the tool, stochastic events can be guarded. This is rendered in HYPE using suitable discontinuous rate functions. The `#subcomponent` section contains the description of subcomponents, which can be parameterised (with respect to variables and events) in order to reuse the same definition more than once. In particular, the user can assign more than one event to the same influence and the influence name is assigned to the whole subcomponent, in order to comply to the restrictions of well-defined HYPE models. The `#component` section, instead, contains the (parametric) description of system components, including the uncontrolled system. The `#controller` section contains the definition of sequential and compound controllers. Finally, the `#system` section specifies the complete system by combining a controller and an uncontrolled system. The code for the example of Figure 1 is given in Figure 2. We would like to add additional parameterisation abilities to the software to support systems where we define many similar components. We address this point further in Section 6.

3 Opportunistic networks

Since stochastic HYPE allows for the modelling of discrete quantities in a fluid manner, it is suitable for network modelling. Furthermore, it has stochastic aspects that model randomness, making it appropriate to model the disconnectedness that can happen in opportunistic networks. Networks are opportunistic when nodes can communicate even though there may never be a direct path between them. They use a *store-carry-forward* approach and decisions about routing are determined dynamically with policies based on the notion of getting a packet closer to its final destination [32]. Delays may occur but networks of this type can be deployed in environments where disconnectedness is possible but increased time for packet delivery is acceptable. The major challenges in such networks [32, 22, 35] include the following.

Disconnectedness: A direct path may occur very infrequently or never between any two nodes in the network.

High latency and low data rate: Due to disconnectedness, there can be significant delays for an individual packet, including those caused by queueing at an intermediate node. This obviously can result in low throughput.

Limited resources: Nodes are often battery-driven and hence need to conserve energy to lengthen their lifetimes. This means that the amount of storage space or strength and length of radio usage for communication are limited. Additionally, nodes may become permanently disabled due to a hostile environment.

Various protocols have been designed to mitigate the problems caused by these challenges and these protocols have specific objectives [35]. The main objective is to maximise the probability of a packet reaching its destination. Ideally, at the same time, both the delivery delay and the resource usage should

be minimised. Storage capacity at nodes should be sufficient, both to cope with the inherent latency and in certain cases, to store copies of messages that could be lost.

There are different ways to categorise routing/forwarding protocols: deterministic versus stochastic [42], with or without infrastructure [32] or most commonly, flooding versus forwarding [22, 35]. In flooding protocols, packets are forwarded to many nodes. Variations include epidemic routing which is based on a model of disease transmission, where nodes that have "recovered" do not forward packets they have already seen [39]. Additional conditions can be added to epidemic routing to reduce resource usage. Other flooding approaches involve the estimation of the probability of delivery by a node, and this is used in deciding which nodes packets should be forwarded to. An example is PROPHET [30].

In forwarding protocols, a single copy of a packet moves through the network. Decisions about which node is the best node to move to can be done by location (how close the node is to the destination node), knowledge about the network provided by oracles [23] or other characteristics of the network that can be obtained by a node through interaction with other nodes. For example, the MaxProp protocol is based on historical data of path likelihood [9].

In the context of our case study, two wildlife monitoring projects are of interest: ZebraNet [25] and SWIM [36]. In the first case, zebra are fitted with collars and the data is collected by a mobile node on a vehicle that moves around the area. In second case, whales are tagged with sensors. In both cases, flooding can be used as the protocol. In flooding, whenever two animals are in sufficiently close proximity, they exchange data. In this way, assuming one animal comes close enough to the mobile nodes or base stations, there is sufficient proximity between animals, and no data buffers become full, all data will arrive at the base station. In the case of ZebraNet, the history-based protocol is also used. Here, each sensor keeps a value which gives an indication of when it last interacted with the mobile node. When deciding which neighbour sensor to send data to, the one with the highest value is chosen. This protocol has been shown to outperform the flooding protocol.

ZebraNet has similarities with carrier-based routing (which is classified by [32] as routing with mobile infrastructure). In these protocols [24, 43], particular mobile nodes which can be called carriers, supports, forwarders, mules or ferries, collect data from other, possibly stationary, nodes. In some protocols, only the ferries collect data and in others, non-ferry nodes exchange data as well. Our case study is based on the former.

3.1 Our case study

As an initial test for our modelling of opportunistic networks, we have chosen a relatively simple scenario. Since our fluid packet approach is most useful in cases where there are large amounts of data, we consider an example where video data is captured by stationary sensors. The idea is that they are motion activated with a low number of activations expected each day. Because these sensors are required to run off battery, it is not desirable for them to have powerful enough radio to share data over distance. Hence, a vehicular ferry moves around the area in which the video sensors are located and returns to a base station where the ferry delivers the data. This scenario could occur in a wildlife reserve or any scenario where video data is to be collected, but is not required in real-time. We do not assume that the primary purpose of the vehicle involved is to collect data from the nodes. It could be involved in supply delivery or game viewing, but we do assume some flexibility in routing as we consider later.

If we assume a fixed disk size for the video sensors, and no restriction on the amount of data that can be delivered to the base station, then the parameters of interest relate to the ferry and include buffer size, route taken and speed of movement. In the next section, we discuss how this can be modelled.

4 A framework for modelling opportunistic networks

The basic element of our model is the network node. To be able to model opportunistic networks in the most general way, we assume that each node has a fixed buffer capacity, and that it can keep track of multiple streams of data. These streams could represent data with different priorities, data with different destinations or different types of data; or combinations of priority, destination and type.

A node should be able to accept input; offer output; discard data from the buffer, either to free up space by dropping current data or to remove stale data, and generate data. Moreover, it should be able to keep track of the total data it has dropped, input or generated. It should also be able to make decisions about what data to input (how much, which and from whom), output (how much, which and to whom) and drop (how much and which). In certain instances, it may also need to make decisions about what data to generate.

To model this node in stochastic HYPE, we require variables to capture the current buffer level for each stream, together with the lifetime input and generated data for each stream, and lifetime dropped data for each stream. This gives us variables $Level_{i,v}$, $TotalIG_{i,v}$ and $TotalD_{i,v}$ for node i and stream v. Clearly, the value of the variables for everything except buffer levels will not decrease.

Each node has subcomponents for input, output, generation, drop and removal (where the second last term refers to discarding current data and the last term to discarding stale data). It also has two further subcomponents to keep track of data input and data generated. We have the following HYPE components for each node i and each stream v. Here, the symbol $\underset{*}{\bowtie}$ indicates synchronisation on all shared events.

$$Node_{i,v} \stackrel{def}{=} Input_{i,v} \underset{*}{\bowtie} Output_{i,v} \underset{*}{\bowtie} Generate_{i,v} \underset{*}{\bowtie} Remove_{i,v} \underset{*}{\bowtie} Drop_{i,v} \underset{*}{\bowtie} KeepI_{i,v} \underset{*}{\bowtie} KeepG_{i,v}$$

The influences that appear in $Input_{i,v}$, $Output_{i,v}$, $Generate_{i,v}$ and $Remove_{i,v}$ are mapped to the variable $Level_{i,j}$, those in $KeepI_{i,v}$ and $KeepG_{i,v}$ are mapped to the variable $TotalIG_{i,j}$ and that in $Drop_{i,v}$ to the variable $TotalD_{i,j}$.

Each node has a controller for the first five subcomponents. Controllers are not required for the last two subcomponents since the events for these subcomponents appear in the controllers for input and generation. Controllers are required for each stream as they may need to be treated separately, for example in the case of one stream being prioritised over another.

$$ConNode_{i,v} \stackrel{def}{=} ConI_{i,v} \underset{*}{\bowtie} ConO_{i,v} \underset{*}{\bowtie} ConG_{i,v} \underset{*}{\bowtie} ConR_{i,v} \underset{*}{\bowtie} ConD_{i,v}$$

The example in Section 2 provides a very simple version of such a node which only has input and output capability. The controllers must deal with aspects such as full and empty buffers, as well as switching between different functions, for example switching between input on and input off.

The next important aspect to model is the interaction between nodes in the network. Each possible connection between two nodes (some nodes may never have the ability to connect) has a controller that synchronises on proximity and brings up the link, takes down the link at the end of proximity or due to any other condition that could cause the link to end, and then does some housekeeping. Hence, we define $ConL_{i,j}$ the controller for the link between nodes i and j.

Then for each stream of data, there is a controller that involves events that determine whether data exchange should happen for that stream and whether the link should be uni- or bidirectional. The directionality of the link depends both on the characteristics of the modelled system and the policies used. This controller also includes a housekeeping subcontroller that synchronises with the housekeeping of the controller of the link between the two nodes.

$$ConStream_{i,j,v} \stackrel{def}{=} ConUni_{i,j,v} \underset{*}{\bowtie} ConUni_{j,i,v} \underset{*}{\bowtie} CBi_{i,j,v} \underset{*}{\bowtie} ConTidy_{i,j,v}$$

Finally, there are controllers which model the proximity of nodes. This is currently done abstractly, using a random variable to capture delays between connectedness. A recent paper describes the expected meeting time between nodes for various mobility models such as random direction, random waypoint and community-based (both for homogeneous and heterogeneous nodes) and suggest that these expected values can be used as rates to describe exponential distributions [37]. This allows us to model these specific types of mobility in this abstract fashion. However, as future work, we plan to develop more concrete models that describe movement in two-dimensional space, since HYPE can model this type of continuous behaviour in a straightforward way.

Hence, to construct an opportunistic network model in stochastic HYPE, we need to define a number of nodes based on our template, the appropriate connection controllers and proximity controllers, and most importantly, the policies and protocols that will be used in the network. Our longer-term goal is to develop a front end that will allow a user to specify nodes, connections, policies and protocols from which a HYPE model will be generated. This will provide a simulation tool for networks that should be faster than simulators that trace every packet, when the number of packets is large.

4.1 Our case study

For our specific case study, we used the basic node we have developed. Video sensor nodes require generation, discard and output capabilities; the ferry requires input and output, and the base station, input only. Possible links between nodes cover upload from sensors to the ferry and upload from the ferry to the base station. There are also two different proximity controllers: one allows the ferry to have a random route between nodes, and the other imposes a fixed route that is cyclically repeated.

In our case study, the policies are straightforward – unidirectional communication between a sensor and the ferry is set up only if the ferry is not currently communicating with another sensor. However, in a scenario with more than one ferry, it would be possible to implement a policy allowing a sensor node to choose which ferry to upload data to, making a decision based on certain ferry characteristics. Since we are dealing with a system where large amounts of data are generated, it makes sense to use this protocol rather than any other. In this scenario, it would be problematic to use flooding as the system is data bound, and hence an excessive transmission and storage of data cannot be recommended.

4.2 Results

The specifics of our case study are as follows: we assume that there are 10 video sensors and that the ferry only collects data during an 8 hour period, and we are interested both it what buffer size the ferry requires and how often there is contact between the ferry and a sensor (described by the mean-time-to-contact variable *MTC*), which is effectively the speed of the ferry.

Each video sensor has 250MB of disk space, and on average will record video three times a day for an average of 3 minutes each time; the video will require 10Mb for each minute. The upload speed from sensor to ferry is 1MB/s and the upload speed from ferry to base station is 30MB/s.

We consider 4 different scenarios. For *raer*, the ferry only returns to base at the end of the 8 hour period and has a random route. In the case of *raef*, the return to base is also at the end but there is a fixed route. For *rtbr* and *rtbf*, the ferry returns to base whenever it is full (incurring a penalty of extra distance to travel) and there are random and fixed routes, respectively. In the experiments to explore different values for mean time to contact, the ferry buffer size was set to 1000MB. In the experiments to consider different ferry buffer size the mean time to contact was set to 15 minutes. Each simulation took around 6.5 seconds on a standard laptop.

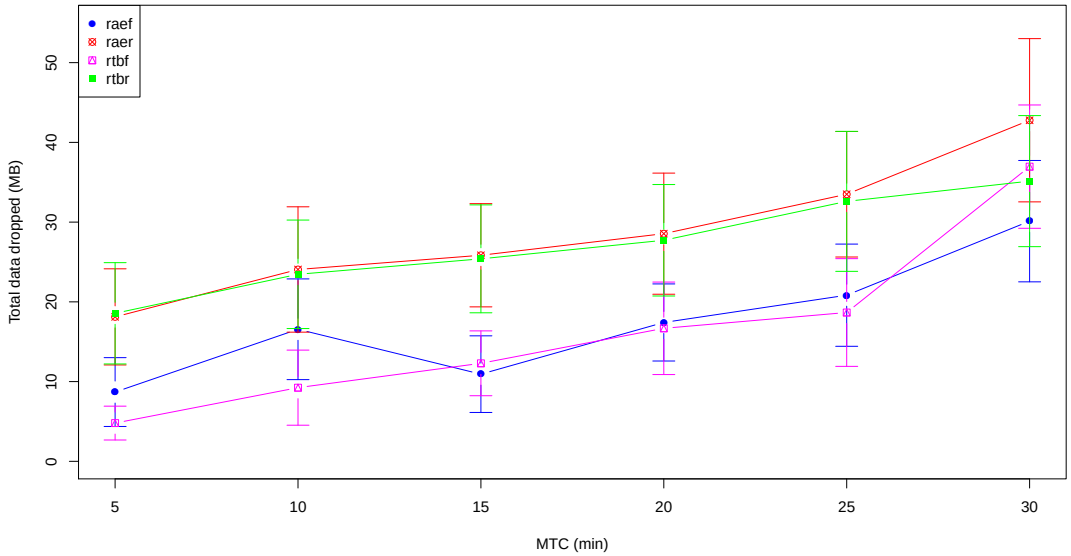


Figure 3: Average (200 simulation runs) data dropped for different mean time to contact (*MTC*) values. The bars delimit the 95% confidence interval. See the text for the description of the four different scenarios.

When we consider data dropped versus mean time to contact (see Figure 3), as mean time to contact increases (which means visits are less frequent) the amount of data dropped increases, which is what we would expect. In general, it can be seen that the fixed routes (*raef* amd *rtbf*) result in less data drop, which can be ascribed to fairness, in that each node get its turn whereas in the random case, it may not get a turn at all. Similarly, Figure 4 show that as frequency of visits decreases, the amount of data collected by the ferry decreases, and the fixed routes (*raef* amd *rtbf*) perform better in collecting more data.

For the ferry buffer size, we found that there appeared to be limited correlation between the different protocols and amount of data dropped, shown in Figure 5. By contrast, in Figure 6, the amount of data collected by the ferry, is understandably reduced for low buffer capacities in the case where the ferry does not return to base when full (*raer* and *raef*). At the higher buffer capacities, again the fixed routes out perform the random routes.

This case study illustrates how stochastic HYPE can be used to model these networks. It is relatively straightforward to add further nodes of all types, and hence model a larger system of the same type.

5 Related work

Other hybrid process algebras to describe hybrid systems include ACP_{hs}^{srt} [6], hybrid χ [4], ϕ-calculus [34] and HyPA [10]. The aspect of HYPE that distinguishes it from these process algebra, is that in HYPE the ODEs emerge from the semantics and are not required to be specified monolithically in the syntax because of the use of individual flows in HYPE. Additionally, unlike the process algebras mentioned and hybrid automata, it is possible to combine two HYPE models where a variable can be shared between models since it is possible to combine all the influences that apply to this shared variable. A more detailed comparison between HYPE and other hybrid modelling formalisms can be found in [17, 16]. To the best

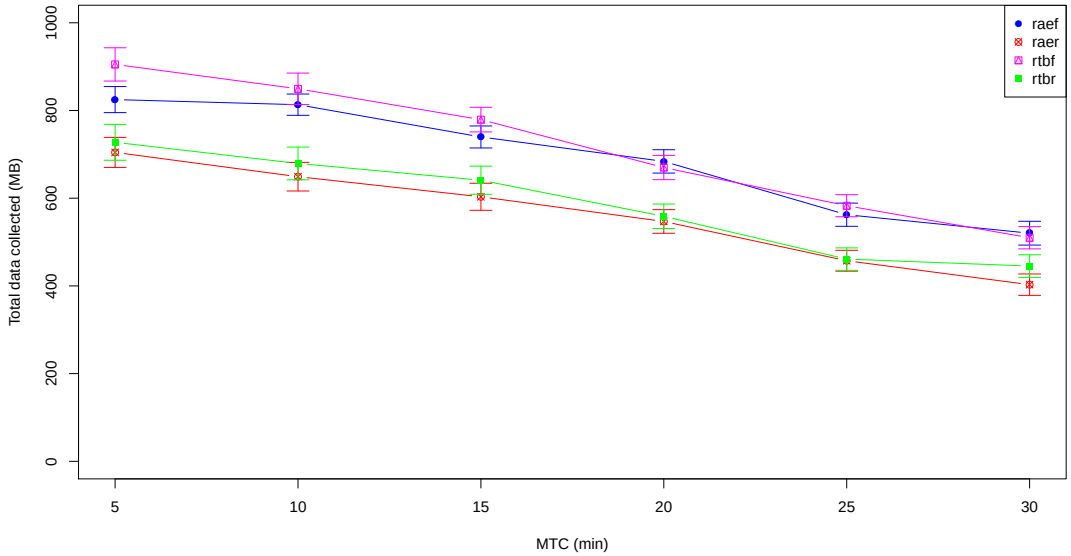


Figure 4: Average (200 simulation runs) data collected by ferry for different mean time to contact (*MTC*) values. The bars delimit the 95% confidence interval.

of our knowledge, no hybrid process algebra has previously been used to model networks using a fluid packet approach. Recently, an approach based on rewriting logic, Hybrid Interacting Maude, has been developed [14] but to date this research has focussed on thermal systems.

Other formal approaches to performance modelling of opportunistic networks have appeared in the literature in recent years. Much work has focussed on modelling the mobility patterns of nodes within the network, a feature that clearly has a strong impact on the performance that can be achieved. Examples include [41] and [37] in which the authors analyse the expected meeting time for various mobility models and bounds on delays. Other papers focus on the performance measures such as message delay and compare, as we do, the routing policies which may be applied. For example, Picu and Spyropoulos [33] use expensive Markov Chain Monte Carlo simulation to assess optimal relay selection for multicast communication in opportunistic networks, while [27] presents analytic bounds on message delivery capacity. In another example [31] considers the provisioning of a network in order to minimize the delay using an analytical model based on queueing theory. Their framework is more general than ours in the sense that services, rather than simply messages, are exchanged opportunistically between nodes and results as well as service requests are also exchanged. However, it should be possible to extend our modelling framework to encompass this richer scenario. Closest to our work in terms of formality is the work of Garetto and Gribaudo, but this presents a purely discrete model in terms of a state-labelled Markov chain which is subjected to probabilistic model checking [19] and is therefore limited in the size of system which can be considered.

Other simulators for opportunistic networks have been proposed, for example the ONE [26] and a virtual test platform [12]. These simulators work at the packet and message level and do not introduce a fluidisation of data flow. Additionally, examples studied quite often consider generation rates as low as a message an hour or a day. By using a fluid approach, we can model much higher generation rates.

Lastly, we mention other fluid approaches to modelling networks (necessarily incomplete due to

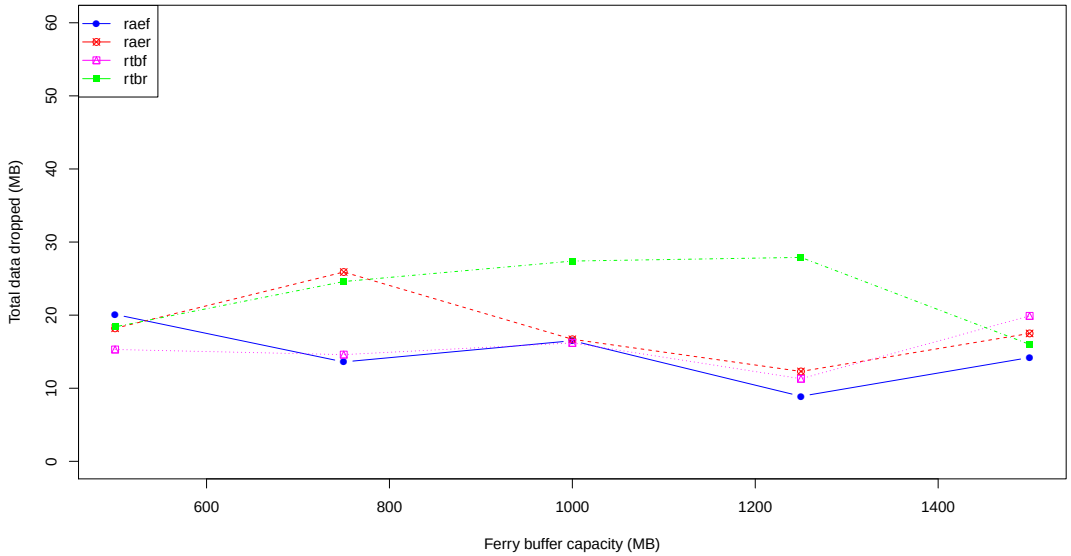


Figure 5: Average (100 simulation runs) data dropped for different ferry buffer sizes

space constraints). These include simulation [29], fluid stochastic Petri nets (FSPNs) [21, 38, 2, 15] and mean field approximations [5, 3]. These approaches, as far as we know, have not been applied to opportunistic networks and they do not offer the compositionality that a process algebra provides.

Petri net approaches include modelling a single cell of a wireless internet access system as a deterministic and stochastic Petri net; a generalised stochastic Petri net; and a FSPN [2]. The FSPN model is much faster to solve than the other two, and has good accuracy apart from a few specific scenarios. The FSPN model has a single fluid place to represent the buffer of the system. Another Petri net approach presents a FSPN model of client-server interaction in a peer-to-peer network [15] which is used to obtain distributions describing file transfer times. The single fluid place represents the download of a file. In both these examples, there is no flow between continuous places, and it is not clear how these models could be extended to model systems of multiple cells, or multiple clients and servers where continuous flows occur betweeen different elements of the system, in contrast to our approach.

6 Further research and conclusions

In terms of future work, we plan to improve the software tool in at least two directions. First, we want to improve the modularisation of the input language, allowing the modeler to write parametric templates corresponding to generic system components like ferries, data stations, and connections. Secondly, we plan to implement in the hybrid simulation more clever management of the discrete variables denoting modes of the automaton, using data structures such as dependency graphs to reduce the amount of times each guard of a discrete transition is tested. Preliminary work on this (together with bytecode on-the-fly compilation of mathematical expressions) has shown a thirty-fold increase in performance over the execution time given in Section 4.2. In addition, we plan to implement a multithread support to exploit multi-core processors to reduce simulation time.

As for the opportunistic networks, exploiting modularisation of HYPE code, we plan to define a

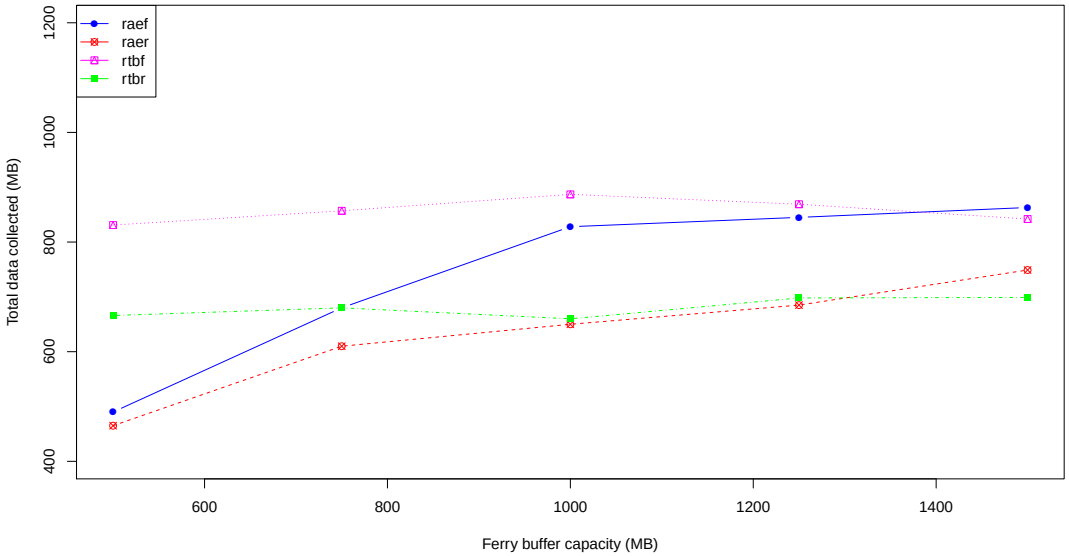


Figure 6: Average (100 simulation runs) data collected by ferry for different ferry buffer sizes

library of components and an higher level graphical interface to construct a model. As mentioned in Section 4, we wish to investigate the use of two-dimensional models of node movement. Furthermore, we plan to explore the use of HYPE bisimulations to reduce model size when possible. In certain cases, it may also be possible to reduce a sequence of events to one event while retaining the same behaviour with respect to simulation. As mentioned in Section 4.2, it should be easy to increase the size of the networks to be modelled, and hence we will investigate how our approach scales and what limitations there might be.

As a general principle, hybrid simulation can be more efficient than discrete event simulation (without continuous flows) as long as the computation time for solving the ODEs is shorter than the computation time for simulating the events that the fluid approach removes. We wish to establish whether this principle holds in modelling opportunistic networks and what the trade-off is between efficiency and accuracy of the modelling. Currently, we have focussed on providing averages and standard deviations over a number of simulation runs. An alternative approach is to formally specify the properties we are interested in using a stochastic logic and then to apply statistical model checking.

To conclude, we have presented a general framework for constructing models of opportunistic networks using stochastic HYPE. We have illustrated this through a case study of a ferry that collects data from video nodes and delivers it to a base station. The results of our simulations of the model show, as we would expect, that fixed routes are likely to be more fair and have less data dropped and more data collected. Additionally, at low buffer sizes, the penalty of returning to the base is justified to avoid dropping data unnecessarily.

Acknowledgements: This research is supported by Royal Society International Joint Project JP090562.

References

[1] *Apache Commons Math Java Library*. Available at `http://commons.apache.org/math/`.

[2] M. Ajmone Marsan, M. Gribaudo, M. Meo & M. Sereno (2001): *On Petri Net-based modeling paradigms for the performance analysis of wireless Internet accesses*. In: *9th International Workshop on Petri Nets and Performance Models*, pp. 19–28, doi:10.1109/PNPM.2001.953352.

[3] R. Bakhshi, L. Cloth, W. Fokkink & B.R. Haverkort (2011): *Mean-field framework for performance evaluation of push-pull gossip protocols*. *Performance Evaluation* 68, pp. 157–179, doi:10.1016/j.peva.2010.08.025.

[4] D.A. van Beek, K.L. Man, M.A. Reniers, J.E. Rooda & R.R.H. Schiffelers (2006): *Syntax and consistent equation semantics of hybrid* χ. *Journal of Logic and Algebraic Programming* 68, pp. 129–210, doi:10.1016/j.jlap.2005.10.005.

[5] M. Benaïm & J.-Y. Le Boudec (2008): *A class of mean field interaction models for computer and communication systems*. *Performance Evaluation* 65, pp. 823–838, doi:10.1016/j.peva.2008.03.005.

[6] J.A. Bergstra & C.A. Middelburg (2005): *Process algebra for hybrid systems*. *Theoretical Computer Science* 335, pp. 215–280, doi:10.1016/j.tcs.2004.04.019.

[7] L. Bortolussi, V. Galpin & J. Hillston (2011): *HYPE with stochastic events*. In: *QAPL 2011*, EPTCS 57, pp. 120–133, doi:10.4204/EPTCS.57.9.

[8] L. Bortolussi & A. Policriti (2009): *Hybrid semantics of stochastic programs with dynamic reconfiguration*. In: *COMPMOD 2009*, EPTCS 6, pp. 63–76, doi:10.4204/EPTCS.6.5.

[9] J. Burgess, B. Gallagher, D. Jensen & B.N. Levine (2006): *MaxProp: Routing for vehicle-based disruption-tolerant networks*. In: *INFOCOM 2006*, pp. 1–11, doi:10.1109/INFOCOM.2006.228.

[10] P.J.L. Cuijpers & M.A. Reniers (2005): *Hybrid process algebra*. *Journal of Logic and Algebraic Programming* 62, pp. 191–245, doi:10.1016/j.jlap.2004.02.001.

[11] M.H.A. Davis (1993): *Markov Models and Optimization*. Chapman & Hall.

[12] N.A. Deepak, R. Thareja & N.A. Nikhil (2008): *Performance analysis and evaluation of delay-tolerant network bundling protocol on a scalable virtual network test platform*. In: *IET International Conference on Wireless, Mobile and Multimedia Networks, 2008*, pp. 52 –55, doi:10.1049/cp:20080143.

[13] M.B. Elowitz & S. Leibler (2000): *A synthetic oscillatory network of transcriptional regulators*. *Nature* 403, pp. 335–338, doi:10.1038/35002125.

[14] M. Fadlisyah, P.C. Ölveczky & E. Ábrahám (2011): *Object-oriented formal modeling and analysis of interacting hybrid systems in HI-Maude*. In: *SEFM 2011*, LNCS 7041, pp. 415–430, doi:10.1007/978-3-642-24690-6_29.

[15] R. Gaeta, M. Gribaudo, D. Manini & M. Sereno (2005): *Fluid stochastic Petri nets for computing transfer time distributions in peer-to-peer file sharing applications*. *ENTCS* 128, pp. 79–99, doi:10.1016/j.entcs.2005.01.014.

[16] V. Galpin, L. Bortolussi & J. Hillston: *HYPE: Hybrid modelling by composition of flows*, doi:10.1007/s00165-011-0189-0. Formal Aspects of Computing, to appear.

[17] V. Galpin, L. Bortolussi & J. Hillston (2009): *HYPE: a process algebra for compositional flows and emergent behaviour*. In: *CONCUR 2009*, LNCS 5710, pp. 305–320, doi:10.1007/978-3-642-04081-8_21.

[18] V. Galpin, J. Hillston & L. Bortolussi (2008): *HYPE applied to the modelling of hybrid biological systems*. *ENTCS* 218, pp. 33–51, doi:10.1016/j.entcs.2008.10.004.

[19] M. Garetto & M. Gribaudo (2006): *Performance analysis of delay tolerant networks with model checking techniques*. In: *QEST 2006*, pp. 73–82, doi:10.1109/QEST.2006.42.

[20] T.A. Henzinger & P.-H. Ho (1995): *HYTECH: The Cornell HYbrid TECHnology Tool*. In: *Hybrid Systems II*, LNCS 999, pp. 265–293, doi:10.1007/3-540-60472-3_14.

[21] G. Horton, V.G. Kulkarni, D.M. Nicol & K.S. Trivedi (1998): *Fluid stochastic Petri nets: Theory, applications, and solution techniques*. *European Journal of Operational Research* 105, pp. 184 – 201, doi:10.1016/S0377-2217(97)00028-3.

[22] C.-M. Huang, K.-C. Lan & C.-Z. Tsai (2008): *A survey of opportunistic networks*. In: *AINA 2008*, pp. 1672–1677, doi:10.1109/WAINA.2008.292.

[23] S. Jain, K.R. Fall & R.K. Patra (2004): *Routing in a delay tolerant network*. In: *ACM SIGCOMM 2004*, pp. 145–158, doi:10.1145/1015467.1015484.

[24] S. Jain, R. Shah, W. Brunette, G. Borriello & S. Roy (2006): *Exploiting mobility for energy efficient data collection in wireless sensor networks*. *MONET* 11, pp. 327–339, doi:10.1007/s11036-006-5186-9.

[25] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.-S. Peh & Daniel Rubenstein (2002): *Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet*. *ACM SIGPLAN Notices* 37, pp. 96–107, doi:10.1145/605397.605408.

[26] A. Keränen, T. Kärkkäinen & J. Ott (2010): *Simulating mobility and DTNs with the ONE*. *Journal of Communications* 5, pp. 92–105, doi:10.4304/jcm.5.2.92-105.

[27] G.Y. Keung, B. Li & Q. Zhang (2011): *Message delivery capacity in delay-constrained mobile sensor networks: Bounds and realization*. *IEEE Transactions on Wireless Communications* 10, pp. 1552–1559, doi:10.1109/TWC.2011.030911.100827.

[28] U. Khadim (2006): *A comparative study of process algebras for hybrid systems*. Report CSR 06-23, Technische Universiteit Eindhoven. `http://alexandria.tue.nl/extra1/wskrap/publichtml/200623.pdf`.

[29] C. Kiddle, R. Simmonds, C. Williamson & B. Unger (2003): *Hybrid packet/fluid flow network simulation*. In: *PADS 2003*, pp. 143 – 152, doi:10.1109/PADS.2003.1207430.

[30] A. Lindgren, A. Doria & O. Schelén (2003): *Probabilistic routing in intermittently connected networks*. *Mobile Computing and Communications Review* 7, pp. 19–20, doi:10.1145/961268.961272.

[31] A. Passarella, M. Kumar, M. Conti & E. Borgia (2011): *Minimum-delay service provisioning in opportunistic networks*. *IEEE Transactions on Parallel and Distributed Systems* 22, pp. 1267–1275, doi:10.1109/TPDS.2010.153.

[32] L. Pelusi, A. Passarella & M. Conti (2006): *Opportunistic networking: data forwarding in disconnected mobile ad hoc networks*. *IEEE Communications Magazine* 44, pp. 134–141, doi:10.1109/MCOM.2006.248176.

[33] A. Picu & T. Spyropoulos (2010): *Distributed stochastic optimization in opportunistic networks: the case of optimal relay selection*. In: *CHANTS '10*, pp. 21–28, doi:10.1145/1859934.1859939.

[34] W.C. Rounds & H. Song (2003): *The Φ-Calculus: A language for distributed control of reconfigurable embedded systems*. In: *HSCC 2003*, LNCS 2623, pp. 435–449, doi:10.1007/3-540-36580-X_32.

[35] J. Shen, S. Moh & I. Chen (2008): *Routing protocols in delay tolerant networks: A comparative survey*. In: *23rd International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC 2008)*, pp. 1577–1580.

[36] T. Small & Z.J. Haas (2003): *The shared wireless infostation model: a new ad hoc networking paradigm (or where there is a whale, there is a way)*. In: *MobiHoc 2003*, pp. 233–244, doi:10.1145/778415.778443.

[37] T. Spyropoulos, K. Psounis & C.S. Raghavendra (2006): *Performance analysis of mobility-assisted routing*. In: *MobiHoc 2006*, pp. 49–60, doi:10.1145/1132905.1132912.

[38] B. Tuffin, D.S. Chen & K.S. Trivedi (2001): *Comparison of hybrid systems and fluid stochastic Petri nets*. *Discrete Event Dynamic Systems: Theory and Applications* 11, pp. 77–95, doi:10.1023/A:1008387132533.

[39] A. Vahdat & D. Becker (2000): *Epidemic routing for partially connected ad hoc networks*. Technical Report CS-2000-06, Duke University. Available at `issg.cs.duke.edu/epidemic/epidemic.pdf`.

[40] D. Wilkinson (2011): *Stochastic Modelling for Systems Biology*. Chapman & Hall/CRC.

[41] K. Xu, P. Hui, V.O.K. Li, J. Crowcroft, V. Latora & P. Lio (2009): *Impact of altruism on opportunistic communications*. In: *First International Conference on Ubiquitous and Future Networks*, ICUFN'09, pp. 153–158, doi:10.1109/ICUFN.2009.5174303.

[42] Z. Zhang (2006): *Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: Overview and challenges*. *IEEE Communications Surveys and Tutorials* 8, pp. 24–37, doi:10.1109/COMST.2006.323440.

[43] W. Zhao, M.H. Ammar & E.W. Zegura (2004): *A message ferrying approach for data delivery in sparse mobile ad hoc networks*. In: *MobiHoc 2004*, pp. 187–198, doi:10.1145/989459.989483.

Weak Markovian Bisimulation Congruences and Exact CTMC-Level Aggregations for Concurrent Processes

Marco Bernardo
Dipartimento di Scienze di Base e Fondamenti – Università di Urbino – Italy

We have recently defined a weak Markovian bisimulation equivalence in an integrated-time setting, which reduces sequences of exponentially timed internal actions to individual exponentially timed internal actions having the same average duration and execution probability as the corresponding sequences. This weak Markovian bisimulation equivalence is a congruence for sequential processes with abstraction and turns out to induce an exact CTMC-level aggregation at steady state for all the considered processes. However, it is not a congruence with respect to parallel composition. In this paper, we show how to generalize the equivalence in a way that a reasonable tradeoff among abstraction, compositionality, and exactness is achieved for concurrent processes. We will see that, by enhancing the abstraction capability in the presence of concurrent computations, it is possible to retrieve the congruence property with respect to parallel composition, with the resulting CTMC-level aggregation being exact at steady state only for a certain subset of the considered processes.

1 Introduction

Several Markovian behavioral equivalences (see [1] and the references therein) have been proposed in the literature for relating and manipulating system models with an underlying continuous-time Markov chain (CTMC) [15] semantics. However, only a few of them are provided with the useful capability of abstracting from internal actions. In particular, [3] has recently addressed the case in which internal actions are exponentially timed – rather than immediate like in [9] – by defining a weak Markovian bisimulation equivalence inspired by the weak (Markovian) isomorphism of [11]. The idea is to reduce to *individual* exponentially timed internal transitions all the *sequences* of exponentially timed internal transitions that traverse states enabling *only* exponentially timed internal actions, with the reduction preserving the average duration and the execution probability of the original sequences.

From a stochastic viewpoint, this reduction amounts to replacing hypoexponentially distributed durations with exponentially distributed durations having the same expected value. As a consequence, processes related by the weak Markovian bisimulation equivalence of [3] may not possess the same transient performance measures, unless they refer to properties of the form mean time to certain events. However, those processes certainly possess the same steady-state performance measures, because the aggregation induced by the considered equivalence on the CTMC underlying each process has been shown to be exact at steady state.

The weak Markovian bisimulation equivalence of [3] is not a congruence with respect to parallel composition, a fact that limits its usefulness for compositional state space reduction purposes. The contribution of this paper is to show that compositionality can be retrieved by enhancing the abstraction capability of the considered equivalence in the presence of parallel composition. The basic idea is allowing a sequence of exponentially timed internal transitions originated from a sequential process to be reduced also in the case in which that process is composed in parallel with other processes enabling *observable* actions. Unfortunately, there is a price to pay for achieving compositionality: exactness at

Preliminary Report. Final version to appear in:
QAPL 2012

steady state will no longer hold for all processes, but only for processes with no synchronization at all and processes whose synchronizations do not take place right before the sequences to be reduced.

This paper is organized as follows. After introducing a Markovian process calculus in Sect. 2 and recalling strong and weak Markovian bisimilarity in Sect. 3, in Sect. 4 we develop a variant of weak Markovian bisimilarity that deals with parallel composition and we investigate its congruence and exactness properties. Finally, in Sect. 5 we provide some concluding remarks.

2 Concurrent Markovian Processes

In order to study properties such as congruence of the variant (to be defined) of the weak Markovian bisimilarity of [3], we introduce typical behavioral operators through a Markovian process calculus (MPC for short). In [3], we have considered sequential processes with abstraction built from operators like the inactive process, exponentially timed action prefix, alternative composition, recursion, and hiding. Here, we include parallel composition too, so as to be able to represent concurrent processes.

As usual, we denote the internal action by τ and we assume that the resulting concurrent processes are governed by the race policy: if several exponentially timed actions are simultaneously enabled, the action that is executed is the one sampling the least duration. We also assume that the duration of an action deriving from the synchronization of two exponentially timed actions is exponentially distributed with a rate obtained by applying (like, e.g., in [10]) some commutative and associative operation denoted by $\otimes$ to the rates of the two original actions.

Definition 2.1 Let $Act_{\rm M} = Name \times \mathbb{R}_{>0}$ be a set of actions, where $Name = Name_{\rm v} \cup \{\tau\}$ is a set of action names – ranged over by a, b – and $\mathbb{R}_{>0}$ is a set of action rates – ranged over by λ, μ, γ. Let *Var* be a set of process variables ranged over by X, Y. The process language $\mathscr{PL}_{\rm M}$ is generated by the following syntax:

P	::=	$\underline{0}$	inactive process
	|	$<a,\lambda>.P$	exponentially timed action prefix
	|	$P+P$	alternative composition
	|	X	process variable
	|	$\mathrm{rec}\,X:P$	recursion
	|	P/H	hiding
	|	$P \|\|_S P$	parallel composition

where $a \in Name$, $\lambda \in \mathbb{R}_{>0}$, $X \in Var$, and $H, S \subseteq Name_{\rm v}$. We denote by $\mathbb{P}_{\rm M}$ the set of closed and guarded process terms of $\mathscr{PL}_{\rm M}$ – ranged over by P, Q. ■

In order to distinguish between process terms such as $<a,\lambda>.\underline{0} + <a,\lambda>.\underline{0}$ and $<a,\lambda>.\underline{0}$, like in [3] the semantic model $[\![P]\!]_{\rm M}$ for a process term $P \in \mathbb{P}_{\rm M}$ is a labeled multitransition system that takes into account the multiplicity of each transition, intended as the number of different proofs for the transition derivation. The multitransition relation of $[\![P]\!]_{\rm M}$ is contained in the smallest multiset of elements of $\mathbb{P}_{\rm M} \times Act_{\rm M} \times \mathbb{P}_{\rm M}$ that satisfies the operational semantic rules in Table 1 – where $\{\_ \hookrightarrow \_\}$ denotes syntactical replacement – and keeps track of all the possible ways of deriving each of its transitions.

3 Strong and Weak Markovian Bisimulation Equivalences

The notion of strong bisimilarity for MPC is based on the comparison of exit rates [11, 10]. The exit rate of a process term $P \in \mathbb{P}_{\rm M}$ with respect to action name $a \in Name$ and destination $D \subseteq \mathbb{P}_{\rm M}$ is the rate at

$$(\text{PRE}_\text{M})\ \frac{}{<a,\lambda>.P \xrightarrow{a,\lambda}_\text{M} P} \qquad (\text{REC}_\text{M})\ \frac{P\{\text{rec}X : P \hookrightarrow X\} \xrightarrow{a,\lambda}_\text{M} P'}{\text{rec}X : P \xrightarrow{a,\lambda}_\text{M} P'}$$

$$(\text{ALT}_{\text{M},1})\ \frac{P_1 \xrightarrow{a,\lambda}_\text{M} P'}{P_1 + P_2 \xrightarrow{a,\lambda}_\text{M} P'} \qquad (\text{ALT}_{\text{M},2})\ \frac{P_2 \xrightarrow{a,\lambda}_\text{M} P'}{P_1 + P_2 \xrightarrow{a,\lambda}_\text{M} P'}$$

$$(\text{HID}_{\text{M},1})\ \frac{P \xrightarrow{a,\lambda}_\text{M} P' \quad a \notin H}{P/H \xrightarrow{a,\lambda}_\text{M} P'/H} \qquad (\text{HID}_{\text{M},2})\ \frac{P \xrightarrow{a,\lambda}_\text{M} P' \quad a \in H}{P/H \xrightarrow{\tau,\lambda}_\text{M} P'/H}$$

$$(\text{PAR}_{\text{M},1})\ \frac{P_1 \xrightarrow{a,\lambda}_\text{M} P_1' \quad a \notin S}{P_1 \|_S P_2 \xrightarrow{a,\lambda}_\text{M} P_1' \|_S P_2} \qquad (\text{PAR}_{\text{M},2})\ \frac{P_2 \xrightarrow{a,\lambda}_\text{M} P_2' \quad a \notin S}{P_1 \|_S P_2 \xrightarrow{a,\lambda}_\text{M} P_1 \|_S P_2'}$$

$$(\text{SYN}_\text{M})\ \frac{P_1 \xrightarrow{a,\lambda_1}_\text{M} P_1' \quad P_2 \xrightarrow{a,\lambda_2}_\text{M} P_2' \quad a \in S}{P_1 \|_S P_2 \xrightarrow{a,\lambda_1 \otimes \lambda_2}_\text{M} P_1' \|_S P_2'}$$

Table 1: Structured operational semantic rules for MPC

which P can execute actions of name a that lead to D:

$$rate(P,a,D) = \sum\{|\, \lambda \in \mathbb{R}_{>0} \mid \exists P' \in D.\, P \xrightarrow{a,\lambda}_\text{M} P' \,|\}$$

where $\{|$ and $|\}$ are multiset delimiters and the summation is taken to be zero if its multiset is empty. By summing up the rates of all the actions of P, we obtain the total exit rate of P, i.e., $rate_\text{t}(P) = \sum_{a \in Name} rate(P,a,\mathbb{P}_\text{M})$, which is the reciprocal of the average sojourn time associated with P.

Definition 3.1 An equivalence relation $\mathscr{B}$ over $\mathbb{P}_\text{M}$ is a Markovian bisimulation iff, whenever $(P_1,P_2) \in \mathscr{B}$, then for all action names $a \in Name$ and equivalence classes $D \in \mathbb{P}_\text{M}/\mathscr{B}$:

$$rate(P_1,a,D) = rate(P_2,a,D)$$

Markovian bisimilarity $\sim_\text{MB}$ is the largest Markovian bisimulation. ■

As shown in [11, 10, 6, 7], the relation $\sim_\text{MB}$ possesses the following properties:

- $\sim_\text{MB}$ is a congruence with respect to all the operators of MPC as well as recursion.
- $\sim_\text{MB}$ has a sound and complete axiomatization whose basic laws are shown below:

$(\mathscr{A}_{\text{MB},1})$	$P_1 + P_2$	$=$	$P_2 + P_1$
$(\mathscr{A}_{\text{MB},2})$	$(P_1 + P_2) + P_3$	$=$	$P_1 + (P_2 + P_3)$
$(\mathscr{A}_{\text{MB},3})$	$P + \underline{0}$	$=$	P
$(\mathscr{A}_{\text{MB},4})$	$<a,\lambda_1>.P + <a,\lambda_2>.P$	$=$	$<a,\lambda_1+\lambda_2>.P$

The last one encodes the race policy and hence replaces the idempotency law $P + P = P$ valid for nondeterministic processes. The other laws are the usual distribution laws for the hiding operator and the expansion law for the parallel composition operator.

- $\sim_{\rm MB}$ induces a CTMC-level aggregation known as ordinary lumpability, which is exact both at steady state and at transient state.
- $\sim_{\rm MB}$ can be decided in polynomial time for all finite-state processes.

In [3], we have weakened the distinguishing power of $\sim_{\rm MB}$ by relating sequences of exponentially timed τ-actions to single exponentially timed τ-actions having the same average duration and execution probability as the sequences. Given $P \in \mathbb{P}_{\rm M}$, we say that P is stable if $P \stackrel{\tau,\lambda}{\not\longrightarrow}_{\rm M} P'$ for all λ and P', otherwise we say that it is unstable. In the latter case, we say that P is fully unstable iff, whenever $P \stackrel{a,\lambda}{\longrightarrow}_{\rm M} P'$, then $a = \tau$. We denote by $\mathbb{P}_{\rm M,fu}$ and $\mathbb{P}_{\rm M,nfu}$ the sets of process terms of $\mathbb{P}_{\rm M}$ that are fully unstable and not fully unstable, respectively.

The most natural candidates as sequences of exponentially timed τ-actions to abstract are those labeling computations that traverse fully unstable states.

Definition 3.2 Let $n \in \mathbb{N}_{>0}$ and $P_1, P_2, \ldots, P_{n+1} \in \mathbb{P}_{\rm M}$. A computation c of length n from P_1 to P_{n+1} having the form $P_1 \stackrel{\tau,\lambda_1}{\longrightarrow}_{\rm M} P_2 \stackrel{\tau,\lambda_2}{\longrightarrow}_{\rm M} \ldots \stackrel{\tau,\lambda_n}{\longrightarrow}_{\rm M} P_{n+1}$ is reducible iff $P_i \in \mathbb{P}_{\rm M,fu}$ for all $i = 1, \ldots, n$. ■

If reducible, the computation c above can be reduced to a single exponentially timed τ-transition whose rate is obtained from the positive real value below:

$$probtime(c) = \left(\prod_{i=1}^{n} \frac{\lambda_i}{rate(P_i,\tau,\mathbb{P}_{\rm M})} \right) \cdot \left(\sum_{i=1}^{n} \frac{1}{rate(P_i,\tau,\mathbb{P}_{\rm M})} \right)$$

by leaving its first factor unchanged and taking the reciprocal of the second one. The value $probtime(c)$ is a measure of the execution probability of c (first factor: product of the execution probabilities of the transitions of c) and the average duration of c (second factor: sum of the average sojourn times in the states traversed by c).

The weak variant of $\sim_{\rm MB}$ defined in [3] is such that (i) processes in $\mathbb{P}_{\rm M,nfu}$ are dealt with as in $\sim_{\rm MB}$ and (ii) the length of reducible computations from processes in $\mathbb{P}_{\rm M,fu}$ to processes in $\mathbb{P}_{\rm M,nfu}$ is abstracted away while preserving the execution probability and the average duration of those computations. In the latter case, we need to lift measure *probtime* from individual reducible computations to multisets of reducible computations. Denoting by $reducomp(P,D,t)$ the multiset of reducible computations from $P \in \mathbb{P}_{\rm M,fu}$ to some P' in $D \subseteq \mathbb{P}_{\rm M}$ whose average duration is $t \in \mathbb{R}_{>0}$, we consider the following t-indexed multiset of sums of *probtime* measures:

$$pbtm(P,D) = \bigcup_{t \in \mathbb{R}_{>0} \text{ s.t. } reducomp(P,D,t) \neq \emptyset} \{\!| \sum_{c \in reducomp(P,D,t)} probtime(c) |\!\}$$

Definition 3.3 An equivalence relation $\mathscr{B} \subseteq (\mathbb{P}_{\rm M,nfu} \times \mathbb{P}_{\rm M,nfu}) \cup (\mathbb{P}_{\rm M,fu} \times \mathbb{P}_{\rm M,fu})$ is a weak Markovian bisimulation iff for all $(P_1, P_2) \in \mathscr{B}$:

- If $P_1, P_2 \in \mathbb{P}_{\rm M,nfu}$, then for all $a \in Name$ and equivalence classes $D \in \mathbb{P}_{\rm M}/\mathscr{B}$:
$$rate(P_1,a,D) = rate(P_2,a,D)$$
- If $P_1, P_2 \in \mathbb{P}_{\rm M,fu}$, then for all equivalence classes $D \in \mathbb{P}_{\rm M,nfu}/\mathscr{B}$:
$$pbtm(P_1,D) = pbtm(P_2,D)$$

Weak Markovian bisimilarity $\approx_{\rm MB}$ is the largest weak Markovian bisimulation. ■

Example 3.4 Typical cases of weakly Markovian bisimilar process terms are:

$$<\tau,\mu>.<\tau,\gamma>.Q \quad <\tau,\gamma>.<\tau,\mu>.Q \quad <\tau,\frac{\mu\cdot\gamma}{\mu+\gamma}>.Q$$

and:

$$<\tau,\mu>.(<\tau,\gamma_1>.Q_1 + <\tau,\gamma_2>.Q_2)$$
$$<\tau,\tfrac{\gamma_1}{\gamma_1+\gamma_2}\cdot\left(\tfrac{1}{\mu}+\tfrac{1}{\gamma_1+\gamma_2}\right)^{-1}>.Q_1 + <\tau,\tfrac{\gamma_2}{\gamma_1+\gamma_2}\cdot\left(\tfrac{1}{\mu}+\tfrac{1}{\gamma_1+\gamma_2}\right)^{-1}>.Q_2$$

and:

$$<\tau,\mu_1>.<\tau,\gamma>.Q_1 + <\tau,\mu_2>.<\tau,\gamma>.Q_2$$
$$<\tau,\tfrac{\mu_1}{\mu_1+\mu_2}\cdot\left(\tfrac{1}{\mu_1+\mu_2}+\tfrac{1}{\gamma}\right)^{-1}>.Q_1 + <\tau,\tfrac{\mu_2}{\mu_1+\mu_2}\cdot\left(\tfrac{1}{\mu_1+\mu_2}+\tfrac{1}{\gamma}\right)^{-1}>.Q_2$$

where $Q, Q_1, Q_2 \in \mathbb{P}_{\mathrm{M,nfu}}$ (see [3] for the details). ■

Similar to weak bisimilarity for nondeterministic processes, $\approx_{\mathrm{MB}}$ is not a congruence with respect to the alternative composition operator. This problem, which has to do with fully unstable process terms, can be prevented by adopting a construction analogous to the one used in [13] for weak bisimilarity over nondeterministic process terms. In other words, we have to apply the exit rate equality check also to fully unstable process terms, with the equivalence classes to consider being the ones with respect to $\approx_{\mathrm{MB}}$.

Definition 3.5 Let $P_1, P_2 \in \mathbb{P}_{\mathrm{M}}$. We say that P_1 is weakly Markovian bisimulation congruent to P_2, written $P_1 \simeq_{\mathrm{MB}} P_2$, iff for all action names $a \in \mathit{Name}$ and equivalence classes $D \in \mathbb{P}_{\mathrm{M}}/\approx_{\mathrm{MB}}$:

$$\mathit{rate}(P_1,a,D) = \mathit{rate}(P_2,a,D)$$

■

As shown in [3], the relation $\simeq_{\mathrm{MB}}$ possesses the following properties:

- $\simeq_{\mathrm{MB}}$ is the coarsest congruence – with respect to all the operators of MPC other than parallel composition, as well as recursion – contained in $\approx_{\mathrm{MB}}$.
- $\simeq_{\mathrm{MB}}$ has a sound and complete axiomatization over the set of sequential process terms (i.e., process terms with no occurrences of the parallel composition operator), whose basic laws are those of $\sim_{\mathrm{MB}}$ plus the following one (which includes the various cases shown in Ex. 3.4):

$$(\mathscr{A}_{\mathrm{MB},5}) \qquad <a,\lambda>.\sum_{i\in I}<\tau,\mu_i>.\sum_{j\in J_i}<\tau,\gamma_{i,j}>.P_{i,j} = <a,\lambda>.\sum_{i\in I}\sum_{j\in J_i}<\tau,\tfrac{\mu_i}{\mu}\cdot\tfrac{\gamma_{i,j}}{\gamma}\cdot\left(\tfrac{1}{\mu}+\tfrac{1}{\gamma}\right)^{-1}>.P_{i,j}$$

  where $I \neq \emptyset$ is a finite index set, $J_i \neq \emptyset$ is a finite index set for all $i \in I$, $\mu = \sum_{i\in I}\mu_i$, and $\gamma = \sum_{j\in J_i}\gamma_{i,j}$ for all $i \in I$.
- $\simeq_{\mathrm{MB}}$ induces a CTMC-level aggregation called W-lumpability, which is exact only at steady state and performs reductions consistent with $\mathscr{A}_{\mathrm{MB},5}$. Moreover, $\simeq_{\mathrm{MB}}$ preserves transient properties expressed in terms of the mean time to certain events.
- $\simeq_{\mathrm{MB}}$ can be decided in polynomial time only for those finite-state processes that are not divergent, i.e., that have no cycles of exponentially timed τ-transitions.

4 Compositionality for Concurrent Processes

The relation $\simeq_{\mathrm{MB}}$ is not a congruence with respect to the parallel composition operator, thus restricting the usefulness for compositional state space reduction purposes of the framework developed in [3].

Example 4.1 Assuming parallel composition to have lower priority than any other operator, it holds that:

$$<a,\lambda>.<\tau,\mu>.<\tau,\gamma>.\underline{0} \simeq_{\mathrm{MB}} <a,\lambda>.<\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0}$$

while:

$$<a,\lambda>.<\tau,\mu>.<\tau,\gamma>.\underline{0} \parallel_\emptyset <a',\lambda'>.\underline{0} \not\simeq_{\mathrm{MB}} <a,\lambda>.<\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} \parallel_\emptyset <a',\lambda'>.\underline{0}$$

First of all, we note that:

$$<\tau,\mu>.<\tau,\gamma>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0} \not\approx_{\rm MB} <\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0}$$

In fact, for $a' \neq \tau$ the two process terms are not fully unstable with:

$$\begin{aligned} rate(<\tau,\mu>.<\tau,\gamma>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0},\tau,[<\tau,\gamma>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0}]_{\approx_{\rm MB}}) &= \mu \\ rate(<\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0},\tau,[<\tau,\gamma>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0}]_{\approx_{\rm MB}}) &= 0 \end{aligned}$$

On the other hand, for $a' = \tau$ the two process terms are fully unstable with:

$$\begin{aligned} pbtm(<\tau,\mu>.<\tau,\gamma>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0},[\underline{0} \,\|_\emptyset\, \underline{0}]_{\approx_{\rm MB}}) &= \{|\,(\tfrac{\mu}{\mu+\lambda'}\cdot\tfrac{\gamma}{\gamma+\lambda'})\cdot(\tfrac{1}{\mu+\lambda'}+\tfrac{1}{\gamma+\lambda'}+\tfrac{1}{\lambda'}), \\ &\quad (\tfrac{\mu}{\mu+\lambda'}\cdot\tfrac{\lambda'}{\gamma+\lambda'})\cdot(\tfrac{1}{\mu+\lambda'}+\tfrac{1}{\gamma+\lambda'}+\tfrac{1}{\gamma}), \\ &\quad (\tfrac{\lambda'}{\mu+\lambda'})\cdot(\tfrac{1}{\mu+\lambda'}+\tfrac{1}{\mu}+\tfrac{1}{\gamma})\,|\} \\ pbtm(<\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0},[\underline{0} \,\|_\emptyset\, \underline{0}]_{\approx_{\rm MB}}) &= \{|\,(\tfrac{\frac{\mu\cdot\gamma}{\mu+\gamma}}{\frac{\mu\cdot\gamma}{\mu+\gamma}+\lambda'})\cdot(\tfrac{1}{\frac{\mu\cdot\gamma}{\mu+\gamma}+\lambda'}+\tfrac{1}{\lambda'}), \\ &\quad (\tfrac{\lambda'}{\frac{\mu\cdot\gamma}{\mu+\gamma}+\lambda'})\cdot(\tfrac{1}{\frac{\mu\cdot\gamma}{\mu+\gamma}+\lambda'}+\tfrac{1}{\frac{\mu\cdot\gamma}{\mu+\gamma}})\,|\} \end{aligned}$$

Thus:

$$[<\tau,\mu>.<\tau,\gamma>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0}]_{\approx_{\rm MB}} \cap [<\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0}]_{\approx_{\rm MB}} = \emptyset$$

and hence:

$$rate(<a,\lambda>.<\tau,\mu>.<\tau,\gamma>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0},a,[<\tau,\mu>.<\tau,\gamma>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0}]_{\approx_{\rm MB}}) = \lambda$$

whereas:

$$rate(<a,\lambda>.<\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0},a,[<\tau,\mu>.<\tau,\gamma>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0}]_{\approx_{\rm MB}}) = 0$$

Also the two divergent process terms $\mathrm{rec}X:<\tau,\mu>.<\tau,\gamma_1>.X$ and $\mathrm{rec}X:<\tau,\mu>.<\tau,\gamma_2>.X$, $\gamma_1 \neq \gamma_2$, are related by $\simeq_{\rm MB}$ but this no longer holds when placing them in the context $\_\,\|_\emptyset <a',\lambda'>.\underline{0}$, $a' \neq \tau$. ■

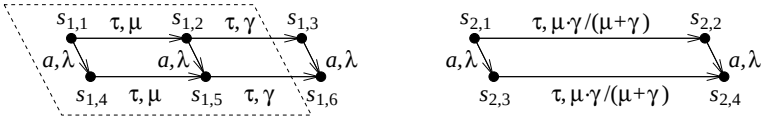
Taking inspiration from the weak isomorphism of [11], in this section we show how to retrieve full compositionality by enhancing the abstraction capability of $\simeq_{\rm MB}$ in the case of concurrent computations. The price to pay is that exactness will hold at steady state only for a certain class of processes.

4.1 Revising Weak Markovian Bisimilarity

As we have seen, $\approx_{\rm MB}$ and $\simeq_{\rm MB}$ abstract from sequences of exponentially timed τ-actions while preserving (at the computation level) their execution probability and average duration and (at the system level) transient properties expressed in terms of the mean time to certain events as well as steady-state performance measures. This kind of abstraction has been done in the simplest possible case: sequences of exponentially timed τ-actions labeling computations that traverse *fully unstable states*.

In order to achieve compositionality when dealing with concurrent processes, a revision of the notion of reducible computation is unavoidable. More precisely, we need to address the case of sequences of exponentially timed τ-actions labeling computations that traverse *unstable states satisfying certain conditions*. The reason is that, if we view a system description as the parallel composition of several sequential processes, any of those processes may have local computations traversing *fully unstable local states*, but in the overall system those local states may be *part of global states that are not fully unstable*.

For instance, this is the case with the process $<\tau,\mu>.<\tau,\gamma>.\underline{0} \,\|_\emptyset\, <a,\lambda>.\underline{0}$, whose underlying labeled multitransition system is depicted below on the left:



As can be noted, the fully unstable local states traversed by the only local computation of the sequential process $<\tau,\mu>.<\tau,\gamma>.\underline{0}$ may become part of unstable global states that are not fully unstable if $a \neq \tau$.

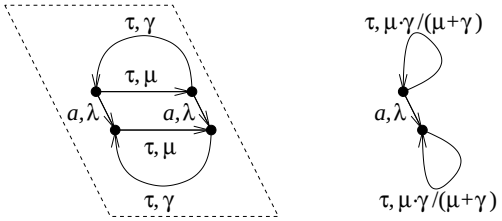
Our objective is to change the notion of reducible computation in such a way that the labeled multitransition system on the left can be regarded as being weakly Markovian bisimilar to the labeled multitransition system on the right. As can be noted, this implies that execution probabilities and average durations can only be preserved *at the level of local computations*, hence transient properties expressed in terms of the mean time to certain events can no longer be preserved at the system level.

In a concurrent setting, a sequence of exponentially timed τ-actions may be replicated due to interleaving, in the sense that it may label several computations that share no transition. The revision of the notion of reducible computation is thus based on the idea that, for each computation that traverses fully unstable local states and is labeled with exponentially timed τ-actions, we have to recognize – and take into account at once – *all the replicas* of that computation and pinpoint their initial and final states. In our example, there are two replicas with initial states $s_{1,1}$ and $s_{1,4}$ and final states $s_{1,3}$ and $s_{1,6}$.

In general, a one-to-one correspondence can be established between the states traversed by any two replicas by following the direction of the transitions. In our example, the pairs of corresponding states are the two initial states $(s_{1,1}, s_{1,4})$, the two intermediate states $(s_{1,2}, s_{1,5})$, and the two final states $(s_{1,3}, s_{1,6})$. We can say that *when moving vertically the current stage of the replicas is preserved*.

In addition to the exponentially timed τ-transition belonging to the replica, any two states traversed by the same replica can only possess transitions that are pairwise identically labeled. Those transitions are originated from (the local states of) sequential processes that are in parallel with (the local state of) the sequential process originating the considered reducible computation. The set of those transitions not belonging to the replica can thus be viewed as the *context* of the replica. In our example, the context of the top replica has a single transition labeled with $<a, \lambda>$, whereas the context of the bottom replica is empty. Thus, *when moving horizontally the context of each replica is preserved*, i.e., the context does not change along a replica. On the other hand, *different replicas may have different contexts*.

With regard to the identification of the boundary of the replicas of a reducible computation, there are two possibilities. One is that the final states have no exponentially timed τ-transition, as in our example. The other is that, at a certain point, each replica has an exponentially timed τ-transition back to one of the preceding states of the replica itself, as shown below with a variant of our example:



In this case, for each replica we view its return state as being its final state. In the figure above, for both replicas the final state coincides with the initial state.

The new notion of replicated reducible computation must be accompanied by an adjustment of the way measure *probtime* and multiset *pbtm* are calculated. Given a computation c of the form $P_1 \xrightarrow{\tau,\lambda_1}_{\mathrm{M}} P_2 \xrightarrow{\tau,\lambda_2}_{\mathrm{M}} \ldots \xrightarrow{\tau,\lambda_n}_{\mathrm{M}} P_{n+1}$ that is reducible in the sense of Def. 3.2, the denominator of the i-th fraction occurring in each of the two factors of $\mathit{probtime}(c)$ can indifferently be $\mathit{rate}(P_i, \tau, \mathbb{P}_{\mathrm{M}})$ or $\mathit{rate}_{\mathrm{t}}(P_i)$: those two values coincide because $P_i \in \mathbb{P}_{\mathrm{M,fu}}$ for all $i = 1, \ldots, n$. In contrast, if the reducible computation c is replicated, each of its replicas has a possibly different context and it is fundamental that $\mathit{rate}(P_i, \tau, \mathbb{P}_{\mathrm{M}})$ values are taken as denominators, so as to focus on τ-transitions. Since there can be τ-transitions also in the context, each destination of those exit rates needs to be a specific set $\mathscr{P}$ containing only the states traversed by the replicas rather than the generic set $\mathbb{P}_{\mathrm{M}}$. Taking into account only τ-transitions leading to states in $\mathscr{P}$ ensures *context independence* in this concurrent setting, which opens

the way to the achievement of the same *probtime* value for all the replicas of a reducible computation.

We are by now ready to provide the definition of replicated reducible computation together with the revision of both *probtime* and *pbtm*. Since several reducible computations can depart from the same state (see the second and the third pair of process terms of Ex. 3.4), in general we will have to handle *replicated trees of reducible computations* rather than replicated individual reducible computations.

In the sequel, we consider $m \in \mathbb{N}_{>0}$ process terms $P_1, P_2, \ldots, P_m \in \mathbb{P}_{\rm M}$ different from each other. We suppose that $P_k \xrightarrow{a_k,\lambda_k}_{\rm M} P_{k+1}$ for all $k = 1, \ldots, m-1$, with P_k having a nonempty tree of computations that are locally reducible for all $k = 1, \ldots, m$ (see $s_{1,1}$ and $s_{1,4}$ in our example). This tree is formalized as the set C_k^τ of all the finite-length computations starting from P_k such that each of them (i) is labeled with a sequence of exponentially timed τ-actions, (ii) traverses states that are all different with the possible exception of the final state and one of its preceding states, and (iii) shares no transitions with computations in $C_{k'}^\tau$ for all $k' \neq k$.

We further suppose that the union of $C_1^\tau, C_2^\tau, \ldots, C_m^\tau$ can be partitioned into $n \in \mathbb{N}_{>0}$ *groups of replicas* each consisting of m computations from all the m sets, such that all the computations in the same group have the same length and are labeled with the same sequence of exponentially timed τ-actions. As a consequence, for all $k = 1, \ldots, m$ we can write:

$$C_k^\tau = \{c_{k,i} \equiv P_{k,i,1} \xrightarrow{\tau,\lambda_{i,1}}_{\rm M} P_{k,i,2} \xrightarrow{\tau,\lambda_{i,2}}_{\rm M} \ldots \xrightarrow{\tau,\lambda_{i,l_i}}_{\rm M} P_{k,i,l_i+1} \mid 1 \leq i \leq n\}$$

where $P_{k,i,1} \equiv P_k$ is the initial state and $l_i \in \mathbb{N}_{>0}$ is the length of the computation for all $i = 1, \ldots, n$.

Definition 4.2 The family of computations $\mathscr{C}^\tau = \{C_1^\tau, C_2^\tau, \ldots, C_m^\tau\}$ is said to be generally reducible, or g-reducible for short, iff either $m = 1$ and for all $i = 1, \ldots, n$:

- $P_{1,i,j} \in \mathbb{P}_{\rm M,fu}$ for all $j = 1, \ldots, l_i$;
- $P_{1,i,l_i+1} \in \mathbb{P}_{\rm M,nfu}$ or $P_{1,i,l_i+1} \equiv P_{1,i,j}$ for some $j = 1, \ldots, l_i$;

or $m \geq 1$, with $P_{1,i,j} \in \mathbb{P}_{\rm M,nfu}$ for all $i = 1, \ldots, n$ and $j = 1, \ldots, l_i$ when $m = 1$, and for all $i = 1, \ldots, n$:

- For all $k = 1, \ldots, m$, $j = 1, \ldots, l_i$, and $<a, \lambda> \in Act_{\rm M}$:
  1. [Deviation from the replica] If $P_{k,i,j} \xrightarrow{a,\lambda}_{\rm M} P'$ with $P' \not\equiv P_{k,i,j+1}$, then:
     a. [change of replica via context] either $P' \equiv P_{k',i,j}$ for some $k' = 1, \ldots, m$;
     b. [change of computation] or $P' \equiv P_{k,i',j'}$ with $a = \tau$ and $\lambda = \lambda_{i',j'-1}$ for some $i' = 1, \ldots, n$ other than i and some $j' = 2, \ldots, l_{i'+1}$.
  2. [Context preservation along the replica] For all $k' = 1, \ldots, m$, it holds that $P_{k,i,j} \xrightarrow{a,\lambda}_{\rm M} P_{k',i,j}$ iff $P_{k,i,j'} \xrightarrow{a,\lambda}_{\rm M} P_{k',i,j'}$ for all $j' = 1, \ldots, l_i$.
  3. [Stage preservation across replicas] For all $i' = 1, \ldots, n$ other than i and $j' = 2, \ldots, l_{i'+1}$, it holds that $P_{k,i,j} \xrightarrow{a,\lambda}_{\rm M} P_{k,i',j'}$ iff $P_{k',i,j} \xrightarrow{a,\lambda}_{\rm M} P_{k',i',j'}$ for all $k' = 1, \ldots, m$.
- [Termination] One of the following holds:

  $\overline{4}$. Whenever there exists $\lambda_{i,l_i+1} \in \mathbb{R}_{>0}$ such that $P_{k,i,l_i+1} \xrightarrow{\tau,\lambda_{i,l_i+1}}_{\rm M} P_{k,i,l_i+2}$ for all $k = 1, \ldots, m$, then at least one of conditions 1, 2, and 3 above is not satisfied by P_{k',i,l_i+1} for some $k' = 1, \ldots, m$.

  $\widetilde{4}$. There is no $\lambda_{i,l_i+1} \in \mathbb{R}_{>0}$ such that $P_{k,i,l_i+1} \xrightarrow{\tau,\lambda_{i,l_i+1}}_{\rm M} P_{k,i,l_i+2}$ for all $k = 1, \ldots, m$.

  $\widehat{4}$. $P_{k,i,l_i+1} \equiv P_{k,i,j}$ for all $k = 1, \ldots, m$ and some $j = 1, \ldots, l_i$. ■

Some comments are now in order:

- In the case that $m = 1$ and all the traversed states are fully unstable (see the "either" option), Def. 4.2 coincides with Def. 3.2 except for the fact that the former considers a tree of computations whilst the latter considers a single computation.
- The case $m = 1$ with $P_{1,i,j} \in \mathbb{P}_{\mathrm{M,nfu}}$ for every $i = 1, \ldots, n$ and $j = 1, \ldots, l_i$ happens when all the sequential process terms in parallel with the one originating the tree of locally reducible computations repeatedly execute a single action (selfloop transition), thus causing no replica of the tree to be formed. Both this case and the case $m \geq 2$ are subject to conditions 1, 2, 3, and 4.
- Condition 1 establishes that each transition deviating (see $P' \not\equiv P_{k,i,j+1}$) from the replica of the considered computation of $\mathscr{C}^\tau$:
  - either is a vertical transition of the context that preserves the current stage of the replicas and hence causes the passage to the corresponding state of another replica ($k' \neq k$) or to the same state of the same replica ($k' = k$, meaning that one of the sequential process terms in parallel with the one originating the considered computation repeatedly executes a single action);
  - or is a transition belonging to some other computation in $\mathscr{C}^\tau$ starting from the same process term P_k as the considered computation.

  These two facts together imply the maximality of $\mathscr{C}^\tau$, because taking into account deviating transitions causes all replicas to be included. In addition, they prevent process terms like $<\tau,\mu>.(<\tau,\gamma>.\underline{0} + <a,\lambda>.\underline{0}) + <a,\lambda>.\underline{0}$ and $<\tau,\frac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} + <a,\lambda>.\underline{0}$ – which do not contain occurrences of parallel composition ($m = 1$) and have no fully unstable states when $a \neq \tau$ – from being deemed to be equivalent.
- Condition 2 is related to condition 1.a and ensures that the context of a replica is preserved along each state traversed by the replica.
- Condition 3 is related to condition 1.b and ensures that any transition belonging neither to the considered computation nor to its context (i.e., belonging to some other computation in $\mathscr{C}^\tau$) is present at the same stage of each replica of the considered computation.
- The three variants of condition 4 establish the boundary of the replicas of the considered computation in a way that guarantees the maximality of the length of the replicas themselves under (i) conditions 1, 2, and 3, (ii) the constraint that all of their transitions are labeled with exponentially timed τ-actions, (iii) and the constraint that all the traversed states are different with the possible exception of the final state and one of its preceding states.

Let $\mathit{initial}(\mathscr{C}^\tau) = \{P_k \mid 1 \leq k \leq m\}$ and $\mathit{final}(\mathscr{C}^\tau) = \{P_{k,i,l_i+1} \mid 1 \leq k \leq m, 1 \leq i \leq n\}$ be the sets of initial states and final states of the computations in $\mathscr{C}^\tau$. In order to avoid interferences between the computations in $C^\tau_1, C^\tau_2, \ldots, C^\tau_m$ and the transitions belonging to the context of those computations, for any computation $c_{k,i}$ in $\mathscr{C}^\tau$ we consider the following context-free measure:

$$\mathit{probtime}_{\mathrm{cf}}(c_{k,i}) = \left(\prod_{j=1}^{l_i} \frac{\lambda_{i,j}}{\mathit{rate}(P_{k,i,j},\tau,\mathscr{P}_k)}\right) \cdot \left(\sum_{j=1}^{l_i} \frac{1}{\mathit{rate}(P_{k,i,j},\tau,\mathscr{P}_k)}\right)$$

where $\mathscr{P}_k = \{P_{k,i',j'} \mid 1 \leq i' \leq n, 2 \leq j' \leq l_{i'+1}\}$. In this way, all replicas of the same computation will have the same $\mathit{probtime}_{\mathrm{cf}}$ measure, as shown below.

Proposition 4.3 Whenever $\mathscr{C}^\tau$ is g-reducible, then for all $k, k' = 1, \ldots, m$ and $i = 1, \ldots, n$:

$$\mathit{probtime}_{\mathrm{cf}}(c_{k,i}) = \mathit{probtime}_{\mathrm{cf}}(c_{k',i})$$

■

Moreover, we replace the generic multiset $pbtm(P,D)$ with the more specific multisets $pbtm_{\rm cf}(P_k, D \cap final(\mathscr{C}^\tau))$ for all $P_k \in initial(\mathscr{C}^\tau)$. The latter multisets are based on $probtime_{\rm cf}$ instead of $probtime$ as well as on $reducomp_{\rm cf}$ instead of $reducomp$, where $reducomp_{\rm cf}(P_k, D \cap final(\mathscr{C}^\tau), t)$ is the multiset of computations identical to those in C_k^τ that go from P_k to $D \cap final(\mathscr{C}^\tau)$ and have average duration t. We point out that computations of length zero are not considered as $t \in \mathbb{R}_{>0}$, so that whenever $P_k \in initial(\mathscr{C}^\tau) \cap D \cap final(\mathscr{C}^\tau)$, then the calculation of $pbtm_{\rm cf}(P_k, D \cap final(\mathscr{C}^\tau))$ does take into account computations identical to those in C_k^τ going from P_k to itself.

Proposition 4.4 Whenever $\mathscr{C}^\tau$ is g-reducible, then for all $k, k' = 1, \ldots, m$:

$$pbtm_{\rm cf}(P_k, final(\mathscr{C}^\tau)) = pbtm_{\rm cf}(P_{k'}, final(\mathscr{C}^\tau))$$

■

We are finally ready to introduce the revised definition of weak Markovian bisimilarity.

Definition 4.5 An equivalence relation $\mathscr{B}$ over $\mathbb{P}_{\rm M}$ is a g-weak Markovian bisimulation iff, whenever $(P_1, P_2) \in \mathscr{B}$, then:

- For all visible action names $a \in Name_{\rm v}$ and equivalence classes $D \in \mathbb{P}_{\rm M}/\mathscr{B}$:
$$rate(P_1, a, D) = rate(P_2, a, D)$$
- If P_1 is not an initial state of any g-reducible family of computations, then P_2 is not an initial state of any g-reducible family of computations either, and for all equivalence classes $D \in \mathbb{P}_{\rm M}/\mathscr{B}$:
$$rate(P_1, \tau, D) = rate(P_2, \tau, D)$$
- If P_1 is an initial state of some g-reducible family of computations, then P_2 is an initial state of some g-reducible family of computations too, and for all g-reducible families of computations $\mathscr{C}_1^\tau$ with $P_1 \in initial(\mathscr{C}_1^\tau)$ there exists a g-reducible family of computations $\mathscr{C}_2^\tau$ with $P_2 \in initial(\mathscr{C}_2^\tau)$ such that for all equivalence classes $D \in \mathbb{P}_{\rm M}/\mathscr{B}$:
$$pbtm_{\rm cf}(P_1, D \cap final(\mathscr{C}_1^\tau)) = pbtm_{\rm cf}(P_2, D \cap final(\mathscr{C}_2^\tau))$$

G-weak Markovian bisimilarity $\approx_{\rm MB,g}$ is the largest g-weak Markovian bisimulation. ■

Example 4.6 The process terms mentioned in each of the three cases of Ex. 3.4 are still related by $\approx_{\rm MB,g}$. Note that each of those process terms is the only initial state of a g-reducible family of computations composed by a single computation (first case) or a single tree of computations (second and third case) traversing only fully unstable states, thus $m = 1$ and the "either" option of Def. 4.2 applies. ■

Example 4.7 Let us reconsider the two process terms at the beginning of Ex. 4.1. Now we have:

$$<a,\lambda>.<\tau,\mu>.<\tau,\gamma>.\underline{0} \approx_{\rm MB,g} <a,\lambda>.<\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0}$$

and:

$$<a,\lambda>.<\tau,\mu>.<\tau,\gamma>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0} \approx_{\rm MB,g} <a,\lambda>.<\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0}$$

because it holds that:

$$<\tau,\mu>.<\tau,\gamma>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0} \approx_{\rm MB,g} <\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0}$$

In fact, for $a' \neq \tau$ the two process terms are the initial states of two g-reducible families of computations $\mathscr{C}_1^\tau$ and $\mathscr{C}_2^\tau$, respectively, each composed of two replicas – the first one having context $\{<a',\lambda'>\}$ and final state $\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0}$ and the second one having empty context and final state $\underline{0} \,\|_\emptyset\, \underline{0}$ – with:

$$\begin{aligned} pbtm_{\rm cf}(<\tau,\mu>.<\tau,\gamma>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0}, D \cap final(\mathscr{C}_1^\tau)) &= \{|\, \tfrac{1}{\mu} + \tfrac{1}{\gamma} \,|\} \\ pbtm_{\rm cf}(<\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0}, D \cap final(\mathscr{C}_2^\tau)) &= \{|\, \tfrac{\mu+\gamma}{\mu\cdot\gamma} \,|\} \end{aligned}$$

whenever D contains the final state $\underline{0} \,\|_\emptyset\, <a',\lambda'>.\underline{0}$, as the way of calculating $probtime_{\rm cf}$ and $pbtm_{\rm cf}$ does not take the context into account.

For $a' = \tau$, in addition to $\mathscr{C}_1^\tau$ and $\mathscr{C}_2^\tau$, the two process terms are the initial states of two further g-reducible families of computations $\mathscr{C}_1'^\tau$ and $\mathscr{C}_2'^\tau$, respectively, each composed of two replicas of length 1 labeled with $<a',\lambda'>$. In this case:

$$\begin{aligned} pbtm_{\rm cf}(<\tau,\mu>.<\tau,\gamma>.\underline{0} \,\|_\emptyset <a',\lambda'>.\underline{0}, D \cap final(\mathscr{C}_1'^\tau)) &= \{\!|\, \tfrac{1}{\lambda'} \,|\!\} \\ pbtm_{\rm cf}(<\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} \,\|_\emptyset <a',\lambda'>.\underline{0}, D \cap final(\mathscr{C}_2'^\tau)) &= \{\!|\, \tfrac{1}{\lambda'} \,|\!\} \end{aligned}$$

whenever D contains the two $\approx_{\rm MB,g}$-equivalent final states $<\tau,\mu>.<\tau,\gamma>.\underline{0} \,\|_\emptyset\, \underline{0}$ and $<\tau,\frac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} \,\|_\emptyset\, \underline{0}$. The two divergent process terms at the end of Ex. 4.1 are not related by $\approx_{\rm MB,g}$ because $\gamma_1 \neq \gamma_2$; hence, they no longer result in a disruption of compositionality when placed in the context $\_ \,\|_\emptyset <a',\lambda'>.\underline{0}$. ■

We conclude by showing that there exists a relationship between $\approx_{\rm MB,g}$ and $\approx_{\rm MB}$ only for process terms that have no cycles of exponentially timed τ-actions. The reason of this limitation is that $\approx_{\rm MB,g}$ imposes checks on those cycles that are not always performed by $\approx_{\rm MB}$, like, e.g., in the case of the two divergent process terms $\mathrm{rec}\,X : <\tau,\gamma_1>.X$ and $\mathrm{rec}\,X : <\tau,\gamma_2>.X$ where $\gamma_1 \neq \gamma_2$.

Proposition 4.8 Let $P_1, P_2 \in \mathbb{P}_{\rm M}$ be not divergent. Then:

$$P_1 \approx_{\rm MB} P_2 \implies P_1 \approx_{\rm MB,g} P_2$$

■

4.2 Congruence Property

The investigation of the compositionality of $\approx_{\rm MB,g}$ with respect to MPC operators leads to results analogous to those for $\approx_{\rm MB}$ [3], plus the achievement of congruence with respect to parallel composition.

Proposition 4.9 Let $P_1, P_2 \in \mathbb{P}_{\rm M}$. Whenever $P_1 \approx_{\rm MB,g} P_2$, then:

1. $<a,\lambda>.P_1 \approx_{\rm MB,g} <a,\lambda>.P_2$ for all $<a,\lambda> \in Act_{\rm M}$.
2. $P_1/H \approx_{\rm MB,g} P_2/H$ for all $H \subseteq Name_{\rm v}$.
3. $P_1 \,\|_S P \approx_{\rm MB,g} P_2 \,\|_S P$ and $P \,\|_S P_1 \approx_{\rm MB,g} P \,\|_S P_2$ for all $S \subseteq Name_{\rm v}$ and $P \in \mathbb{P}_{\rm M}$. ■

The relation $\approx_{\rm MB,g}$ is not a congruence with respect to the alternative composition operator due to fully unstable process terms: for instance, it holds that $<\tau,\mu>.<\tau,\gamma>.\underline{0} \approx_{\rm MB,g} <\tau,\frac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0}$ whereas $<\tau,\mu>.<\tau,\gamma>.\underline{0} + <a,\lambda>.\underline{0} \not\approx_{\rm MB,g} <\tau,\frac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} + <a,\lambda>.\underline{0}$. In fact, if it were $a \neq \tau$, then we would have:

$$\begin{aligned} rate(<\tau,\mu>.<\tau,\gamma>.\underline{0} + <a,\lambda>.\underline{0}, \tau, [\underline{0}]_{\approx_{\rm MB,g}}) &= 0 \\ rate(<\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} + <a,\lambda>.\underline{0}, \tau, [\underline{0}]_{\approx_{\rm MB,g}}) &= \tfrac{\mu\cdot\gamma}{\mu+\gamma} \end{aligned}$$

otherwise for $a = \tau$ the two process terms would be the initial states of two g-reducible families of computations, respectively, each composed of a single tree of computations with final state $\underline{0}$ and we would have:

$$\begin{aligned} pbtm_{\rm cf}(<\tau,\mu>.<\tau,\gamma>.\underline{0} + <a,\lambda>.\underline{0}, \{\underline{0}\}) &= \{\!|\, \tfrac{\mu}{\mu+\lambda} \cdot \left(\tfrac{1}{\mu+\lambda} + \tfrac{1}{\gamma}\right), \tfrac{\lambda}{\mu+\lambda} \cdot \tfrac{1}{\mu+\lambda} \,|\!\} \\ pbtm_{\rm cf}(<\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.\underline{0} + <a,\lambda>.\underline{0}, \{\underline{0}\}) &= \{\!|\, \tfrac{1}{\frac{\mu\cdot\gamma}{\mu+\gamma}+\lambda} \,|\!\} \end{aligned}$$

The congruence violation with respect to the alternative composition operator can be prevented by adopting a construction analogous to the one used in [13] for weak bisimilarity over nondeterministic process terms and adapted in [3] to $\approx_{\rm MB}$. Therefore, we have to apply the exit rate equality check for τ-actions also to process terms that are initial states of g-reducible families of computations, with the equivalence classes to consider being the ones with respect to $\approx_{\rm MB,g}$.

Definition 4.10 Let $P_1, P_2 \in \mathbb{P}_{\rm M}$. We say that P_1 is g-weakly Markovian bisimulation congruent to P_2, written $P_1 \simeq_{\rm MB,g} P_2$, iff for all action names $a \in Name$ and equivalence classes $D \in \mathbb{P}_{\rm M}/\approx_{\rm MB,g}$:

$$rate(P_1, a, D) = rate(P_2, a, D)$$

■

Proposition 4.11 $\sim_{\rm MB} \subset \simeq_{\rm MB,g} \subset \approx_{\rm MB,g}$, with $\simeq_{\rm MB,g} = \approx_{\rm MB,g}$ over the set of process terms of $\mathbb{P}_{\rm M}$ that are not initial states of any g-reducible family of computations. ■

Proposition 4.12 Let $P_1, P_2 \in \mathbb{P}_{\rm M}$ and $<a,\lambda> \in Act_{\rm M}$. Then:

$$<a,\lambda>.P_1 \simeq_{\rm MB,g} <a,\lambda>.P_2 \iff P_1 \approx_{\rm MB,g} P_2$$ ■

The relation $\simeq_{\rm MB,g}$ turns out to be the coarsest congruence – with respect to all the operators of MPC as well as recursion – contained in $\approx_{\rm MB,g}$, as shown below.

Theorem 4.13 Let $P_1, P_2 \in \mathbb{P}_{\rm M}$. Whenever $P_1 \simeq_{\rm MB,g} P_2$, then:

1. $<a,\lambda>.P_1 \simeq_{\rm MB,g} <a,\lambda>.P_2$ for all $<a,\lambda> \in Act_{\rm M}$.
2. $P_1 + P \simeq_{\rm MB,g} P_2 + P$ and $P + P_1 \simeq_{\rm MB,g} P + P_2$ for all $P \in \mathbb{P}_{\rm M}$.
3. $P_1/H \simeq_{\rm MB,g} P_2/H$ for all $H \subseteq Name_{\rm v}$.
4. $P_1 \|_S P \simeq_{\rm MB,g} P_2 \|_S P$ and $P \|_S P_1 \simeq_{\rm MB,g} P \|_S P_2$ for all $S \subseteq Name_{\rm v}$ and $P \in \mathbb{P}_{\rm M}$. ■

Theorem 4.14 Let $P_1, P_2 \in \mathbb{P}_{\rm M}$. Then $P_1 \simeq_{\rm MB,g} P_2$ iff $P_1 + P \approx_{\rm MB,g} P_2 + P$ for all $P \in \mathbb{P}_{\rm M}$. ■

With regard to recursion, we need to extend $\simeq_{\rm MB,g}$ to open process terms in the usual way. Similar to other congruence proofs for bisimulation equivalence with respect to recursion, here we rely on a notion of g-weak Markovian bisimulation up to $\approx_{\rm MB,g}$ inspired by the notion of Markovian bisimulation up to $\sim_{\rm MB}$ of [5]. This notion differs from its nondeterministic counterpart used in [13] due to the necessity of working with equivalence classes in this Markovian setting.

Definition 4.15 Let $P_1, P_2 \in \mathcal{PL}_{\rm M}$ be process terms containing free occurrences of $k \in \mathbb{N}$ process variables $X_1, \ldots, X_k \in Var$ at most. We define $P_1 \simeq_{\rm MB,g} P_2$ iff $P_1\{Q_i \hookrightarrow X_i \mid 1 \leq i \leq k\} \simeq_{\rm MB,g} P_2\{Q_i \hookrightarrow X_i \mid 1 \leq i \leq k\}$ for all $Q_1, \ldots, Q_k \in \mathcal{PL}_{\rm M}$ containing no free occurrences of process variables. ■

Definition 4.16 Let $^+$ denote the operation of transitive closure for relations. A binary relation $\mathcal{B}$ over $\mathbb{P}_{\rm M}$ is a g-weak Markovian bisimulation up to $\approx_{\rm MB,g}$ iff, whenever $(P_1, P_2) \in \mathcal{B}$, then:

- For all visible action names $a \in Name_{\rm v}$ and equivalence classes $D \in \mathbb{P}_{\rm M}/(\mathcal{B} \cup \mathcal{B}^{-1} \cup \approx_{\rm MB,g})^+$:
$$rate(P_1, a, D) = rate(P_2, a, D)$$
- If P_1 is not an initial state of any g-reducible family of computations, then P_2 is not an initial state of any g-reducible family of computations either, and for all equivalence classes $D \in \mathbb{P}_{\rm M}/(\mathcal{B} \cup \mathcal{B}^{-1} \cup \approx_{\rm MB,g})^+$:
$$rate(P_1, \tau, D) = rate(P_2, \tau, D)$$
- If P_1 is an initial state of some g-reducible family of computations, then P_2 is an initial state of some g-reducible family of computations too, and for all g-reducible families of computations $\mathcal{C}_1^\tau$ with $P_1 \in initial(\mathcal{C}_1^\tau)$ there exists a g-reducible family of computations $\mathcal{C}_2^\tau$ with $P_2 \in initial(\mathcal{C}_2^\tau)$ such that for all equivalence classes $D \in \mathbb{P}_{\rm M}/(\mathcal{B} \cup \mathcal{B}^{-1} \cup \approx_{\rm MB,g})^+$:
$$pbtm_{\rm cf}(P_1, D \cap final(\mathcal{C}_1^\tau)) = pbtm_{\rm cf}(P_2, D \cap final(\mathcal{C}_2^\tau))$$ ■

Proposition 4.17 Let $\mathcal{B}$ be a relation over $\mathbb{P}_{\rm M}$. If $\mathcal{B}$ is a g-weak Markovian bisimulation up to $\approx_{\rm MB,g}$, then $(P_1, P_2) \in \mathcal{B}$ implies $P_1 \approx_{\rm MB,g} P_2$ for all $P_1, P_2 \in \mathbb{P}_{\rm M}$. Moreover $(\mathcal{B} \cup \mathcal{B}^{-1} \cup \approx_{\rm MB,g})^+ = \approx_{\rm MB,g}$. ■

Theorem 4.18 Let $P_1, P_2 \in \mathcal{PL}_{\rm M}$ be process terms containing free occurrences of $k \in \mathbb{N}$ process variables $X_1, \ldots, X_k \in Var$ at most. Whenever $P_1 \simeq_{\rm MB,g} P_2$, then:

$$\mathrm{rec}\, X_1 : \ldots : \mathrm{rec}\, X_k : P_1 \simeq_{\rm MB,g} \mathrm{rec}\, X_1 : \ldots : \mathrm{rec}\, X_k : P_2$$ ■

4.3 Exactness at Steady State

We conclude by examining the exactness of the CTMC-level aggregation induced by $\approx_{\mathrm{MB,g}}$ and $\simeq_{\mathrm{MB,g}}$. In general, a CTMC aggregation is said to be exact at steady state (resp. transient state) iff the steady-state (resp. transient) probability of being in a macrostate of an aggregated CTMC is the sum of the steady-state (resp. transient) probabilities of being in each of the constituent microstates of the original CTMC from which the aggregated one has been obtained. This property implies the preservation of steady-state (resp. transient) reward-based performance measures across CTMC models.

The aggregation to examine – which we call GW-lumpability – shares with the one induced by $\approx_{\mathrm{MB}}$ and $\simeq_{\mathrm{MB}}$ – called W-lumpability in [3] – the characteristic of viewing certain sequences of exponentially timed τ-actions to be equivalent to individual exponentially timed τ-actions having the same average duration and the same execution probability as the corresponding sequences when the latter are considered locally to the processes originating them.[1] On the other hand, due to the idea of context embodied in the notion of g-reducible family of computations and the consequent capability of distinguishing between action disabling and action interruption, a notable difference between GW-lumpability and W-lumpability is that the former may aggregate states also in the case of concurrent processes, while the latter cannot.

Reducing a computation formed by at least two exponentially timed τ-transitions to a single exponentially timed τ-transition with the same average duration amounts to approximating a hypoexponentially (or Erlang) distributed random variable with an exponentially distributed random variable having the same expected value. This implies that, in general, GW-lumpability cannot preserve transient performance measures, as was the case with W-lumpability [3]. However, while W-lumpability at least preserves transient properties expressed in terms of the mean time to certain events, this is no longer the case with GW-lumpability as we have seen at the beginning of Sect. 4.1.

What turns out for GW-lumpability is that, similar to W-lumpability, it preserves steady-state performance measures, provided that the states traversed by any replica of a reducible computation have the same rewards and the transitions – belonging to the replica or to the context – departing from any two traversed states have pairwise identical rewards. However, unlike W-lumpability, we have to confine ourselves to processes in which synchronizations (if any) do not take place right before the beginning of computations that are reducible according to the "or" option of Def. 4.2. This constraint comes from the insensitivity conditions for generalized semi-Markov processes mentioned in [12, 8, 11].

Theorem 4.19 GW-lumpability is exact at steady state over every process term $P \in \mathbb{P}_{\mathrm{M}}$ such that, for all g-reducible families of computations $\mathscr{C}^{\tau}$ in $[\![P]\!]_{\mathrm{M}}$ with size $m \geq 2$, or size $m = 1$ and all the traversed states being not fully unstable, no state in $\mathit{initial}(\mathscr{C}^{\tau})$ is the target state of a transition in $[\![P]\!]_{\mathrm{M}}$ arising from the synchronization of two or more actions. ■

Example 4.20 In order to illustrate the need for the constraint on synchronizations in Thm. 4.19, consider the following two process terms:

$$P_1 \equiv \mathrm{rec}\, X : <\tau,\mu>.<\tau,\gamma>.<b,\delta>.X \parallel_{\{b\}} \mathrm{rec}\, Y : <a,\lambda>.<b,\delta>.Y$$
$$P_2 \equiv \mathrm{rec}\, X : <\tau,\tfrac{\mu\cdot\gamma}{\mu+\gamma}>.<b,\delta>.X \parallel_{\{b\}} \mathrm{rec}\, Y : <a,\lambda>.<b,\delta>.Y$$

Observe that $P_1 \approx_{\mathrm{MB,g}} P_2$ and that $[\![P_1]\!]_{\mathrm{M}}$ and $[\![P_2]\!]_{\mathrm{M}}$ are given by the two labeled multitransition systems depicted at the beginning of Sect. 4.1, respectively, with an additional transition labeled with $<b,\delta>$ from the final state to the initial one. In the case that $\mu = \gamma = \lambda = \delta = 1$ and $\delta \otimes \delta = \delta$, it turns out that the steady-state probability distribution for $[\![P_1]\!]_{\mathrm{M}}$ is as follows:

[1] To be precise, since the Markov property of the original CTMC is not preserved but the aggregated stochastic process is still assumed to be a CTMC, it would be more appropriate to call those aggregations pseudo-aggregations [14].

$$\begin{array}{lcllcllcl} \pi[s_{1,1}] & = & \frac{2}{13} & \pi[s_{1,2}] & = & \frac{1}{13} & \pi[s_{1,3}] & = & \frac{1}{13} \\ \pi[s_{1,4}] & = & \frac{2}{13} & \pi[s_{1,5}] & = & \frac{3}{13} & \pi[s_{1,6}] & = & \frac{4}{13} \end{array}$$

whereas the steady-state probability distribution for $[\![P_2]\!]_{\mathrm{M}}$ is as follows:

$$\begin{array}{lcllcl} \pi[s_{2,1}] & = & \frac{2}{10} & \pi[s_{2,2}] & = & \frac{1}{10} \\ \pi[s_{2,3}] & = & \frac{4}{10} & \pi[s_{2,4}] & = & \frac{3}{10} \end{array}$$

Thus, the CTMC underlying $[\![P_2]\!]_{\mathrm{M}}$ is not an exact aggregation of the CTMC underlying $[\![P_1]\!]_{\mathrm{M}}$ because:

$$\begin{array}{lcllcl} \pi[s_{1,1}]+\pi[s_{1,2}] & \neq & \pi[s_{2,1}] & \pi[s_{1,3}] & \neq & \pi[s_{2,2}] \\ \pi[s_{1,4}]+\pi[s_{1,5}] & \neq & \pi[s_{2,3}] & \pi[s_{1,6}] & \neq & \pi[s_{2,4}] \end{array}$$

As can be noted, the transition in $[\![P_1]\!]_{\mathrm{M}}$ labeled with $<b,\delta>$ arises from the synchronization of two b-actions and its target state is the initial state of a computation belonging to a g-reducible family with size $m = 2$; hence, Thm. 4.19 does not apply.

In contrast, if we consider a synchronization-free variant of the two process terms above like for instance:

$$\begin{array}{lcl} P_3 & \equiv & \mathrm{rec}\, X : <\tau,\mu>.<\tau,\gamma>.<b_1,\delta_1>.X \parallel_{\emptyset} \mathrm{rec}\, Y : <a,\lambda>.<b_2,\delta_2>.Y \\ P_4 & \equiv & \mathrm{rec}\, X : <\tau,\frac{\mu\cdot\gamma}{\mu+\gamma}>.<b_1,\delta_1>.X \parallel_{\emptyset} \mathrm{rec}\, Y : <a,\lambda>.<b_2,\delta_2>.Y \end{array}$$

we have that for $\mu = \gamma = \lambda = \delta_1 = \delta_2 = 1$ the steady-state probability distribution for $[\![P_3]\!]_{\mathrm{M}}$ is:

$$\begin{array}{lcllcllcl} \pi[s_{3,1}] & = & \frac{1}{6} & \pi[s_{3,2}] & = & \frac{1}{6} & \pi[s_{3,3}] & = & \frac{1}{6} \\ \pi[s_{3,4}] & = & \frac{1}{6} & \pi[s_{3,5}] & = & \frac{1}{6} & \pi[s_{3,6}] & = & \frac{1}{6} \end{array}$$

and the steady-state probability distribution for $[\![P_4]\!]_{\mathrm{M}}$ is:

$$\begin{array}{lcllcl} \pi[s_{4,1}] & = & \frac{2}{6} & \pi[s_{4,2}] & = & \frac{1}{6} \\ \pi[s_{4,3}] & = & \frac{2}{6} & \pi[s_{4,4}] & = & \frac{1}{6} \end{array}$$

hence the CTMC underlying $[\![P_4]\!]_{\mathrm{M}}$ is an exact aggregation of the CTMC underlying $[\![P_3]\!]_{\mathrm{M}}$ because:

$$\begin{array}{lcllcl} \pi[s_{3,1}]+\pi[s_{3,2}] & = & \pi[s_{4,1}] & \pi[s_{3,3}] & = & \pi[s_{4,2}] \\ \pi[s_{3,4}]+\pi[s_{3,5}] & = & \pi[s_{4,3}] & \pi[s_{3,6}] & = & \pi[s_{4,4}] \end{array}$$

■

5 Conclusion

In this paper, we have introduced $\approx_{\mathrm{MB,g}}$ and $\simeq_{\mathrm{MB,g}}$ as variants of the weak Markovian bisimulation equivalences $\approx_{\mathrm{MB}}$ and $\simeq_{\mathrm{MB}}$ proposed in [3], which suffer from a limited usefulness for state space reduction purposes as they are not congruences with respect to the parallel composition operator. The motivation behind $\approx_{\mathrm{MB,g}}$ and $\simeq_{\mathrm{MB,g}}$ is thus that of retrieving full compositionality. Taking inspiration from the idea of preserving the context of [11], this has been achieved by enhancing the abstraction capability – with respect to $\approx_{\mathrm{MB}}$ and $\simeq_{\mathrm{MB}}$ – when dealing with concurrent computations. The price to pay for the resulting compositional abstraction capability is that the exactness at steady state of the induced CTMC-level aggregation does not hold for all the considered processes – as it was for $\approx_{\mathrm{MB}}$ and $\simeq_{\mathrm{MB}}$ – but only for sequential processes with abstraction and concurrent processes whose synchronizations do not take place right before the beginning of computations to be reduced. Additionally, not even transient properties expressed in terms of the mean time to certain events are preserved in general.

With regard to [11], where weak isomorphism has been studied, our equivalences $\approx_{\mathrm{MB,g}}$ and $\simeq_{\mathrm{MB,g}}$ have been developed in the more liberal bisimulation framework. A more important novelty with respect to weak isomorphism is that we have considered not only individual sequences of exponentially timed τ-actions. In fact, we have addressed trees of exponentially timed τ-actions and we have established the conditions under which such trees can be reduced – also in the presence of parallel composition – by

locally preserving both the average duration and the execution probability of their branches.

Another approach to abstracting from τ-actions in an exponentially timed setting comes from [4], where a variant of Markovian bisimilarity was defined that checks for exit rate equality with respect to all equivalence classes apart from the one including the processes under examination. Congruence and axiomatization results were provided for the proposed equivalence, and a logical characterization based on CSL was illustrated in [2]. However, unlike $\approx_{\mathrm{MB,g}}$ and $\simeq_{\mathrm{MB,g}}$, nothing was said about exactness.

As far as future work is concerned, we would like to investigate equational and logical characterizations of $\simeq_{\mathrm{MB,g}}$ as well as conduct case studies for assessing its usefulness in practice (especially with respect to the constraint on synchronizations that guarantees steady-state exactness). With regard to verification issues, since $\simeq_{\mathrm{MB}} \subset \simeq_{\mathrm{MB,g}}$ for non-divergent process terms, we have that the equivalence checking algorithm developed for $\simeq_{\mathrm{MB}}$ in [3] can be exploited for compositional state space reduction with respect to $\simeq_{\mathrm{MB,g}}$, by applying it to each of the sequential processes composed in parallel.

Acknowledgment: This work has been funded by MIUR-PRIN project *PaCo – Performability-Aware Computing: Logics, Models, and Languages*.

References

[1] A. Aldini, M. Bernardo, and F. Corradini, *"A Process Algebraic Approach to Software Architecture Design"*, Springer, 2010.

[2] C. Baier, J.-P. Katoen, H. Hermanns, and V. Wolf, *"Comparative Branching-Time Semantics for Markov Chains"*, in Information and Computation 200:149–214, 2005.

[3] M. Bernardo, *"Weak Markovian Bisimulation Congruences and Exact CTMC-Level Aggregations for Sequential Processes"*, to appear in Proc. of TGC 2011.

[4] M. Bravetti, *"Revisiting Interactive Markov Chains"*, in Proc. of MTCS 2002, ENTCS 68(5):1–20.

[5] M. Bravetti, M. Bernardo, and R. Gorrieri, *"A Note on the Congruence Proof for Recursion in Markovian Bisimulation Equivalence"*, in Proc. of PAPM 1998, pp. 153–164.

[6] P. Buchholz, *"Exact and Ordinary Lumpability in Finite Markov Chains"*, in Journal of Applied Probability 31:59–75, 1994.

[7] S. Derisavi, H. Hermanns, and W.H. Sanders, *"Optimal State-Space Lumping in Markov Chains"*, in Information Processing Letters 87:309–315, 2003.

[8] W. Henderson and D. Lucic, *"Aggregation and Disaggregation Through Insensitivity in Stochastic Petri Nets"*, in Performance Evaluation 17:91–114, 1993.

[9] H. Hermanns, *"Interactive Markov Chains"*, LNCS 2428, 2002.

[10] H. Hermanns and M. Rettelbach, *"Syntax, Semantics, Equivalences, and Axioms for MTIPP"*, in Proc. of PAPM 1994, pp. 71–87.

[11] J. Hillston, *"A Compositional Approach to Performance Modelling"*, Cambridge University Press, 1996.

[12] K. Matthes, *"Zur Theorie der Bedienungsprozesse"*, in Proc. of the *3rd Prague Conf. on Information Theory, Statistical Decision Functions and Random Processes*, pp. 513–528, 1962.

[13] R. Milner, *"Communication and Concurrency"*, Prentice Hall, 1989.

[14] G. Rubino and B. Sericola, *"Sojourn Times in Finite Markov Processes"*, in Journal of Applied Probability 27:744–756, 1989.

[15] W.J. Stewart, *"Introduction to the Numerical Solution of Markov Chains"*, Princeton University Press, 1994.