Pre-Proceedings of

Quantitative Aspects of Programming Languages and Systems 2017

Uppsala, Sweden, April 23, 2017

Preface

This document contains the pre-proceedings of QAPL 2017, the 15th international workshop on Quantitative Aspects of Programming Languages and Systems, held on Sunday April 23, 2017 in Uppsala in the context of ETAPS 2017, the 12th European Joint Conferences on Theory and Practice of Software.

The aim of the QAPL workshops is to discuss new developments on the quantitative evaluation of systems, with an emphasis on quantitative aspects of computation, broadly construed. The papers presented at the workshops discuss theory, engineering methodologies, tools, case studies, as well as experience reports where quantitative properties such as bandwidth, cost, energy, memory, performance, probability, reliability, security, and time are first-class citizens.

The QAPL 2017 program featured two invited presentations, one by Erika Ábrahám (RWTH Aachen) entitled *Divide and Conquer: Variable Set Separation in Hybrid Systems Reachability Analysis*, and by Andrea Vandin (IMT Lucca) entitled *Language-Based Abstractions for Dynamical Systems*, completed by six technical papers and two presentation reports.

The program committee of QAPL 2017 consisted of the following members:

- Alessandro Abate, University of Oxford, UK
- Alessandro Aldini, University of Urbino “Carlo Bo”, IT
- Pedro D’Argenio, Universidad Nacional de Córdoba, AR
- Josée Desharnais, Université Laval, CA
- Alessandra Di Pierro, Universitá di Verona, IT
- Antonio Filieri, Imperial College London, UK
- Jane Hillston, University of Edinburgh, UK
- Jan Křetinský, TU Munich, DE
- Mieke Massink, CNR–ISTI, IT
- David Šafránek, Masaryk University, CZ
- Ana Sokolova, University of Salzburg, AT
- Jiri Srba, Aalborg University, DK
- Marielle Stoelinga University of Twente, NL
- Andrea Turrini, Chinese Academy of Sciences, Beijing, CN
- Erik de Vink, Eindhoven University of Technology, NL
- Herbert Wiklicky, Imperial College London, UK

As workshop co-chairs of QAPL 2017 we are grateful to all who have contributed to make this installment of the QAPL workshop series a success.

Erik de Vink and Herbert Wiklicky, April 2017

Programme

09:00 - 10:00 INVITED TALK
Erika Abraham (with Stefan Schupp and Johanna Nellen): *Divide and Conquer: Variable Set Separation in Hybrid Systems Reachability Analysis*

10:00 - 10:30 COFFEE BREAK

10:30 - 11:00 PAPER
Sebastian Arming, Ezio Bartocci and Ana Sokolova: *SEA-PARAM: Exploring Schedulers in Parametric MDPs*

11:00 - 11:30 PAPER
Sebastian Kpper, Barbara Knig and Christina Mika: *Paws: A Tool for the Analysis of Weighted Systems*

11:30 - 12:00 PAPER
Stephan Brandauer and Tobias Wrigstad: *Mining for Safety using Interactive Trace Analysis*

12:00 - 12:30 PAPER
Valentina Castiglioni and Simone Tini: *Logical Characterization of Trace Metrics*

12:30 - 14:00 LUNCH

14:00 - 15:00 INVITED TALK
Andrea Vandin: *Language-based abstractions for dynamical systems*

15:00 - 15:30 PAPER
Diego Latella and Mieke Massink: *Design and Optimisation of the FlyFast Front-end for Attribute-based Coordination*

15:30 - 16:00 COFFEE BREAK

16:00 - 16:30 PAPER
Jacob Lidman and Josef Svenningsson: *Bridging static and dynamic program analysis using fuzzy logic*

16:30 - 17:00 PRESENTATION
Yuri Gil Dantas, Tobias Hamann, Heiko Mantel and Johannes Schickel: *An Experimental Study of a Bucketing Approach*

17:00 - 17:30 PRESENTATION
Pranav Ashok, Jan Kretinsky, Tobias Meggendorfer, Przemyslaw Daca and Krishnendu Chatterjee: *Mean-payoff objectives for Markov Decision Processes*

Divide and Conquer: Variable Set Separation in Hybrid Systems Reachability Analysis*

Stefan Schupp Johanna Nellen Erika Ábrahám
RWTH Aachen University, Germany
{ stefan.schupp | johanna.nellen | abraham }@cs.rwth-aachen.de

In this paper we propose an improvement for flowpipe-construction-based reachability analysis techniques for hybrid systems. Such methods apply iterative successor computations to pave the reachable region of the state space by state sets in an over-approximative manner. As the computational costs steeply increase with the dimension, in this work we analyse the possibilities for improving scalability by dividing the search space in sub-spaces and execute reachability computations in the sub-spaces instead of the global space. We formalise such an algorithm and provide experimental evaluations to compare the efficiency as well as the precision of our sub-space search to the original search in the global space.

1 Introduction

The increasing usage of digital control for safety-critical dynamical systems has resulted in an increasing need for formal verification approaches for *hybrid systems*, i.e., for systems with mixed discrete-continuous behaviour, which are often modelled as *hybrid automata*. Due to intensive research, nowadays several approaches and tools exist for the *reachability analysis* of hybrid automata. As the reachability problem for hybrid automata is in general undecidable, most approaches compute an *over-approximation* of the set of states that are reachable in a given hybrid automaton model. Due to the over-approximation, these techniques can be used to prove the safety of system models, i.e., the fact that a given set of unsafe states is not reachable in the model, but they cannot be used to prove unsafety.

In this work we focus on *flowpipe-construction-based* reachability analysis techniques. These techniques use certain data types to *represent* state sets, whereas each representation has its strengths and weaknesses in terms of precision, memory requirements, and efficiency of certain operations on them which are needed for the reachability computations. The reachability analysis starts from an initial state set and iteratively over-approximates successors by further state sets. For a given set of states, the successors via a discrete computation step (*jump*) are over-approximated by a single set, the successors via time evolution (the so-called *flowpipe*) are covered in an over-approximative manner by a sequence of state sets.

Unfortunately, these successor computations often lead to either strong over-approximations or high computational costs. Though state-of-the-art tools like SpaceEx [7], Flow* [4], or HyPro [19] can already successfully verify a wide range of challenging applications, they still have problems to analyse large models with complex behaviour. Such models arise for example from applications, where a physical or chemical plant is controlled by a discrete controller. Our focus is on digital control by programs running on *programmable logic controllers* (*PLCs*). To build a formal model of such a system, the PLCs, the programs running on them, the dynamic plant behaviour, and the interactions between these

*This work was partially supported by the German Research Council (DFG) in the context of the HyPro project.

Submitted to:
QAPL 2017

components can be modelled by a hybrid automaton, to which available reachability analysis tools can be applied. For practically relevant systems, however, the size of the resulting composed models often exceeds the capabilities of state-of-the-art tools.

Whereas general techniques to increase the scalability of reachability analysis are hard to develop, for dedicated model types there might be some hot spots. Models of PLC-controlled plants have some specific properties we can exploit: Firstly, they possess a relevant number of *discrete variables*. Secondly, some actions are triggered by deadlines, modelled by the values of *clocks* along with corresponding thresholds. Thirdly, the evolution of some physical quantities might depend on the time *linearly*, others not. In standard reachability analysis, these model parts are handled uniquely. In this paper we propose to split the state space into several sub-spaces, between which the dependence is loose enough to execute successor computations independently. Though this procedure leads to additional over-approximation, the error can be reduced. Furthermore, we show on some experiments that this additional over-approximation is often minor and is well compensated by the reduced computational requirements.

We are aware of the work [5] that is closely related to the work described in this paper. The authors of [5] also use variable set separation and computations in sub-spaces, but with two main differences. On the one hand, the work [5] is more general as they allow also closer dependencies between the sub-spaces than we can support. On the other hand, their work is restricted to Taylor models, whereas our approach is applicable to any state set representation type.

Overview After providing some preliminaries in Section 2 and a description of our HyPro programming library in Section 3, in Section 4 we describe our method for the separation of variable dimensions and the modified reachability algorithm. We provide some experimental results in Section 5 before we conclude the paper in Section 6.

2 Preliminaries

Hybrid automata For a given set $X = \{x_1, \ldots, x_d\}$ of variables let $Pred_X$ be the set of all quantifier-free arithmetic predicates with free variables from X, using the standard syntax and semantics over the real domain. We use the notation $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_d\}$ to represent first derivatives and $X' = \{x'_1, \ldots, x'_d\}$ to represent the result of discrete resets of variable values. Sometimes we also see the variable space as the d-dimensional real space and use the vector notation $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$.

Definition 1 ([10]). *A* hybrid automaton $\mathcal{H} = (Loc, X, Flow, Inv, Edge, Init)$ *is a tuple specifying*

- *a finite set* Loc *of* locations *or* control modes*;*
- *a finite ordered set* $X = \{x_1, \ldots, x_d\}$ *of real-valued* variables*, where* d *is the* dimension *of* $\mathcal{H}$*;*
- *for each location its* flow *or* dynamics *by the function* $Flow : Loc \to Pred_{X \cup \dot{X}}$*;*
- *for each location an* invariant *by the function* $Inv : Loc \to Pred_X$*;*
- *a finite set* $Edge \subseteq Loc \times Pred_X \times Pred_{X \cup X'} \times Loc$ *of* discrete transitions *or* jumps*. For a jump* $(l_1, g, r, l_2) \in Edge$*,* l_1 *is its* source *location,* l_2 *is its* target *location,* g *specifies the jump's* guard*, and* r *its* reset *function;*
- *an* initial *predicate for each location by the function* $Init : Loc \to Pred_X$.

In this paper we consider only *autonomous linear* hybrid automata whose initial conditions, invariants and jump guards are linear and can be written in the form $Ax \leq b$ (where $x = (x_1, \ldots, x_d)$ are the

model variables, A is a $d \times d$ matrix and b a d-dimensional vector), whose jump resets are also linear and can be written as $x' = Ax$, and whose flows are defined by conjunctions of linear *ordinary differential equations* (*ODEs*), which can be written[1] as $\dot{x} = Ax$. Note that such automata allow only linear predicates, whose solutions are convex polytopes.

A *state* $(l,x) \in Loc \times \mathbb{R}^d$ of a hybrid automaton specifies the location $l \in Loc$ in which the control resides and the current values $x \in \mathbb{R}^d$ of the variables. For $p \subseteq \mathbb{R}^d$, by (l,p) we denote the state set $\{(l,x) \mid x \in p\}$. An *execution* $(l_0,x_0) \xrightarrow{t_0} (l_1,x_1) \xrightarrow{e_1} (l_2,x_2) \xrightarrow{t_2} \dots$ of a hybrid automaton starts in an initial state (l_0,x_0) such that x_0 satisfies $Init(l_0)$, and executes a sequence of alternating continuous and discrete steps. A continuous step $(l_i,x_i) \xrightarrow{t_i} (l_{i+1},x_{i+1})$ with $l_i = l_{i+1}$ models time evolution: starting from x_i, the variable values evolve according to the flow (ODEs) $Flow(l_i)$ of the current location for t_i time units, where the location's invariant must hold during the whole duration of the step. A discrete step $(l_i,x_i) \xrightarrow{e_i} (l_{i+1},x_{i+1})$ typically models controller execution: if the source of a jump e_i is l_i, the guard of e_i is satisfied by x_i, the reset predicate of e_i is satisfied by (x_i,x_{i+1}), and l_{i+1}'s invariant is true for x_{i+1} then the jump e_i can be taken, moving the control from l_i to l_{i+1} with resulting variable values x_{i+1}.

A state of a hybrid automaton $\mathscr{H}$ is called *reachable* if there is an execution leading to it. Given a set T of unsafe states, $\mathscr{H}$ is called *safe* if no state from T is reachable in $\mathscr{H}$.

Hybrid automata can be composed using *parallel composition*, which we do not define here formally. Intuitively, jumps in different components can be synchronised (using synchronisation labels) if they should take place simultaneously, whereas local computation steps can also be executed in isolation; time evolves simultaneously in all components.

Reachability analysis The *reachability problem* for hybrid automata is the problem to decide whether a given state (or any state from a given set) is reachable in a hybrid automaton. As the reachability problem for hybrid automata is in general undecidable, some approaches aim at computing an *over-approximation* of the set of reachable states of a given hybrid automaton. We focus on approaches based on *flowpipe construction*, which iteratively over-approximate the set of reachable states by the union of a set of state sets. To *represent* a state set, typically either a *geometric* or a *symbolic* representation is used. Geometric representations specify state sets by geometric objects like boxes, (convex) polytopes, zonotopes, or ellipsoids, whereas symbolic representations use, e.g., support functions or Taylor models. These representations might have major differences in the precision of the representation (the size of over-approximation), the memory requirements and the computational effort needed to apply operations like intersection, union, linear transformation, Minkowski sum or test for emptiness. For example, boxes perform well in terms of computational effort for set operations, but usually introduce a large over-approximation error. Thus the choice of the representation is a compromise between the advantages and disadvantages regarding these measures.

Before the reachability analysis starts, all predicates $\varphi \in Pred_X$ in the respective linear hybrid automaton (initial predicates, invariants, jump guards) as well as the unsafe state set need to be represented in some state set representation (usually the same representation for all predicates). Furthermore, the jump resets need to be formalised as linear transformations.

For a given state set p, flowpipe-construction-based approaches compute the successors of the states from p by first over-approximating the set of states reachable via time evolution (*flowpipe*) and afterwards the set of states reachable from the flowpipe as jump successors. Time evolution is usually restricted to a *time horizon* (either per location or for the whole execution), which is divided into smaller time steps. The states reachable from p via one time step are over-approximated by a state set p_1, for which again

[1] Note that ODEs of the form $\dot{x} = Ax + b$ can be also encoded without a b component on the cost of new variables with zero derivatives. A similar approach is possible for jump resets of the form $x' = Ax + b$.

the time successors p_2 via one time step are computed. This procedure is repeated until the time horizon is reached or the successor set gets empty (due to the violation of the current location's invariant). The union of the resulting state sets $p_1, \ldots, p_k$, which are called *flowpipe segments*, over-approximates the flowpipe. For each outgoing jump and each of the flowpipe segments the jump successors are computed, to which the above procedure is applied iteratively until a given upper bound (*jump depth*) on the number of jumps is reached or until a fixed point is detected. Thus the reachability computation results in a *search tree* with state sets as nodes. In order to reduce the computational effort, *clustering* and *aggregation* can be applied to over-approximate the successors of the flowpipe segments for a given jump by a fewer number of segments respectively by a single state set.

Programmable logic controllers *Programmable logic controllers* (*PLCs*) are digital controllers widely used in industrial applications, for instance in production chains. A PLC has input and output pins that are connected with the sensors and the actuators of a plant. Control programs running on a PLC specify the output of the PLC in dependence of its input. These control programs are executed in a cyclic manner. First, the PLC *reads* the current state of the sensors and the actuators of the plant and stores this information in input registers. Next all programs on the PLC *execute* in parallel to compute the next output values based on the last input, and store the results in some output registers. These computations might use local variables, stored in some local registers. In the last step of the cycle the PLC *writes* the computed output values to the output pins that are connected to the actuators of the plant. In contrast to some implementations that assure a cycle duration within a time interval, for simplicity in this work we assume a constant cycle time (however, our approach can be easily extended to interval durations).

To model a plant we introduce variable sets V_{cont}, V_{act}, and V_{sen} to represent the state of physical quantities, the actuators, respectively the sensors (see left of Figure 1). For the modelling of a controller we use sets V_{in}, V_{out}, and V_{loc} of variables to represent the PLC registers for input, output respectively local variables. Additionally, we need one variable (clock) per PLC to account for the PLC cycle time.

A schematic overview of the hybrid automaton we use to model PLC-controlled plants is shown in the right of Figure 1. We could model the system by specifying hybrid automata models for the plant, the PLC, and the programs running on the PLC, and compose them using label synchronisation to model synchronous events in the PLC cycle. However, these models allow heavy interleaving between continuous time evolution and discrete PLC computation steps, leading to models that pose a challenge for reachability analysis tools. Therefore, we make use of the fact that the PLC execution between reading the input and writing the output has no influence on the plant's state: we model the plant evolution and the concurrent cyclic PLC execution by toggling between a controller model and a plant model, assuming that all controller actions are executed instantaneously after the input is read, the plant evolves for the duration of the PLC cycle, and the output is written at the end of the cycle. We refer to [18] for more information on the modelling of PLC-controlled plants.

3 The HyPro Library

As mentioned before, there are several state set representations that can be used in flowpipe-construction-based reachability analysis algorithms. Hybrid systems reachability analysis tools like, e.g., CORA [1], FLOW* [4], HYCREATE [11], HYREACH [13], SOAPBOX [9], and SPACEEX [7] implement different techniques using different geometric or symbolic state set representations, each of them having individual strengths and weaknesses. For example, SPACEEX uses support functions, whereas FLOW* makes use of Taylor models.

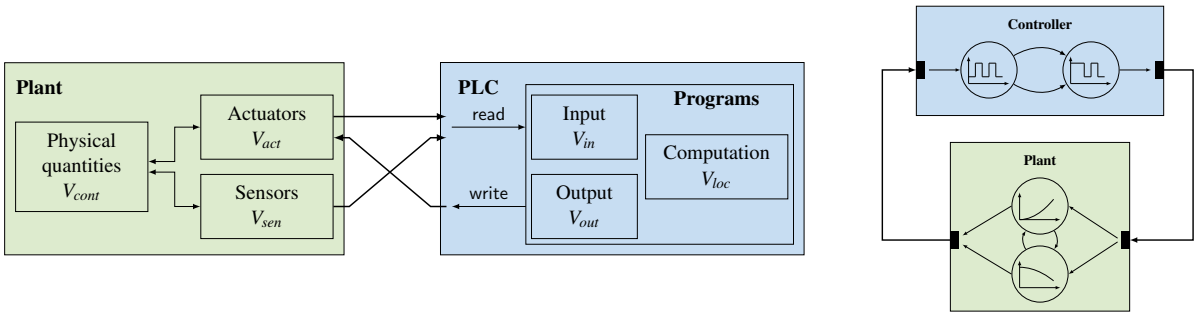


Figure 1: PLC controller: Interface between plant and controller (left) and cyclic execution model (right).

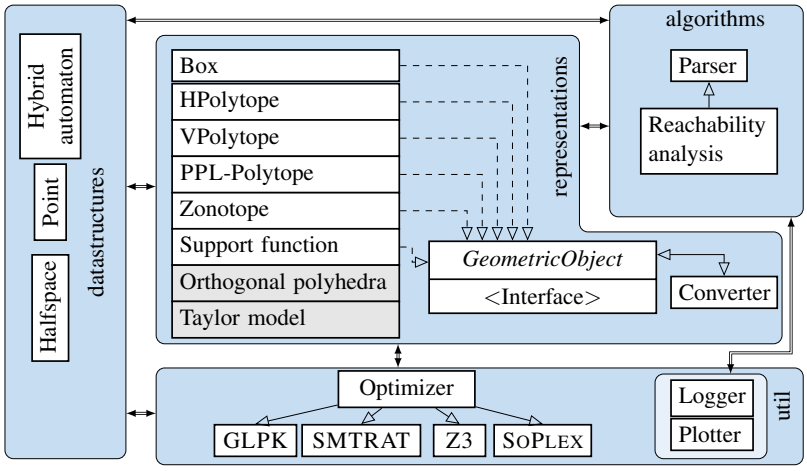


Figure 2: HyPro class structure [19].

The implementation of state set representations is tedious and time-consuming, and impedes the (even prototypical) implementation of new reachability analysis algorithms. To offer assistance for rapid implementation, we developed a free and open-source C++ programming library HyPro [19] (see Figure 2), which we will use in our experiments and which is published at `https://github.com/hypro/hypro`. HyPro contains implementations for several *state set representations* such as boxes [16], convex polytopes [21], zonotopes [8], support functions [14], orthogonal polyhedra [3], and Taylor models [4], different *operations* on them which are needed for the implementation of flowpipe-construction-based reachability analysis algorithms, and *conversions* between the different representations. *Reduction techniques* can be applied to reduce the representation sizes on the cost of additional over-approximation.

The implemented representations (with the exceptions of orthogonal polyhedra and Taylor models, depicted grey in Figure 2) share a *unified interface* to allow the usage of different representations within a single algorithm. This property is not only important for extensibility with new representations but also, e.g., for the implementation of counterexample-guided abstraction refinement (CEGAR) algorithms: the search can start with a low-precision but computationally cheap representation such as boxes, and it can be refined along paths that are detected to be potentially unsafe by switching to a high-precision but computationally more expensive representation.

Another important feature of HyPro is that it is templated in the *number type*, such that it can be instantiated both with exact as well as with inexact arithmetic. Linear solver backends such as GLPK

[15], SMTRAT [6], SoPlex [20], and Z3 [17], which are needed for the implementation of different operations and conversions, can be exchanged by the user by her tool of choice. The library is *thread-safe*, thus parallelisation can be exploited by the user. The efficient usage of the library is further eased by a *model parsing* module, a *plotting* engine, and various *debugging* tools.

In this work we illustrate the advantages of the HyPro library by proposing an algorithm to reduce the computational effort of the search on the cost of precision loss. Due to space restriction, we do not discuss refinement steps in this paper, but mention here that using HyPro the proposed method can be embedded into a CEGAR approach: if a potentially unsafe path is detected, more precise analysis can be used to check safety along those paths.

4 Reachability Analysis based on Variable Set Separation

Variable set separation and projective representation For practically relevant applications, the previously described modelling approach for PLC-controlled plants by hybrid automata leads to huge models, even if we exploit the mentioned reduction by restricted interleaving. The most serious problem is the high dimensionality: the variable set contains variables modelling the plant dynamics, the states of sensors and actuators, the input and output values of the PLC, the local variables used in program executions, and clocks for PLC cycle synchronisation. The high dimensionality leads to complex state set representations, causing heavy memory consumption and computationally expensive applications of state set operations during the reachability analysis.

To increase scalability and thus to allow the analysis of larger models, we start with some observations. Firstly, the variables of the PLC are *discrete* and thus their values do not change dynamically during time evolution but only upon taking a discrete transition in the controller part of the composed hybrid automaton. Furthermore, the states of actuators and sensors can be modelled by discrete variables, as actuator states change discretely (when writing the output) and the sensor values are relevant only at the beginning of each cycle (read plant state) as depicted in Figure 1. Thus only the physical quantities and the cycle clocks evolve continuously. Finally, computing flowpipes for clocks and other variables with constant derivatives can usually be done easier than for dynamics specified by general ODEs.

These observations gave us the idea to divide the variable set X into several disjoint subsets $X = X_1 \cup \ldots \cup X_n$ such that variables in the same subset X_i have some common properties relevant for reachability analysis. Once the variables are classified this way, we could try to modularise the reachability analysis computation by computing in the sub-spaces defined by the variable subsets, instead of computing in the global space. However, in order to compute reachability in the sub-spaces, the variables in different subsets must be independent in the sense that their evolutions do not directly influence each other. To be more formal, all predicates $\varphi \in Pred_X$ in the hybrid automaton definition must be decomposable to a conjunction $\varphi = \varphi_1 \wedge \ldots \varphi_n$ of predicates $\varphi_i \in Pred_{X_i}$ over the respective variable subsets X_i, and similarly for jump resets from $Pred_{X \cup X'}$ and flows from $Pred_{X \cup \dot{X}}$. If this condition holds then we call the subsets $X_1, \ldots, X_n$ themselves as well as variables from two different subsets *syntactically independent*.

Such a classification of the variable set X into syntactically independent subsets $X_1, \ldots, X_n$ allows us to represent (global) state sets $(l, p) \subseteq Loc \times \mathbb{R}^d$ by their projections $p \downarrow_{X_i} = p_i \subseteq \mathbb{R}^{|X_i|}$ to the sub-spaces; we call $(l, p_1, \ldots, p_n)$ the *projective representation* of (l, p) with respect to the variable separation $X_1, \ldots, X_n$. Note that the projective representation drops the connection between the sub-spaces and is therefore *over-approximative*, i.e., $p \subseteq p_1 \times \ldots \times p_n$ but in general $p \neq p_1 \times \ldots \times p_n$. One exception is the state set representation by *boxes*: the cross product of the projections of a box is the box itself, therefore the projective representation of boxes is exact.

Reachability computation based on variable set separation Given a separation of the variable set X into syntactically independent subsets $X_1,\ldots,X_n$ and projective representations based on this separation, we can over-approximate successors of a state set (l,p) by computing successors of its projective representation $(l,p_1,\ldots,p_n)$ in each sub-space modularly. As the computational effort for reachability analysis heavily increases with the dimension, this modular approach will help to reduce the running time. To explain why we need syntactical independence for sub-space computations, we first need a more formal description of how successor sets are computed:
(1) Reachability analysis computes for an initial set (l,p) the first flowpipe segment (l,Ω_0) that over-approximates all states reachable from p within a time interval $[0,\delta]$ in l as $\Omega_0 = (conv(p \cup e^{\delta A}p) \oplus \mathscr{B}) \cap Inv(l)$, where the flow in location l is $\dot{x} = Ax$, $e^{\delta A}$ is the matrix exponential for δA, $e^{\delta A}p$ are the states reachable from p at time point δ, $conv(\cdot)$ is the convex hull operator, $S_1 \oplus S_2 = \{a+b \mid a \in S_1 \wedge b \in S_2\}$ is the Minkowski sum of two sets, the bloating with the box $\mathscr{B}$ accounts for the non-linear behaviour between the time points 0 and δ, and $Inv(l)$ is the invariant for location l.
(2) Flowpipe segments (l,Ω_i) over-approximating the flowpipe within the time interval $[i\delta,(i+1)\delta]$ for $i > 0$ are computed by $\Omega_i = e^{\delta A}\Omega_{i-1} \cap Inv(l)$.
(3) For each jump e with source l, guard g, reset $x' = A'x$ and target location l', each flowpipe segment Ω_i is checked for possible successors along e by checking $(A'(\Omega_i \cap g)) \cap Inv(l')$ for emptiness. Non-empty successors are collected, possibly aggregated, and considered as initial state set(s) for location l'.

For each decomposition of X into syntactically independent variable sets $X_1,\ldots,X_n$, any flow $\dot{x} = Ax$ can be decomposed into $\wedge_{i=1}^n \dot{X}_i = A_i X_i$ (where we overload the notation X_i to also denote the sequence of variables in X_i). Similarly, $Inv(l) = \wedge_{i=1}^n Inv(l)_i$ with $Inv(l)_i \in Pred_{X_i}$. Furthermore, let $\mathscr{B}_i = \mathscr{B} \downarrow_{X_i}$ be the projection of $\mathscr{B}$ to X_i. Let $(l,p_0,\ldots,p_n)$ be the projective representation of a state set p,

$$\Omega_{0,i} = (conv(p_i \cup e^{\delta A_i} p_i) \oplus \mathscr{B}_i) \cap Inv(l)_i$$

and for each $j > 0$

$$\Omega_{j,i} = (e^{\delta A_i}\Omega_{j-1,i} \oplus \mathscr{B}_i) \cap Inv(l)_i\,.$$

Then $\Omega_j \subseteq \Omega_{j,1} \times \ldots \times \Omega_{j,n}$.

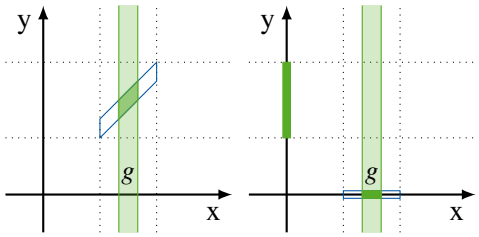


Figure 3: Intersection of a flowpipe segment with an invariant using global (left) and separated (right) variable sets.

The computations in the sub-spaces are precise as long as the initial set p is a box and the flowpipe resides inside the invariant, i.e., if p is a box and $\Omega_{m,i} \subseteq Inv(l)_i$ for all $1 \le m \le j$ and all $1 \le i \le n$ then $\Omega_j \downarrow_i = \Omega_{j,i}$.

However, syntactical independence does not imply semantical independence, as the different dimensions are usually still implicitly connected by the passage of time. If one of the projections runs out of a non-trivial invariant then the intersection with the (projection of the) invariant in a sub-space does not necessarily affect the computations in the other sub-spaces, thus the result might become over-approximative (see Figure 3). To increase precision, we can at least incorporate that if the projection of a flowpipe segment gets empty in one of the sub-spaces then the whole flowpipe segment gets empty: instead of $\Omega_{j,i}$ we use $\Omega'_{j,i}$ that is $\Omega_{j,i}$ if none of $\Omega_{j,k}$, $k = 1,\ldots,n$ is empty and the empty set otherwise.

For successors along jumps, the reachability computations in the sub-spaces work similarly. Also here, additional over-approximation might be introduced by intersections with guards and invariants in target locations, which we try to reduce by the above-described emptiness check.

As we use modular computations in sub-spaces, on the one hand our method speeds up reachability computations, but on the other hand it introduces additional over-approximations. Therefore, in our experiments we will thoroughly analyse the effect of our approach both to the running time as well as to the

over-approximation error. Besides the reduced computational effort, our method has further advantages. For example, state sets in the sub-spaces can be represented independently of each other, using different state set representations. We observed that the discrete variables can often be represented by boxes without serious over-approximation, whereas the plant dynamics requires a more precise representation, e.g. by support functions. Furthermore, for sub-spaces defined by clocks or by variables with constant derivatives one could use different, computationally less expensive techniques for computing flowpipes.

The algorithm Our reachability analysis algorithm based on variable set separation is presented in Algorithm 1. The input is a hybrid automaton H, a parameter δ that specifies the length of a time step in the flowpipe construction, a global time horizon T and a jump depth D that specify upper bounds on the total time duration respectively on the total number of jumps in the considered execution paths, and an aggregation flag that specifies whether aggregation should be applied to the successors of segments of a flowpipe along a jump. The algorithm outputs a set of flowpipe segments R, where the union of the segments in R is an over-approximation of the set of states that are reachable in H within the given time and jump bounds.

In a preprocessing step, we separate the variable set into three syntactically independent subsets of discrete variables, clocks, and the rest (lines 2-4). We have chosen this variable set separation as it seems to be practically helpful in our experiments, but other separation criteria could be defined, too. A state set representation can be chosen for each sub-space independently (line 5); for readability, in the algorithm we do not distinguish between state sets and their representations syntactically.

The invariant conditions for the initial states are checked for each initial set and each sub-space independently in the lines 8-13 and if the intersections of the initial sets with the invariants are non-empty in all sub-spaces then they are added to a set P of state sets, whose successors need to be determined. Additionally to the location and the projective representation of state sets, we attach to the state sets the time interval within which the represented states can be reached.

As long as P is not empty, we choose a state set $p \in P$. Before computing its flowpipe, we determine the set of jumps with p's location as source. As the values of discrete transitions do not change during time evolution, we can skip those jumps whose guard is violated by the initial values of the discrete variables, and store the remaining ones in the set E (line 16). The sets P^e will be used to collect non-empty successors from different flowpipe segments along the jumps $e \in E$.

Next we compute the segments of p's flowpipe as explained previously (lines 19-23). Instead of one single flowpipe of dimension d, our algorithm computes lower-dimensional flowpipes in the sub-spaces using a common time step size and a global time horizon (line 19) to be able to connect the flowpipe segments computed in the sub-spaces. The variable *isFirst*, initialised in line 17, is used to remember whether the flowpipe segment to be computed next is the first one (as the first one needs special handling).

Once the flowpipe segments are computed, their jump successors are determined and collected in the sets P^e for each outgoing jump $e \in E$ (lines 25-29), if aggregation is activated then they are aggregated (per jump, line 34), and finally added to P for further iterative successor computation. Note that the flowpipe as well as the jump successor computations in the sub-spaces are time-synchronised: if a successor set gets empty in one of the sub-spaces then the computation is stopped.

5 Experimental Results

All computations were carried out on an Intel Core i7 CPU with 8 cores and 16 GB RAM. We used the time step size $\delta = 0.01$ and unlimited jump depth. To be able to express a global time horizon, each model is equipped with a global clock.

Input: Hybrid system model $H = (Loc, X, Flow, Inv, Edge, Init)$, time step $\delta \in \mathbb{Q}_{\geq 0}$, global time horizon $T \in \mathbb{Q}_{\geq 0}$, jump depth $D \in \mathbb{N}_{\geq 0}$, aggregation flag $aggregation \in \{0,1\}$.

Output: An over-approximation $\{(l,x) | (l, p_{disc}, p_{clock}, p_{rest}, [t_1,t_2]) \in R \wedge x \in p_{disc} \times p_{clock} \times p_{rest}\}$ of the states reachable within the given bounds.

X_{disc} := maximal set of variables from X with derivatives 0 that are syntactically independent from $X \setminus X_{disc}$;
X_{clock} := maximal set of variables from X with derivatives 1 that are syntactically independent from $X \setminus X_{clock}$;
$X_{rest} := X \setminus (X_{disc} \cup X_{clock})$;
choose a representation type for each of X_{disc}, X_{clock}, and X_{rest};
bring each predicate φ in H to an equivalent form $\varphi_{disc} \wedge \varphi_{clock} \wedge \varphi_{rest}$, where each φ_i, $i \in \{disc, clock, rest\}$, is a predicate from $Pred_{X_i}$ resp. $Pred_{X_i \cup X_i'}$ (jump resets) resp. $Pred_{X_i \cup \dot{X}_i}$ (flows);
$P := \emptyset$; $R := \emptyset$;
foreach *location* $l \in Loc$ **do**
    let $p_i := Init(l)_i \cap Inv(l)_i$ for each $i \in \{disc, clock, rest\}$;
    if $p_{disc} \neq \emptyset \wedge p_{clock} \neq \emptyset \wedge p_{rest} \neq \emptyset$ **then**
        add $(l, p_{disc}, p_{clock}, p_{rest}, [0,0])$ to P
    end
end
while $P \neq \emptyset$ **do**
    choose $p = (l, p_{disc}, p_{clock}, p_{rest}, [t_1,t_2]) \in P$ and remove p from P;
    $E := \{(e, p^e_{disc}) \mid e = (l,g,r,l') \in Edge \wedge p^e_{disc} = jump(p_{disc}, g_{disc}, r_{disc}, Inv(l')_{disc}) \neq \emptyset\}$;
    $isFirst := true$; **foreach** $(e, p^e_{disc}) \in E$ **do** $P^e := \emptyset$;
    while *true* **do**
        `// Compute flowpipe, considering also the invariant of the location`
        if $t_1 < T$ **then**
            $p_{clock} := flow(p_{clock}, Flow(l)_{clock}, Inv(l)_{clock}, \delta, isFirst)$; **if** $p_{clock} = \emptyset$ **then** break;
            $p_{rest} := flow(p_{rest}, Flow(l)_{rest}, Inv(l)_{rest}, \delta, isFirst)$; **if** $p_{rest} = \emptyset$ **then** break;
            $t_2 := t_2 + \delta$; **if** $\neg isFirst$ **then** $t_1 := t_1 + \delta$;
        end
        $R := R \cup \{(l, p_{disc}, p_{clock}, p_{rest}, [t_1,t_2])\}$;
        `// Compute jump successors`
        foreach $((l,g,r,l'), p^e_{disc}) \in E$ **do**
            $p^e_{clock} := jump(p_{clock}, g_{clock}, r_{clock}, Inv(l')_{clock})$; **if** $p^e_{clock} = \emptyset$ **then** continue;
            $p^e_{rest} := jump(p_{rest}, g_{rest}, r_{rest}, Inv(l')_{rest})$; **if** $p^e_{clock} = \emptyset$ **then** continue;
            add $(l', p^e_{disc}, p^e_{clock}, p^e_{rest}, [t_1,t_2])$ to P^e;
        end
        $isFirst := false$; **if** $t_1 \geq T$ **then** break;
    end
    foreach $e \in E$ **do**
        if $P^e \neq \emptyset$ **then**
            if *aggregation* **then** $P := P \cup \{aggregate(P^e)\}$; **else** $P := P \cup P^e$
        end
    end
end
return R

Algorithm 1: Reachability analysis algorithm based on variable set separation for hybrid automata.

Table 1: Model sizes of the benchmarks.

Benchmark	Type	#variables			#modes		#jumps
		disc.	clocks	rest	controller	plant	
Leaking tank	original	0	0	12	8	3	34
	timed	0	2	10	8	3	34
	discrete	9	0	3	8	3	34
	timed & discrete	9	2	1	8	3	34
Two tanks	original	0	0	22	20	14	296
	timed	0	3	19	20	14	296
	discrete	17	0	5	20	14	296
	timed & discrete	17	3	2	20	14	296
Thermostat	original	0	0	8	6	2	18
	timed	0	2	6	6	2	18
	discrete	5	0	3	6	2	18
	timed & discrete	5	2	1	6	2	18

Benchmarks For our experiments we used three well-known benchmarks, which we slightly modified by adding model components for PLC controllers. Besides increasing the number of modes, these extensions add variables with discrete behaviour (i.e. with zero derivatives) to model the actuators and sensors of the plant and the input, output, and local variables of the controller. Furthermore, one clock variable is added for each introduced PLC controller to model the cycle time, and one discrete variable to store the controller mode. In our experiments we compare the analysis of the benchmarks without variable separation ("original") with variable-set-separation-based analysis separating only clocks ("timed"), only discrete variables ("discrete"), and both ("timed & discrete"). The sizes of the models are shown in Table 1. The modified versions of the benchmarks are accessible as part of our benchmark collection [2]. A binary of our implementation can be found at [12].

Leaking tank This benchmark models a water tank which leaks, i.e., it has a constant outflow. The tank can be refilled from an unlimited external resource with a constant inflow that is larger than the outflow. The PLC controller triggers refilling (by switching a pump on) if a sensor indicates a low water level ($h \leq 6$). If the water level is high ($h \geq 12$) the controller stops refilling (switches the pump off). Adding the controller introduces two controller input variables for low and high water levels, variables for the actuator (pump) state in the plant and the controller, and a variable to store the controller mode. Furthermore, a new clock is added to model the PLC cycle time. Besides the controller we also model a user which can manually switch the pump on and off as far as the water level allows it. In our implementation, the user constantly toggles between the pump states on and off. We analyse the system behaviour over a global time horizon of 40 seconds using a PLC cycle time of 2 seconds.

Two tanks This benchmark models the water levels of two water tanks in a closed system. Each tank has a constant inflow and a constant outflow. The tanks are connected via pipes, such that the amount of water outflow of the first tank is equal to the inflow of the second tank and vice versa. One pump per pipe allows to enable/disable the water flow. We add a controller to the two tank system that controls the pumps. A pump is switched off if the water level of the source tank is low ($h \leq 8$) or if the water level of the target tank is high ($h \geq 32$). Each time a pump is switched off by the controller, the other pump is switched on to balance the water levels in the tanks. The introduction of the controller adds variables to model sensing low and high water levels of both tanks and variables to model the actuator (pump) states in the plant and the controller. Moreover, we add a variable to store the controller mode and a new clock

Table 2: Benchmark results for different separation set-ups. Running times are in seconds, time-out (TO) was 20 minutes, in brackets we list the number of flowpipes computed.

			HyPro				SpaceEx
Benchmark	Rep.	Agg	original	timed	discrete	timed & discrete	original
Leaking tank	box	agg	2.70 (662)	2.08 (662)	1.06 (662)	1.13 (662)	3.67 (200)
	box	none	2.62 (662)	2.09 (662)	1.06 (662)	1.13 (662)	3.82 (200)
	sf	agg	TO (18)	TO (28)	161.12 (662)	37.03 (662)	448.3 (425)
	sf	none	TO (583)	1044.97 (662)	19.49 (662)	5.84 (662)	444.82 (425)
Two tanks	box	agg	4.39 (470)	2.60 (470)	0.97 (470)	1.15 (470)	5.49 (195)
	box	none	4.46 (470)	2.68 (470)	1.02 (470)	1.16 (470)	5.53 (195)
	sf	agg	TO (4)	TO (4)	900.11 (470)	329.80 (470)	TO (171)
	sf	none	TO (54)	TO (64)	35.04 (470)	14.64 (470)	TO (172)
Thermostat	box	agg	0.07 (95)	0.09 (95)	0.06 (95)	0.06 (95)	0.57 (95)
	box	none	0.11 (95)	0.09 (95)	0.06 (95)	0.06 (95)	0.57 (95)
	sf	agg	35.87 (95)	22.69 (95)	1.17 (95)	0.29 (95)	9.89 (84)
	sf	none	30.41 (95)	20.19 (95)	1.18 (95)	0.30 (95)	9.91 (84)

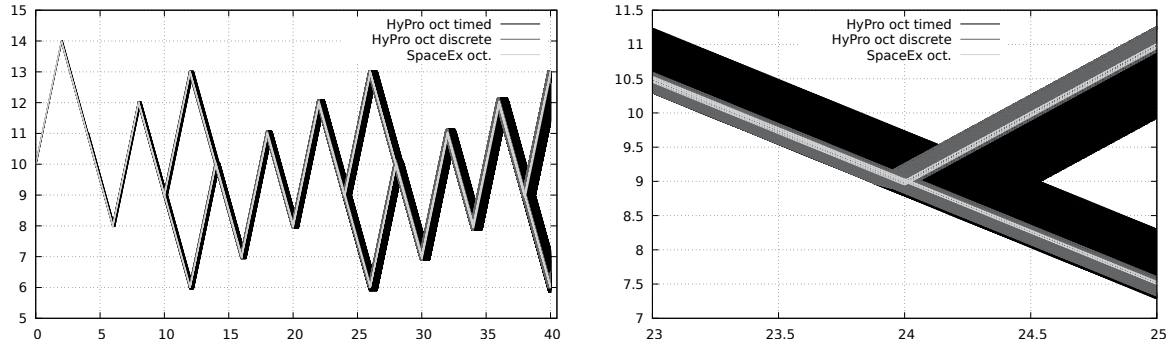


Figure 4: SpaceEx and HyPro results on the leaking tank benchmark with support function representation (using a regular octagonal (oct) template for evaluation), when HyPro separates either only clocks or only discrete variables.

to model the PLC cycle time. Again, we model a user which switches the pumps manually on or off as far as the water levels allow it. We implemented a user that toggles the state of each pump in each PLC cycle. The global time horizon and a PLC cycle time were set to 20 seconds respectively 1 second.

Thermostat In this benchmark a heater with a thermostat controller is modelled. Initially, the temperature is $T = 20°C$ and the heater is on. The controller keeps the temperature T between $16°C$ and $24°C$. The heater is switched off if the temperature rises above $23°C$ and it is switched on at a temperature below $18°C$. Adding a controller to the model introduces new variables for the low and high temperature sensors in the controller, a variable for the actuator (heater) state in the plant and the controller, and a variable to store the controller mode. Additionally, we introduce a new clock for the cycle time of the PLC. The global time horizon is 10 seconds and the PLC cycle time is 0.5 seconds.

Results We implemented our algorithm using the HyPro library and evaluated it on the above benchmarks. Table 2 shows results from our tool and SpaceEx version 0.9.8f. In our tool we used boxes and support functions (evaluated in 8 directions) to represent state sets, whereas in SpaceEx we used support functions with 4 and 8 directions, as SpaceEx does not support explicit box representations.

The HYPRO and SPACEEX results are not fully comparable because SPACEEX implements a fixed-point detection algorithm but HYPRO does not. The leaking tank benchmark as well as the two tank benchmark both cause branching in the execution paths which are merged later (see Figure 4). Our implementation does not recognise the merging of these paths and fully computes each branch independently. Thus HYPRO needs to compute a higher number of flowpipes (given in brackets behind the running times in Table 2) than SPACEEX. Another difference is that in HYPRO we varied the state set representation for the continuous sets between boxes and support functions (similarly to SPACEEX) but used boxes for the discrete and the clock variable sets in all settings.

Using variable separation clearly improves the running times, due to computations in lower-dimensional sub-spaces. However, we can also observe on the Figures 4 and 5, which show plots for the detected reachable regions for the leaking tank and the thermostat, that separating the clock variables (which measure the cycle time and the global time) introduces a slight over-approximation.

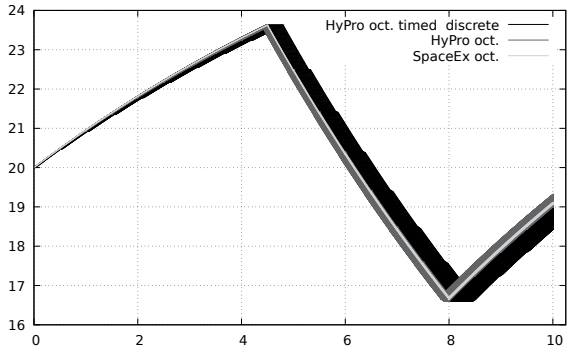


Figure 5: SPACEEX and HYPRO results on the thermostat benchmark with support function representation (using an octagonal (oct) template for evaluation), when HYPRO separates clocks as well as discrete variables.

The influence of the discrete variable separation is in general larger than the influence of a clock separation, probably because in our benchmarks the discrete variables outnumber the clocks. Nonetheless a separation of clocks already shows a speed-up of about 30%. As mentioned before, we used boxes as a state set representation for the set of discrete variables, which does not introduce any further over-approximation error, as the discrete variables themselves are all syntactically independent. We can observe that using boxes as a state set representation, our implementation outperforms SPACEEX (even when a lot more flowpipes are computed), which is expected, as boxes in general require less computational effort than support functions (evaluated in 4 directions) in reachability analysis.

In HYPRO, aggregation causes longer running times because in the current implementation aggregation is realised by a conversion of the single sets (which are to be aggregated) to polytopes, which is computationally expensive, especially in higher dimensions.

6 Conclusion

In this paper we presented an approach to reduce the computational effort in the reachability analysis of hybrid systems for certain applications. Our experimental results indicate that even state-of-the art reachability analysis tools struggle to analyse high-dimensional models with relatively simple dynamics, which are common in the application area of controlled plants.

In general controlled plants are composed of many single components such as the set of controllers or the physical quantities of the plant. A naive approach models each of these components and the full model is the result of a parallel composition of the single components. Even the relatively simple examples used in Section 5 yield large models which put state-of-the-art reachability analysis tools to their limits. To increase scalability, domain-specific knowledge helps to create more sophisticated and smaller models. For example knowing that PLC computation as well as the plant's behaviour do not

interfere during a PLC cycle already allows to prohibit arbitrary switching between the controller and the plant, which reduces the model complexity.

In contrast to common benchmarks for hybrid systems, our plant models exhibit a large number of discrete variables accounting for the controller's behaviour. Currently available tools do not distinguish between the different dynamics of variables, thus discrete variables usually are treated as continuous variables and unnecessarily increase the dimension of the state space.

Our approach allows to split variable sets according to their dynamics, which has a positive effect on the running times, as reachability analysis algorithms can be protected from working in high-dimensional spaces. We can observe that the distribution of variables to the different sets has a high influence on the computation time. We expect that splitting the set of continuous variables (if applicable) into multiple, independent sets with fewer variables each will result in the best results regarding computation time. Furthermore in applications with several PLC controllers, each control program operates independently, which allows to build separate variable sets for each controller. Depending on the dimension of the individual sets and the associated dynamics for the contained variables, utilizing individual state set representations can be beneficial. As state sets for independent discrete variables are always hyper-rectangles, using boxes instead of other, computationally more expensive state set representations has shown great improvements in terms of runtime. For higher dimensional state sets support functions can be expected to perform better than other state set representations.

In our application scenario, syntactically independent variable sets are directly given. In general, this is not necessarily the case for hybrid system models. Transforming the state space can help to identify independent variable sets and allows to apply the presented approach to systems where the independent variable sets are not obvious.

As to future work, we will improve our implementation by adding fixed-point detection and a more sophisticated implementation for state set aggregation. Second, we will embed the presented approach into a CEGAR framework to refine potentially unsafe paths. Finally, we also work on parallelisation approaches for flowpipe computations.

References

[1] Matthias Althoff & John M. Dolan (2014): *Online verification of automated road vehicles using reachability analysis*. *IEEE Transaction on Robotics* 30(4), pp. 903–918.

[2] *Benchmarks of continuous and hybrid systems*. Available at `http://ths.rwth-aachen.de/research/projects/hypro/benchmarks-of-continuous-and-hybrid-systems/`.

[3] Olivier Bournez, Oded Maler & Amir Pnueli (1999): *Orthogonal polyhedra: Representation and computation*. In: *Proc. HSCC'99*, *LNCS* 1569, Springer, pp. 46–60.

[4] Xin Chen, Erika Ábrahám & Sriram Sankaranarayanan (2013): *Flow*: An analyzer for non-linear hybrid systems*. In: *Proc. CAV'13*, *LNCS* 8044, Springer, pp. 258–263.

[5] Xin Chen & Sriram Sankaranarayanan (2016): *Decomposed reachability analysis for nonlinear systems*. In: *Proc. RTSS'16*, IEEE Computer Society Press, pp. 13–24.

[6] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp & Erika Ábrahám (2015): *SMT-RAT: An open source C++ toolbox for strategic and parallel SMT solving*. In: *Proc. SAT'15*, *LNCS* 9340, Springer, pp. 360–368.

[7] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang & Oded Maler (2011): *SpaceEx: Scalable verification of hybrid systems*. In: *Proc. CAV'11*, *LNCS* 6806, Springer, pp. 379–395.

[8] Antoine Girard (2005): *Reachability of uncertain linear systems using zonotopes*. In: *Proc. HSCC'05*, *LNCS* 3414, Springer, pp. 291–305.

[9] Willem Hagemann, Eike Möhlmann & Astrid Rakow (2014): *Verifying a PI controller using SoapBox and Stabhyli: Experiences on establishing properties for a steering controller*. In: *Proc. ARCH'14*, *EPiC Series in Computer Science* 34, EasyChair.

[10] Thomas A. Henzinger (1996): *The theory of hybrid automata*. In: *Proc. LICS'96*, IEEE Computer Society Press, pp. 278–292.

[11] *HyCreate*. Available at `http://stanleybak.com/projects/hycreate/hycreate.html`.

[12] *HyPro Project website*. Available at `http://ths.rwth-aachen.de/research/projects/hypro/`.

[13] HyReach. Available at `https://embedded.rwth-aachen.de/doku.php?id=en:tools:hyreach`.

[14] Colas Le Guernic & Antoine Girard (2010): *Reachability analysis of linear systems using support functions*. *Nonlinear Analysis: Hybrid Systems* 4(2), pp. 250–262.

[15] Andrew Makhorin: *GNU Linear Programming Kit home page*. Available at `http://www.gnu.org/software/glpk/glpk.html`.

[16] Ramon E. Moore, Ralph Baker Kearfott & Michael J. Cloud (2009): *Introduction to interval analysis*. SIAM.

[17] Leonardo M. de Moura & Nikolaj Bjørner (2008): *Z3: An efficient SMT solver*. In: *Proc. TACAS'08*, *LNCS* 4963, Springer, pp. 337–340.

[18] Johanna Nellen (2016): *Analysis and synthesis of hybrid systems in engineering applications*. Ph.D. thesis, RWTH Aachen University, Aachen. Available at `https://publications.rwth-aachen.de/record/680323`.

[19] Stefan Schupp, Erika Abraham, Ibtissem Ben Makhlouf & Stefan Kowalewski (2017): *HyPro: A C++ library for state set representations for hybrid systems reachability analysis*. In: *Proc. NFM'17*, LNCS, Springer. To appear.

[20] Roland Wunderling (1996): *Paralleler und objektorientierter simplex-algorithmus*. Ph.D. thesis, Technische Universität Berlin.

[21] Günter M. Ziegler (1995): *Lectures on polytopes*. 152, Springer.

SEA-PARAM: Exploring Schedulers in Parametric MDPs

Sebastian Arming
University of Salzburg, Austria

Ezio Bartocci
TU Wien, Austria

Ana Sokolova
University of Salzburg, Austria

We study parametric Markov decision processes (PMDPs) and their reachability probabilities "independent" of the parameters. Different to existing work on parameter synthesis (implemented in the tools PARAM and PRISM), our main focus is on describing different types of optimal deterministic memoryless schedulers for the whole parameter range. We implement a simple prototype tool SEA-PARAM that computes these optimal schedulers and show experimental results.

1 Introduction

A Markov decision process (MDP) [2] is a state-based Markov model in which a state can perform one of the available action-labeled transitions after which it ends up in a next state according to a probability distribution on states. The choice of a transition to take is nondeterministic, but once a transition is chosen the behaviour is probabilistic. MDPs provide a valuable mathematical framework to solve control and dependability problems in a wide range of applications, including the control of epidemic processes [20], power management [24], queueing systems [28], and cyber-physical systems [21]. MDPs are also known as reactive probabilistic systems [19, 10] and closely related to probabilistic automata [27].

In this paper, we study *parametric* Markov decision processes (PMDPs) [6, 12]. These are models in which (some of) the transition probabilities depend on a set of parameters. An example of an action in a PMDP is tossing a (possibly unfair) coin which lands heads with probability p and tails with probability $1-p$ where $p \in [0,1]$ is a parameter. Hence, a PMDP represents a whole family of MDPs—one for each valuation of the parameters.

We study reachability properties in PMDPs. To explain what we do exactly, let us take a step back. If an MDP can only perform a single action in each state, then it is a Markov chain (MC). If a PMDP can perform a single action in each state, then it is a *parametric* Markov chain (PMC). Given a start state and a target state in a PMC, the probability of reaching the target from the start state is a *rational function* in the set of parameters. This rational function can be elegantly computed by the method of Daws [8] providing arithmetic interpretation for regular expressions. The method has been further developed and efficiently implemented in the tool PARAM [13, 11, 12].

Clearly, there is no such thing as the probability of reaching a target state from a starting state in an MDP: such a reachability probability depends on which actions were taken along the way, i.e., of how the nondeterministic choices were resolved. What is usually of interest though are the min/max reachability probabilities, i.e., among all possible ways to resolve the nondeterministic choices, those that provide minimal/maximal probability of reaching a state. Nondeterministic choices are resolved using schedulers or policies, and luckily when it comes to min/max reachability probabilities *simple* schedulers suffice [2]. Simple schedulers are deterministic and memoryless, i.e., history independent. Given an MDP, a simple scheduler induces an MC, and the reachability probabilities under this scheduler are simply the reachability probabilities of the induced MC.

Submitted to:
QAPL 2017

With PMDPs, the situation is even more delicate. The probability of reaching a target state from a starting state depends on the scheduler, i.e., on how the nondeterministic choices were resolved, as well as on the values of the parameters. The full reachability picture looks like a sea — each scheduler imposes a rational function — a *wave* — over the parameter range; the sea then consists of all the waves.

There are two possible scenarios of interest:

(1) We have access to the parameters.

(2) We have no access to the parameters, they represent uncertainty or noise or choices of the environment.

In case (1), parameter synthesis is the problem to solve. The parameter synthesis problem comes in two flavours: (a) Find the parameter values that maximise / minimise the reachability probability; (b) For each value of the parameters, find the max/min reachability probability. These are the problems that have attracted most attention in the analysis of PMCs [4, 14, 23, 9, 25] and PMDPs [6, 12], see Section 2 for more details.

In this paper, we consider case (2) and propose solutions for imposing bounds on the reachability probabilities throughout the whole parameter range.

In particular we:

- Start by enumerating all simple schedulers and computing their corresponding rational functions.
- Identify classes of optimal schedulers, for different problems of interest, see Section 5.
- Provide a tool that computes optimal schedulers in each of the classes for a given PMDP, see Section 7.

We admit that we take upon this task knowing that it is computationally hard. Already the number of simple schedulers is exponential in the number of states of the involved PMDP. Optimisation is in general also hard (computing maxima, minima, and integrals of the involved rational functions), see Section 7 for references and more details. Nevertheless, the analysis that we aim at is not an online analysis, but rather a preprocessing step, and even if it may only work on small examples, it provides insight in the behaviour of a system and its schedulers.

Our tool extensively uses the state-of-the-art tools PARAM1 [13, 11] and PARAM2 [12] for efficient computation of the rational functions, see Section 2 and Section 7 for details. Once we have all schedulers and their respective waves, we feed the waves to a numerical tool that allows us to calculate the optimal schedulers.

We experiment with a class of examples describing the behaviour of a robot walking in a labyrinth grid. Each position on the grid is a state of the MDP, and the available actions are N, S, E, and W, describing the directions (north, south, east, and west) of a possible move. Not all actions are available in every state. Some states represent holes (sinks) in which no action is available, others correspond to border-positions, and hence some actions are disabled. See Section 6 for further description of our class of examples. One small concrete example in this class is the PMDP describing a 2×2 grid with a sink at position $(1,2)$. In $(1,1)$ the actions N and E are available, in $(2,1)$ the actions N and W, and in $(2,2)$, the actions S and W.

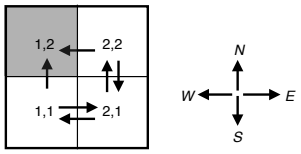
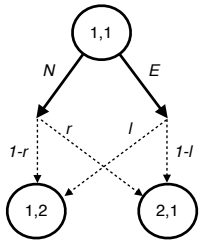
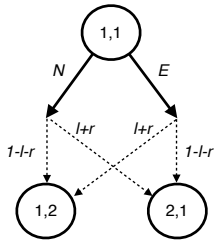


Figure 1: 2×2 Labyrinth with sink at (1,2)

The model is parametric with two parameters l and r having the following meaning: In a state s with an enabled action M, the robot moves forward with probability $1 - l - r$ to its intended state s', or ends

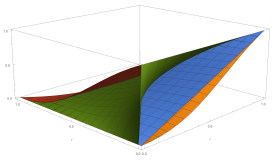


(a) Fixed failure

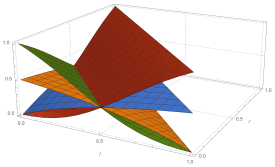


(b) Fixed success

Figure 2: The behavior of state $(1,1)$



(a) Fixed failure



(b) Fixed success

Figure 3: The sea of reachability probabilities from state $(1,1)$ to state $(2,2)$

up in the state s_l left of s (in the direction of the move) with probability l, and in the state s_r right of s (in the directions of the move) with probability r, provided both states s_l and s_r exist. If one of s_l or s_r does not exist, then we consider two scenarios:

- Fixed failure: In this scenario, the probability to the existing state s_l or s_r remains l or r, but the probability of reaching s' increases to $1-l$ or $1-r$, respectively.
- Fixed success: Here, the probability to s' remains the same, and the probability to s_l or s_r (whichever exists) becomes $l+r$.

Figure 2 shows the behaviour of state $(1,1)$ in both scenarios and Figure 3 pictures all waves – rational functions corresponding to reachability of target state $(2,2)$ from the starting state $(1,1)$ – in each of the two scenarios.

As we can see even in this small example, the reachability probability varies through the parameter range and significantly depends on the chosen scheduler. For different purposes, different schedulers may be preferred. We identify ten classes of optimal schedulers that may be preferred in certain cases. For example, one may wish to use a scheduler that guarantees highest reachability probability for any value of the parameter, if such a scheduler exists. We call such a scheduler *dominant*. That would be the red scheduler in Figure 3a. In Figure 3b there is no dominant scheduler. However, one may prefer a scheduler that reaches the maximum value (reachability probability 1 in this case) for some value of the parameter. We call such schedulers *optimistic*. In Figure 3b the red and the green schedulers are optimistic. In addition, one may prefer the red over the green, as under the assumption of a uniform distribution of parameters, the red has a higher value over a larger parameter region — we call such schedulers *expecation* schedulers. For yet another purpose, one may prefer the yellow or the blue scheduler, as its difference in reachability probabilities is the smallest — the *bound* scheduler according to our definition.

See Section 5 for the exact definitions of all classes of optimal schedulers.

Finally, we mention that we started analysing simple schedulers as a first step in the scheduler analysis of PMDPs. As we discuss in Section 8, we are aware that optimal schedulers (in our classes) need not be simple. Nevertheless, we believe that conquering simple schedulers is an important first step.

2 Related Work

In the last decade there has been a growing interest in studying parametric probabilistic models [8, 18] where some of the probabilities (or rates) in the models are not known a-priori. These models are very useful when certain quantities (e.g. fault rates, packet loss ratios, etc.) are partially available (which is often the case) or unavailable at the design time of a system. In his seminal work [8], Daws studies the problem of symbolic model checking of parametric probabilistic Markov chains. He provides a method based on regular expressions extraction and state elimination to symbolically express the probability to reach a target state from a starting state as a multivariate rational function whose domain is the parameter space. This technique was further investigated and implemented in the PARAM1 and PARAM2 tools [13, 11] and it is now also included in the popular PRISM model checker [17]. In this context, the problem of *parameter synthesis* for a parametric Markov chain consists of solving a constrained nonlinear optimisation problem where the objective function is a multivariate rational function representing the probability to satisfy a given reachability property depending on the parameters. As discussed in [16, 18] and later in this paper, when the order of these multivariate rational functions is high, such constrained optimisation problem can become computationally very expensive.

In [4] Bartocci et al. introduce a complementary technique to *parameter synthesis*, called *model repair*, that exploits the PARAM1 tool in combination with a nonlinear optimisation tool to find automatically the minimal change of the parameter values required for a model to satisfy a given reachability property that the model originally violates. In this case the problem boils down to solving a nonlinear optimisation program having for *objective function* an L2-norm (quadratic and indeed suitable for convex optimisation) measuring the distance between the original parameter values and the new ones and having as *constrains* the multivariate rational function associated with the reachability property.

Recently, more sophisticated symbolic parameter synthesis techniques [14, 23, 9, 25] based also on SMT solvers and greedy approaches [23] have further improved this field of research. At the same time statistical-based approaches leveraging powerful machine learning techniques [5, 3] have been shown to provide better scaling of the model checking problem for large parametric continuous Markov chains when the number of parameters is limited and the event of satisfying the property is not rare.

All the aforementioned methods do not natively support nondeterministic choice and are indeed not suitable for solving parametric Markov decision processes. The parametric model checking problem for this class of models has been addressed so far in the literature using two complementary methods [6, 12].

The first method, implemented in PARAM2 [12], is a *region-based approach* where the parameter space is divided into regions representing sets of parameter valuations. For each region, lower and upper bounds on optimal parameter values are computed by evaluating the edge points of the regions. Given a desired level of precision for the result as input, the algorithm decides whether to further split the region into smaller ones to be explored or to terminate with the intervals found. The correctness and the termination of this algorithm is guaranteed only under certain assumptions as discussed in [12]. The second method [6] is a sampling-based approach (i.e., based on sampling methods like the Metropolis-Hastings algorithm, particle swarm optimisation, and the cross-entropy method) that are used to search the parameter space. These heuristics usually do not guarantee that global optimal parameters will be found. Furthermore, when the regions of the parameters satisfying a requirement are very small, a large

amount of simulations is required.

We just became aware of a very recent work of Cubuktepe et al. [7] (to appear in TACAS'17) where the authors consider the problem of parameter synthesis in parametric Markov decision processes using signomial programs, a class of nonconvex optimisation problems for which it is possible to provide suboptimal solutions.

3 Markov Chains and Markov Decision Processes

Definition 1 (Markov chain)**.** A (discrete-time) Markov chain (MC) is a pair $M = (S,P)$ where:

- S is a countable set of states, and
- $P : S \times S \to [0,1]$ is a transition probability function such that for all s in S, $\sum_{s' \in S} P(s,s') = 1$. ◇

Given an MC $M = (S,P)$ and two states $s,t \in S$, we denote the probability to reach t from s by $\Pr^M(s,t)$. If M is clear from the context, we will omit the superscript in the reachability probability.

We next present the definition of an MDP without atomic propositions and rewards, as they do not play a role for what follows.

Definition 2 (Markov decision process)**.** A (discrete-time) Markov Decision Process (MDP) is a triple $M = (S, \mathrm{Act}, P)$ where:

- S is a countable set of states,
- Act is a set of actions,
- $P : S \times \mathrm{Act} \times S \to [0,1]$ is a transition probability function such that forall s in S and a in Act we have $\sum_{s' \in S} P(s,a,s') \in \{0,1\}$. ◇

In this paper we only consider finite MCs and MDPs, that is MCs and MDPs in which the set of states S (and actions Act) is finite. If needed, we may also specify a distinguished initial state $s_0 \in S$ in an MC or an MDP.

An action a is *enabled* in an MDP state s iff $\sum_{s' \in S} P(s,a,s') = 1$. We denote by $\mathrm{Act}(s)$ the set of enabled actions in state s. It is often required that $\mathrm{Act}(s) \neq \emptyset$ in an MDP, but we omit this requirement. A state s for which $\mathrm{Act}(s) = \emptyset$ is called a *sink*. A *simple scheduler* resolves the nondeterministic choice, selecting at each non-sink state s one of the enabled actions $a \in \mathrm{Act}(s)$. A synonym for a simple scheduler is deterministic memoryless/history-independent scheduler.

Definition 3 (Simple scheduler)**.** Given an MDP $M = (S, \mathrm{Act}, P)$, a simple scheduler ξ of M is a function $\xi : S \to \mathrm{Act} + 1$ where $1 = \{\bot\}$ and $+$ denotes disjoint union, satisfying $\xi(s) \in \mathrm{Act}(s)$ for all $s \in S$ such that $\mathrm{Act}(s) \neq \emptyset$, and $\xi(s) = \bot$ otherwise. ◇

Definition 4 (Scheduler-induced Markov chain)**.** Let ξ be a simple scheduler of an MDP M. Then the ξ-induced Markov chain is the Markov chain $M_\xi = (S, P_\xi)$ where $P_\xi(s,t) = P(s, \xi(s), t)$ if $\xi(s) \neq \bot$ and $P_\xi(s,s) = 1$ otherwise. ◇

Note that in this work we only consider simple schedulers. This justifies our nonstandard (and much simpler) definition of an induced Markov chain. From now on we will sometimes simply say *scheduler* for a simple scheduler.

Definition 5 (Maximum/Minimum reachability probabilities)**.** Given an MDP $M = (S, \mathrm{Act}, P)$ and two states $s,t \in S$, the maximum reachability probability from s to t is

$$\Pr^M_{\max}(s,t) = \max_{\xi} \Pr^{M_\xi}(s,t),$$

and similarly, the minimum reachability probability from s to t is given by

$$\Pr^{M}_{\min}(s,t) = \min_{\xi} \Pr^{M_\xi}(s,t),$$

where ξ ranges over all simple schedulers. We call a scheduler ξ a *maximal (minimal) scheduler from s to t* iff $\Pr^{M_\xi}(s,t)$ is the maximal (minimal) reachability probability from s to t. ◇

4 Parametric MCs and MDPs

We first recall the notion of a rational function (following [13, 11], with a small restriction). Let $V = \{x_1,\ldots,x_n\}$ be a fixed set of variables. An *evaluation* is a function $v\colon V \to \mathbb{R}$. A *polynomial* over V is a function

$$g(x_1,\ldots,x_n) = \sum_{i_1,\ldots,i_n} a_{i_1,\ldots,i_n} x_1^{i_1}\cdots x_n^{i_n},$$

where $i_j \in \mathbb{N}$ for $1 \leq j \leq n$ and each $a_{i_1,\ldots,i_n} \in \mathbb{R}$. A *rational function* over V is a quotient

$$f(x_1,\ldots,x_n) = \frac{g_1(x_1,\ldots,x_n)}{g_2(x_1,\ldots,x_n)}$$

of two polynomials g_1 and g_2 over V. By $\mathscr{F}_V$ we denote the set of rational functions over V. Hence, a rational function is a symbolic representation of a function from $\mathbb{R}^n$ to $\mathbb{R}$. Given $f \in \mathscr{F}_V$ and an evaluation v, we write $f\langle v\rangle$ for $f(v(x_1),\ldots,v(x_n))$.

It is now straightforward to extend MCs and MDPs with parameters [8, 18, 13, 11]. Again, we only consider finite models.

Definition 6 (Parametric Markov chain)**.** A parametric (discrete-time) Markov chain (PMC) is a triple $M = (S,V,P)$ where:

- S is a finite set of states,
- V is a finite set of parameters, and
- $P : S \times S \to \mathscr{F}_V$ is the parametric probability transition function. ◇

Given a PMC $M = (S,V,P)$, a valuation v of the parameters induces an MC $M_v = (S,P_v)$ where $P_v(s,s') = P(s,s')\langle v\rangle$ for all $s,s' \in S$, if for all s in S we have $\sum_{s'\in S} P(s,s')\langle v\rangle = 1$. If a valuation v induces a Markov chain on M, then we call v *admissible*. The set of all admissible valuations for M is the *parameter space* of M.

Similarly, we define parametric MDPs.

Definition 7 (Parametric Markov Decision Process)**.** A parametric (discrete-time) Markov Decision Process (PMDP) is a tuple $M = (S,\mathrm{Act},V,P)$ where:

- S is a finite set of states,
- Act is a finite set of actions,
- V is a finite set of parameters, and
- $P : S \times \mathrm{Act} \times S \to \mathscr{F}_V$ is the parametric transition probability function. ◇

Also here a valuation may induce an MDP from a PMDP, in which case we call it *admissible*. Given a PMDP $M = (S, \mathrm{Act}, V, P)$, a valuation v of the parameters induces an MDP $M_v = (S, \mathrm{Act}, P_v)$ where $P_v(s,a,s') = P(s,a,s')\langle v \rangle$, if for all s in S and a in Act we have $\sum_{s' \in S} P(s,a,s')\langle v \rangle \in \{0,1\}$. Also here, the set of admissible valuations is the *parameter space* of M.

Notice that a PMDP M and its v-induced MDP M_v have the same set of states and actions, as well as the same sets of enabled actions in each state, and therefore they have the same simple schedulers. Now, starting from a PMDP M, and given its scheduler ξ, one may: (1) first consider the ξ-induced PMC M_ξ and then the v-induced MC $(M_\xi)_v$ for a valuation v, or (2) one first takes the valuation-induced MDP M_v and then its scheduler-induced MC $(M_v)_\xi$. The result is the same and hence we write $M_{\xi,v}$ for the ξ-and-v-induced MC.

We now fix a source state s in a PMDP, and a target state t and discuss the reachability probabilities that are now dependent on both the choice of a scheduler ξ and the choice of a parameter valuation v. Given a valuation v and a scheduler ξ, the reachability probability is $\Pr^{M_{\xi,v}}(s,t)$. The (reachability probability) *wave* corresponding to ξ is a rational function f_ξ in the set of parameters, such that $f_\xi\langle v \rangle = \Pr^{M_{\xi,v}}(s,t)$. The (reachability probability) *sea* consists of all f_ξ for all schedulers ξ.

We also write (for a PMDP M):

$$\begin{array}{lcl} \Pr^{M_v}_{\max}(s,t) & = & \max_\xi \Pr^{M_{\xi,v}}(s,t), \\ \Pr^{M_\xi}_{\max}(s,t) & = & \max_v \Pr^{M_{\xi,v}}(s,t), \\ \Pr^{M}_{\max}(s,t) & = & \max_{\xi,v} \Pr^{M_{\xi,v}}(s,t), \end{array}$$

and similarly for the minimum reachability probabilities.

5 Classes of Optimal Schedulers

In this section we define and discuss a selection of types of optimal schedulers. This is meant to serve as an invitation for the reader to further develop useful notions of optimality.

Our initial idea is the following: Once we have generated all rational functions (corresponding to all schedulers), a type of optimality assigns a score to each rational function (and hence to the scheduler inducing it). The optimal schedulers of this type then maximise or minimise the assigned score.

We introduce the notion of a *dominant scheduler* and nine additional types of optimal schedulers. These types are: the optimistic, the pessimistic, the bound, the expectation, the stable, the ε-bounded, the ε-stable, and the ε-bounded- and ε-stable-robust. We next present the definition for each of them. For simplicity, we may use scheduler and function interchangeably — thus identifying a scheduler and its induced rational function when no confusion may arise.

Definition 8 (Dominant scheduler)**.** A scheduler ω is dominant if at any parameter valuation v, its function has the maximal value of all functions of all schedulers, i.e., $\forall v. \forall \xi . f_\omega\langle v \rangle \geq f_\xi\langle v \rangle$. ◇

Definition 9 (Optimistic scheduler)**.** A scheduler ω is optimistic, if its function has the maximal maximum value of all functions of all schedulers, i.e.,

$$\Pr^{M_\omega}_{\max}(s,t) = \max_\xi \Pr^{M_\xi}_{\max}(s,t) = \Pr^{M}_{\max}(s,t).$$ ◇

Definition 10 (Pessimistic scheduler)**.** A scheduler is pessimistic, if its function has the maximal minimum value of all functions of all schedulers, i.e.,

$$\Pr^{M_\omega}_{\min}(s,t) = \max_\xi \Pr^{M_\xi}_{\min}(s,t).$$ ◇

Definition 11 (Bound scheduler)**.** A scheduler is bound, if its function has the minimal range, i.e., minimal difference between its maximal and minimal value of all functions of all schedulers, i.e.,

$$\Pr_{\max}^{M_\omega}(s,t) - \Pr_{\min}^{M_\omega}(s,t) = \min_\xi \left(\Pr_{\max}^{M_\xi}(s,t) - \Pr_{\min}^{M_\xi}(s,t) \right). \quad \diamond$$

Definition 12 (ε-Bounded scheduler)**.** A scheduler ξ is ε-bounded if the length of the (closed-interval) range of its function is bounded by ε, i.e.,

$$\Pr_{\max}^{M_\xi}(s,t) - \Pr_{\min}^{M_\xi}(s,t) \leq \varepsilon$$

for a non-negative real number ε. $\diamond$

Definition 13 (ε-Bounded robust scheduler)**.** A scheduler ω is ε-bounded robust if it is the maximal among all ε-bounded schedulers, i.e., $\forall v. \forall\ \varepsilon\text{-bound } \xi . f_\omega\langle v\rangle \geq f_\xi\langle v\rangle$. $\diamond$

The intuition behind these types of optimal schedulers is the following. If a user does not know the value of the parameters, then taking the

- dominant scheduler guarantees that one can do as good as it gets independent of the parameters;
- optimistic scheduler guarantees that one can do as good as it gets in case the parameters are the best possible;
- pessimistic scheduler guarantees that no matter what the parameters are, even in the worst case we will perform better than the worst case of any other scheduler;
- bound scheduler guarantees that one will see minimal difference in reachability probability by varying the parameters;
- ε-boundness is an absolute notion guaranteeing that such a scheduler never has a larger difference in reachability probability than ε;
- finally, ε-bounded robustness gives the maximal scheduler among all ε-bounded ones.

Dominant, ε-bounded, and ε-bounded robust schedulers need not exist.

Note that computing optimistic, pessimistic, bound, ε-bounded, and ε-bounded robust schedulers requires computing the maximum and the minimum of the involved functions, which is in general hard [16], see Section 7 for more details.

The following classes do not require computing extremal values and may provide a better global picture of the reachability probabilities. Their optimality is based on maximising/minimising or bounding the probability mass over the whole parameter space, also allowing for specifying a probability distribution on the parameter space. If the distribution of parameters is unknown, we assume uniform distribution. However, it is likely that a distribution of parameters is known or can be estimated, in which case these schedulers take it into account. From now on, Let p denote a probability density function over the parameter space.

Before we proceed, let us define the expectation and variance of a scheduler. The expectation of a scheduler ξ is $E(\xi) = E(f_\xi) = \int f_\xi \, dp$ and the variance is $\mathrm{Var}(\xi) = E(\xi - E(\xi))^2$. Note that here $\xi - E(\xi)$ denotes the rational function $f_\xi - E(\xi)$.

Definition 14 (Expectation Scheduler)**.** A scheduler is an expectation scheduler, if its function has the maximal expected value of all functions of all schedulers, i.e., ω is an expectation scheduler if $E(\omega) = \max_\xi E(\xi)$. $\diamond$

Definition 15 (Stable scheduler)**.** A scheduler ω is stable, if its function has the minimal variance, i.e.,

$$\mathrm{Var}(\omega) = \min_\xi \mathrm{Var}(\xi). \quad \diamond$$

Definition 16 (ε-Stable scheduler)**.** A scheduler ξ is ε-stable if its variance is bounded by ε, i.e.,

$$\mathrm{Var}(\xi) \leq \varepsilon$$

for a non-negative real number ε. ◇

Definition 17 (ε-Stable robust scheduler)**.** A scheduler ω is ε-stable robust if it its expectation is maximal among all ε-stable schedulers, i.e., $E(\omega) = \max_{\varepsilon\text{-stable } \xi} E(\xi)$. ◇

If a dominant scheduler exists, then it is also optimistic, pessimistic, and expectation optimal.

Example 1. Consider the 2×2 labyrinth with sink at $(1,2)$ from Figure **??** in the introduction.

In the fixed failure case, Figure 3a, the red scheduler is dominant (and hence optimistic, pessimistic, and expectation optimal). All schedulers are optimistic, pessimistic, and bound. The yellow scheduler is stable, and the blue is (median variance)-stable.

In the fixed success case, Figure 3b, there is no dominant scheduler. The red and green schedulers are optimistic, all are pessimistic, the yellow and the blue are bound. The red is expectation optimal, the yellow is stable, and the blue is (median variance)-stable robust.

6 Parametric Labyrinths

The class of examples of a robot in a labyrinth provides a wide playground for studying parametric models. We consider $n \times n$ labyrinths. States are the positions in the labyrinth, and the set of actions is $\{N,S,E,W\}$.

Taking an action probabilistically determines the next state, as the robot may indeed reach the intended new position or fail to do so and end up in another unintended position. There are many ways to specify what happens if the robot fails, we chose the way as in the example in the introduction: our robot can fail to reach the intended position and instead end up left or right of its current position with a certain probability.

A most general way to turn this into a parametric model is to consider all probabilities depending on a parameter, e.g. in every state, for every action, there is a parameter that provides the probability to fail left and another that provides the probability to fail right, and the probability of success is determined by the values of these two parameters. This results in a model with $8|S| = 8n^2$ parameters.

We simplify this general scenario and limit the parameters to smaller numbers. In particular, we consider models with k parameters where

(1) $k = 8$ and we take per action two parameters (e.g. for action N, the probability to fail left with action N and the probability to fail right), which are then the same in every state whenever this action is taken.

(2) $k = 2$ and we take two parameters l and r that serve the purpose like in (1) and in the example in the introduction for every state and every action.

(3) $k = 1$ and we have a single parameter p in the model that serves the purpose like in (2) for every state and every action.

In all of these cases for states on the boundary we consider one of the two scenarios – fixed failure or fixed success – as specified for the example in the introduction.

In addition, we experiment with making some states sink states, just like we did with state $(1,2)$ in the introduction example.

7 Implementation and Experiments

We have implemented a first prototype of SEA-PARAM leveraging the open-source parametric model checking framework of the PRISM model checker [17] and Wolfram Mathematica[1].

SEA-PARAM receives as input a PMDP and a reachability property. Firstly, it explores all the possible memoryless schedulers generating for each of them a multivariate rational function that maps the parameter space into the probability to satisfy the desired property. For the generation and the manipulation of the multivariate rational functions, PRISM leverages the Java Algebra Systems (JAS)[2]. This task is embarrassingly parallel, since each memoryless scheduler can be treated independently from the others. We exploit this with a concurrent implementation, which leads to constant (given by the number of cores) speed-up. However, in the worst-case the number of schedulers (which we straightforwardly enumerate in this first attempt) can be exponential in the number of states, resulting in exponential running time.

After the memoryless schedulers enumeration and function computation, the corresponding multivariate rational functions are evaluated according to a chosen optimality criterion using a script developed within the Wolfram Mathematica framework. We chose Mathematica for the ability to quickly implement our different formal notions of optimality criteria for the schedulers provided in the paper. The Mathematica program takes as input the list of schedulers with their corresponding functions generated in the previous step and computes a score for each multivariate rational function. This task can again be computed in parallel for each multivariate rational function. Nevertheless, again, in general the computation of the score of a multivariate rational function is NP-hard [15, 26, 22]. For example, already the minimisation of a multi-variate quadratic function over the unit cube is NP-hard, see e.g. [22] for a reduction from SUBSET-SUM. For several classes of well-behaved functions (e.g. convex functions or unimodal ones) our scores can be efficiently computed. We know for sure that not all our functions are convex or unimodal, but there is still a chance that the functions form another well-behaved class. We intend to explore this possibility in future work.

Note that, since we generate a list of schedulers together with their rational functions, it is straightforward to find the scheduler corresponding to a rational function.

7.1 Experiments

The experiments reported here ran on a unified memory architecture (UMA) machine with four 10-core 2GHz Intel Xeon E7-4850 processors supporting two hardware threads (hyper-threads) per core, 128GB of main memory, and Linux kernel version 4.4.0. The first part (based on PRISM SVN revision 11807) was compiled and run with OpenJDK 1.8.0. The second part was executed in Mathematica 11.0 using 16 parallel kernels. During our experiments we identified a bug in a greatest-common-divisor (gcd) procedure of the JAS library, that resulted in computing wrong functions. We work around this bug by substituting a simpler gcd procedure. All of our code and detailed results of the experiments can be found at [1].

Table 1 gives an overview of our experimental results. We present 23 experiments in total, for the various scenarios described in Section 6. For each scenario the table shows the number of schedulers, the number of unique functions (as two schedulers might have the same rational function), as well as the running times of key parts of our system. In particular, we show the running time for the computation of the rational functions (column PRISM) and the computation of the expectation and optimistic optimal

[1] https://www.wolfram.com/mathematica/

[2] http://krum.rz.uni-mannheim.de/jas/

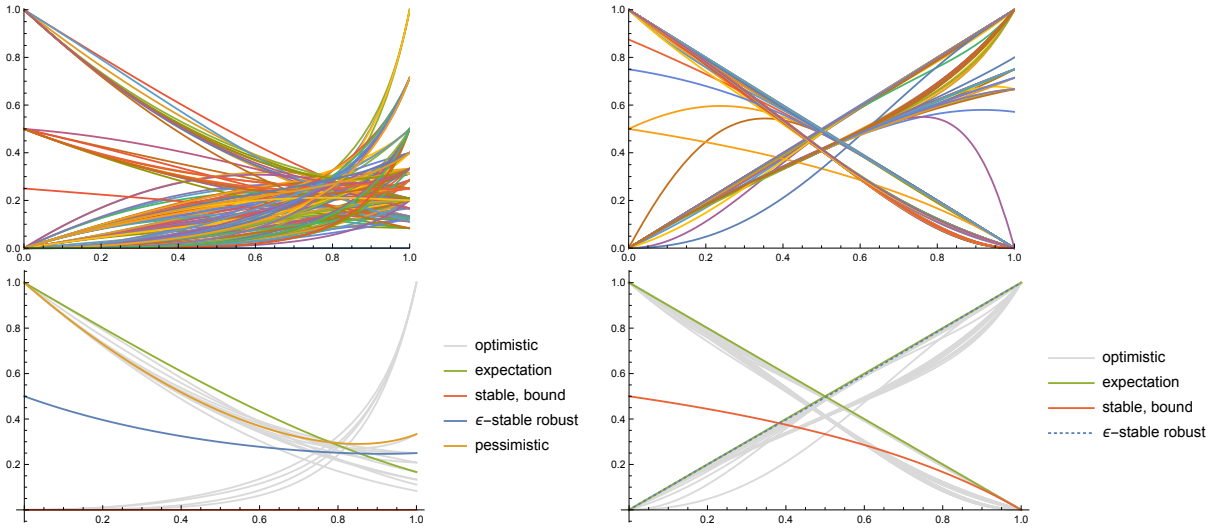
Grid scenario					Number of		Execution time in seconds		
k	size	type	target	sinks	schedulers	functions	PRISM	optimistic	expectation
8	2x2	ff	(2,2)	(1,2)	4	4	0.11	1.39	1.40
8	2x2	fs	(2,2)	(1,2)	4	4	0.10	2.19	19.09
2	2x2	ff	(2,2)	(1,2)	4	4	0.07	0.58	0.71
2	2x2	fs	(2,2)	(1,2)	216	63	1.76	1.90	0.26
2	3x3	ff	(1,3)	(1,2),(2,2)	432	120	5.68	3.94	0.69
2	3x3	ff	(2,2)	(1,2)	864	398	13.53	15.36	2.11
2	3x3	ff	(3,3)	(1,2)	648	246	6.95	7.83	0.98
2	3x3	ff	(3,3)	(2,2)	4	4	0.06	0.31	0.05
2	3x3	fs	(1,3)	(1,2), (2,2)	216	63	1.85	1.85	0.42
2	3x3	fs	(2,2)	(1,2)	432	120	6.53	4.29	0.78
2	3x3	fs	(3,3)	(1,2)	864	399	12.14	18.64	2.63
2	3x3	fs	(3,3)	(2,2)	648	234	9.18	8.13	1.32
1	2x2	ff	(2,2)	(1,2)	4	4	0.06	1.40	0.04
1	2x2	fs	(2,2)	(1,2)	216	60	0.78	5.67	0.16
1	3x3	ff	(1,3)	(1,2), (2,2)	432	114	2.29	9.72	0.18
1	3x3	ff	(2,2)	(1,2)	864	390	6.46	34.48	0.52
1	3x3	ff	(3,3)	(1,2)	648	122	3.48	12.57	0.19
1	3x3	ff	(3,3)	(2,2)	4	4	0.05	1.41	0.04
1	3x3	fs	(1,3)	(1,2), (2,2)	216	63	1.09	4.87	0.12
1	3x3	fs	(2,2)	(1,2)	432	114	1.72	11.21	0.19
1	3x3	fs	(3,3)	(1,2)	864	391	5.63	34.01	0.54
1	3x3	fs	(3,3)	(2,2)	648	124	2.42	10.21	0.17
1	4x4	fs	(4,4)	(1,2)	4478976	2010270	89677.00	42824.00	32986.00

Table 1: Overview of the experimental results

schedulers. We selected these two classes of optimal schedulers as they illustrate the characteristics of our two score classes (integral/mass vs extremal values) the best. Our largest experiments involves a 4x4 labyrinth with a single parameter; it results in over 2 million distinct rational functions (and takes significant amount of time to compute).

In Figure 4 we plot the rational functions of two 3x3 labyrinths with one parameter, to give a flavour of the different schedulers encountered. In both cases no scheduler is dominant and several of them are optimistic. In Figure 4a we see a single expectation scheduler (actually two schedulers with a single rational function) and a single pessimistic scheduler (again actually two schedulers), while in Figure 4b there are two symmetric expectation schedulers, and *all* schedulers are pessimistic (as they all have minimal value 0). In both scenarios the stable and the bound schedulers coincide - in Figure 4a the corresponding function is constant 0. We also show an ε-stable robust scheduler with ε chosen to be the median variance of the rational functions. In Figure 4b this yields a function very close to the expectation scheduler function with slightly lower variance.

Note that we plot the rational functions for the optimal schedulers. From these functions we can look up the corresponding schedulers. For instance, the function labeled expectation in Figure 4a corresponds to the two expectation optimal schedulers that in $(1,1)$ take E or N respectively; take N in $(1,2)$, $(3,1)$, and $(3,2)$; and take E in $(1,3)$, $(2,3)$ (and of course in $(2,1)$ where there is no other choice).

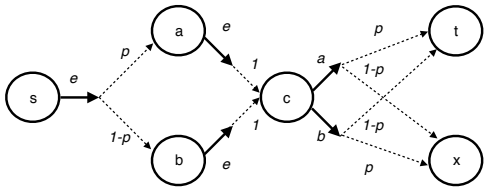


(a) fixed failure from (1,1) to (3,3) with a sink at (2,2) (b) fixed success from (1,1) to (2,2) with a sink at (1,2)

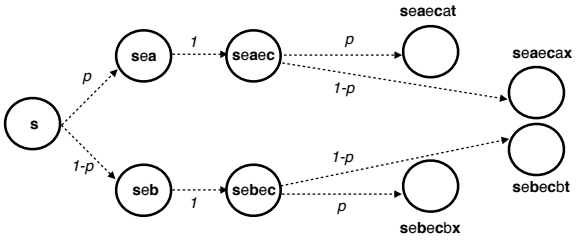
Figure 4: The rational functions of schedulers for two 3x3 labyrinths with $k = 1$

8 Discussion

In our first-version prototype implementation of SEA-PARAM we focus on simple schedulers, which are already exponentially many. However, not always simple schedulers are optimal according to our optimality definitions. A history dependent (hence not simple) scheduler may estimate the parameters and thus provide better behaviour than any simple scheduler, as we show with the following example.



(a) An example PMDP M



(b) Induced MC by a history-dependent scheduler

Figure 5: History dependency

Consider the PMDP M in Figure 5a where p is a parameter. There are two simple schedulers for M: α with $\alpha(\mathbf{c}) = a$ and β with $\beta(\mathbf{c}) = b$ (all other states are mapped to the single available action and **t** and **x** are sink states). Their corresponding rational functions are $f_\alpha(p) = p$ and $f_\beta(p) = 1 - p$.

Consider now the history-dependent scheduler χ of M that schedules a in state **c** if and only if the state **a** has been visited before. The χ-induced MC is shown in Figure 5b. The rational function corresponding to χ is $f_\chi(p) = p^2 + (1-p)^2$. All three rational functions are depicted in Figure 6, and χ wins in all optimality classes against α and β.

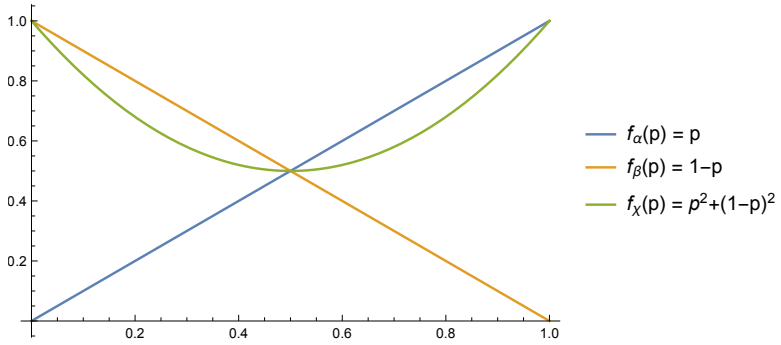


Figure 6: The rational functions of α, β, and χ; χ wins in optimality

We aim at broadening our scheduler exploration to history-dependent schedulers in the near future.

Acknowledgments. This work was supported by the Austrian National Research Network RiSE/SHiNE (S11405-N23 and S11411-N23) project funded by the Austrian Science Fund (FWF) and partially by the Fclose (Federated Cloud Security) project funded by UnivPM.

References

[1] Sebastian Arming, Ezio Bartocci & Ana Sokolova (2017): *SEA-PARAM*. `https://github.com/sarming/sea-param`.

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

[3] Ezio Bartocci, Luca Bortolussi, Laura Nenzi & Guido Sanguinetti (2015): *System design of stochastic models using robustness of temporal properties*. *Theor. Comput. Sci.* 587, pp. 3–25, doi:10.1016/j.tcs.2015.02.046.

[4] Ezio Bartocci, Radu Grosu, Panagiotis Katsaros, C. R. Ramakrishnan & Scott A. Smolka (2011): *Model Repair for Probabilistic Systems*. In: *Proc. of TACAS 2011: the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, *LNCS* 6605, Springer, pp. 326–340.

[5] Luca Bortolussi, Dimitrios Milios & Guido Sanguinetti (2016): *Smoothed model checking for uncertain Continuous-Time Markov Chains*. *Inf. Comput.* 247, pp. 235–253, doi:10.1016/j.ic.2016.01.004.

[6] Taolue Chen, Ernst Moritz Hahn, Tingting Han, Marta Z. Kwiatkowska, Hongyang Qu & Lijun Zhang (2013): *Model Repair for Markov Decision Processes*. In: *Proc. of TASE 2013: the seventh International Symposium on Theoretical Aspects of Software Engineering*, IEEE Computer Society, pp. 85–92.

[7] Murat Cubuktepe, Nils Jansen, Sebastian Junges, Joost-Pieter Katoen, Ivan Papusha, Hasan A. Poonawala & Ufuk Topcu (2017): *Sequential Convex Programming for the Efficient Verification of Parametric MDPs*. In: *Proc. of TACAS 2017*, Springer, p. to appear.

[8] Conrado Daws (2005): *Symbolic and Parametric Model Checking of Discrete-Time Markov Chains*. In: *Proc. of ICTAC 2004: the International Colloquium of Theoretical Aspects of Computing*, *LNCS* 3407, Springer, pp. 280–294.

[9] Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius, Matthias Volk, Harold Bruintjes, Joost-Pieter Katoen & Erika Ábrahám (2015): *PROPhESY: A PRObabilistic ParamEter SYnthesis Tool*. In: *Proc.*

of CAV 2015: the 27th International Conference Computer Aided Verification, *LNCS* 9206, Springer, pp. 214–231.

[10] Rob J. van Glabbeek, Scott A. Smolka & Bernhard Steffen (1995): *Reactive, Generative and Stratified Models of Probabilistic Processes*. *Inf. Comput.* 121(1), pp. 59–80, doi:10.1006/inco.1995.1123.

[11] Ernst Moritz Hahn, Tingting Han & Lijun Zhang (2011): *Probabilistic reachability for parametric Markov models*. *STTT* 13(1), pp. 3–19.

[12] Ernst Moritz Hahn, Tingting Han & Lijun Zhang (2011): *Synthesis for PCTL in Parametric Markov Decision Processes*. In: *Proc. of NFM 2011: the third Third International Symposium on NASA Formal Methods*, *LNCS* 6617, Springer, pp. 146–161.

[13] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter & Lijun Zhang (2010): *PARAM: A Model Checker for Parametric Markov Models*. In: *Proceedings of the 22nd International Conference on Computer Aided Verification(CAV 2010)*, pp. 660–664.

[14] Nils Jansen, Florian Corzilius, Matthias Volk, Ralf Wimmer, Erika Ábrahám, Joost-Pieter JKatoen & Bernd Becker (2014): *Accelerating Parametric Probabilistic Verification*. In: *Quantitative Evaluation of Systems - 11th International Conference, QEST 2014, Florence, Italy, September 8-10, 2014. Proceedings*, *LNCS* 8657, Springer, pp. 404–420.

[15] Akitoshi Kawamura (2011): *Computational Complexity in Analysis and Geometry*. Ph.D. thesis, University of Toronto.

[16] Vladik Kreinovich, Anatoly Lakeyev, Jiří Rohn & Patrick Kahl (1998): *Computational Complexity and Feasibility of Data Processing and Interval Computations*. *Applied Optimization* 10, Springer US, Boston, MA.

[17] Marta Z. Kwiatkowska, Gethin Norman & David Parker (2011): *PRISM 4.0: Verification of Probabilistic Real-Time Systems*. In: *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, *Lecture Notes in Computer Science* 6806, Springer, pp. 585–591.

[18] Ruggero Lanotte, Andrea Maggiolo-Schettini & Angelo Troina (2007): *Parametric probabilistic transition systems for system design and analysis*. *Form. Asp. Comput.* 19(1), pp. 93–109.

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

[20] C. Lefevre (1981): *Optimal control of a birth and death epidemic process*. *Oper. Res.* 29(5), pp. 971–982.

[21] A. I. Medina Ayala, S. B. Andersson & C. Belta (2012): *Probabilistic control from time-bounded temporal logic specifications in dynamic environments*. In: *Proc. of ICRA 2012*, IEEE, pp. 4705–4710.

[22] Katta G Murty & Santosh N Kabadi (1987): *Some NP-complete problems in quadratic and nonlinear programming*. *Mathematical Programming* 39(2), pp. 117–129.

[23] Shashank Pathak, Erika Ábrahám, Nils Jansen, Armando Tacchella & Joost-Pieter Katoen (2015): *A Greedy Approach for the Efficient Repair of Stochastic Models*. In: *Proc. of NFM 2015: the 7th International Symposium on NASA Formal Methods*, *LNCS* 9058, Springer, pp. 295–309.

[24] Q. Qiu, Q. Wu & M. Pedram (2001): *Stochastic modeling of a power-managed system-construction and optimization*. *IEEE T. Comput. Aid. D.* 20(10), pp. 1200–1217.

[25] Tim Quatmann, Christian Dehnert, Nils Jansen, Sebastian Junges & Joost-Pieter Katoen (2016): *Parameter Synthesis for Markov Models: Faster Than Ever*. In: *Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings*, *Lecture Notes in Computer Science* 9938, pp. 50–67, doi:10.1007/978-3-319-46520-3.

[26] Sartaj Sahni (1974): *Computationally Related Problems*. *SIAM J. Comput. ()* 3(4), pp. 262–279.

[27] R. Segala & N.A. Lynch (1994): *Probabilistic Simulations for Probabilistic Processes*. In: *Proc. Concur'94*, LNCS 836, pp. 481–496.

[28] Linn I. Sennott (1998): *Stochastic Dynamic Programming and the Control of Queueing Systems*. John Wiley & Sons, Inc.

PAWS: A Tool for the Analysis of Weighted Systems

Barbara König
Universität Duisburg-Essen
barbara_koenig@uni-due.de

Sebastian Küpper
Universität Duisburg-Essen
sebastian.kuepper@uni-due.de

Christina Mika
Universität Duisburg-Essen
christine.mika@uni-due.de

PAWS is a tool to analyse the behaviour of weighted automata and conditional transition systems. At its core PAWS is based on a generic implementation of algorithms for checking language equivalence in weighted automata and bisimulation in conditional transition systems. This architecture allows for the use of arbitrary user-defined semirings. New semirings can be generated during run-time and the user can rely on numerous automatisation techniques to create new semiring structures for PAWS' algorithms. Basic semirings such as distributive complete lattices and fields of fractions can be defined by specifying few parameters, more exotic semirings can be generated from other semirings or defined from scratch using a built-in semiring generator. In the most general case, users can define new semirings by programming (in C#) the base operations of the semiring and a procedure to solve linear equations and use their newly generated semiring in the analysis tools that PAWS offers.

1 Introduction

In recent times, modelling techniques have shifted from a purely acceptance-based reasoning to one that takes various notions of weight and quantities into consideration. The theory of weighted automata gives rise to a flexible framework, where acceptance behaviour can be quantified in numerous ways. Probabilities are commonly used to express the likelihood of making a given transition, whereas the tropical semiring is often used to denote the cost of making a transition. Due to the great variety of semirings the theory of weighted automata can be applied to a multitude of different fields of interest, such as natural language processing, biology or performance modelling (see e.g. [12]). Typical questions regarding weighted automata concern their language (or traces), for instance language equivalence. Due to the generality of the notion of weighted automata, they are a versatile means for modelling purposes, though decidability results are not as strong as for non-deterministic automata. In particular, it is well-known that language equivalence is an undecidable problem for weighted automata over the tropical semiring [16]. This, however, is not a damning result for language equivalence, since it is also well-known that language equivalence is decidable for many other semirings such as the two-valued boolean algebra (for which weighted automata are just nondeterministic finite automata) or fields (where decision procedures run in polynomial time [14, 7]). PAWS[1] is a tool that offers algorithms to decide language equivalence and the threshold or universality problem for weighted automata.

Based on our previous work in [15] and [5], the tool PAWS implements two different approaches to language equivalence checks for weighted automata. One approach employs a backwards search assuming at first that all states are language equivalent and exploring words from the end to the beginning to determine non-equivalent pairs of states, similar to partition refinement. Different from usual partition refinement algorithms, this algorithm does not necessarily terminate at the moment when the partitions cannot be further refined in one step, because refinements can happen at a later point of time. The termination condition has to be chosen differently here, and as expected, the algorithm does not necessarily

[1] The tool can be downloaded from www.ti.inf.uni-due.de/research/tools/paws/

Submitted to:
QAPL 2017

terminate for all semirings (see the undecidability result by Krob [16]). However, for many semirings it does and in either case it is always a suitable semi-decision procedure for the absence of language equivalence. This approach is closely related to the line of work started by Schützenberger, [18] and later generalised in [3], using the notion of conjugacy.

Additionally, PAWS offers a second approach to decide language equivalence [5], which stands in the tradition of Bonchi's and Pous' seminal work on equivalence checks for NFAs using up-to techniques [6]. Here, a language equivalence relation on vectors is built, starting from the initial pair of vectors suspected to be language equivalent. As the algorithm progresses, it builds a language equivalence relation similar to a bisimulation relation and stops at the moment when a suitable relation proving language equivalence is found, or a witness to the contrary appears. The algorithm works up-to congruence and therefore prunes the relation on-the-fly, dropping redundant vector pairs. The flexibility of this optimised variant of the algorithm is reduced when compared to the partition refinement algorithm, but it can still be used for a variety of user-generated semirings, such as rings and l-monoids, and can lead to an exponential speed up in some cases. Based on a similar approach, PAWS also offers a decision procedure for the universality problem for weighted automata over the tropical semiring of the natural numbers, which is potentially exponentially faster than a naive approach due to Kupferman et al. [2]. The universality problem checks whether from a given starting vector, all words have a weight smaller than or equal to a given threshold.

Finally, PAWS also considers conditional transition systems, which, rather than adding weights to transitions, extend traditional transition systems by means of conditions, or product versions to enable flexible modelling of software product lines, while taking possible upgrades between different products of a software product line into account [8]. For these kinds of systems, bisimulation rather than language equivalence is considered, because the user experience for different products is in the focus. The bisimulation check can be performed on any finite distributive lattice defined via its set of join irreducible elements. Alternatively, a BDD-based implementation of (certain) lattices is offered and allows for a significantly faster bisimulation check as presented in [4]. Again, the bisimulation check is parametrised over the lattice used and can accept any lattice, either defined directly via its irreducible elements, or using the BDD-based approach, as input.

A key feature of PAWS is its extensibility. The algorithms are parametrised over the semiring and it is therefore possible to use the algorithms PAWS offers, not only for the semirings that come pre-implemented, but also for newly generated semirings. For that purpose, PAWS offers ways of adding new semirings and executing algorithms for these semirings. All algorithms are implemented generically and can be used for various semirings or l-monoids, provided all necessary operations such as addition, multiplication, and solving linear equations are specified. Therefore, PAWS is equipped with a semiring generator that allows to generate new semirings that are not pre-implemented and to define weighted automata or conditional transition systems over these. Specifically, we have built several layers of automatisation, so that whenever one, e.g., defines a complete distributive lattice, it suffices to give a partial order, from which the lattice of downward-closed sets is generated [11] and all operations are provided automatically. Building semirings from other semirings using crossproducts is almost completely automatised and modulo rings are automatised using Hensel liftings [10]. In addition, it is possible to add arbitrary semirings by providing code for the operations mentioned before.

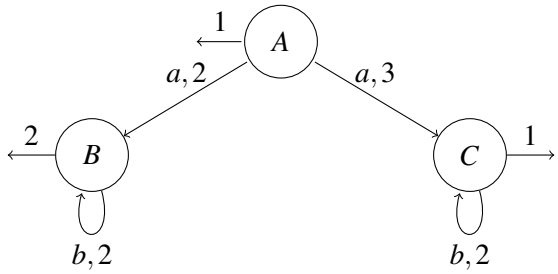
2 Preliminaries: System Types and Decision Problems

Here we give a short overview over the systems that PAWS can analyse, i.e. weighted automata and conditional transition systems. For that purpose we require the notions of semirings and distributive

lattices.

- A *semiring* is a tuple $\mathbb{S} = (S, +, \cdot, 0, 1)$ where $(S, +, 0)$ is a commutative monoid, $(S, \cdot, 1)$ is a monoid, 0 *annihilates* $\cdot$ (i.e., $0 \cdot s_1 = 0 = s_1 \cdot 0$) and $\cdot$ *distributes over* $+$ (i.e., $(s_1 + s_2) \cdot s_3 = s_1 \cdot s_3 + s_2 \cdot s_3$ and $s_3 \cdot (s_1 + s_2) = s_3 \cdot s_1 + s_3 \cdot s_2$, for all $s_1, s_2, s_3 \in S$).
- A *complete distributive lattice* is a partially ordered set $(L, \sqsubseteq)$ where for all subsets $L' \subseteq L$ of L the infimum $\bigsqcap L'$ and the supremum $\bigsqcup L'$ w.r.t. the order $\sqsubseteq$ exists and infimum distributes over finite suprema: $(\ell_1 \sqcup \ell_2) \sqcap \ell_3 = (\ell_1 \sqcap \ell_3) \sqcup (\ell_2 \sqcap \ell_3)$ for all $\ell_1, \ell_2, \ell_3 \in L$. Together with $\top = \bigsqcup L$ and $\bot = \bigsqcap L$, a complete distributive lattice forms a semiring $\mathbb{L} = (L, \sqcup, \sqcap, \bot, \top)$.
- An l-monoid is a lattice $(L, \sqcup, \sqcap, \bot, \top)$ together with a monoid $(L, \cdot, 1)$ such that $\cdot$ distributes over $\sqcup$. If $\bot$ annihilates $\cdot$, i.e. $\bot \cdot \ell = \bot$ for all $\ell \in L$, then the l-monoid can be regarded as a semiring $(L, \sqcup, \cdot, \bot, 1)$.

A weighted automaton (WA) can be understood as a non-deterministic automaton that additionally carries weights from a given semiring $\mathbb{S}$ on each transition, as well as a termination weight for each state. Rather than accepting or rejecting a word w, a state x in a weighted automaton associates it with a value from $\mathbb{S}$. This value can be obtained as follows: Take the sum of the weight of all w-labelled paths p starting in x. The weight of a path p is the product of all transition weights along p, including the termination weight. Consider for instance the following simple weighted automaton over the field $\mathbb{R}$:



The weight of the word *ab* in state *A* can be computed as follows: there exist two paths to consider, (A, a, B, b, B) and (A, a, C, b, C). The first path has the weight $2 \cdot 2 \cdot 2 = 8$ and the second path has the weight $3 \cdot 2 \cdot 1 = 6$, so overall, the weight of the word *ab* in *A* is $8 + 6 = 14$. For weighted automata, PAWS offers an algorithm to decide which pairs of states are language equivalent, i.e. assign the same weight to all words. In general, this problem is undecidable, but for many specific semirings (e.g. the reals) it is decidable.

PAWS offers two algorithms to decide language equivalence. The more generally applicable algorithm called *Language Equ.(Complete)* in the tool, can be applied to any semiring and yields, if it terminates, a complete characterisation of language equivalence in the automaton.

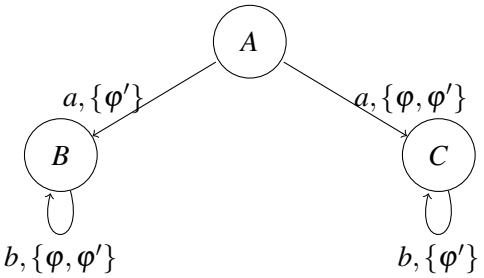
For complete and completely distributive lattices, and more generally, completely distributive l-monoids, an optimisation up to congruence is available (*Language Equ.(Up-To)*). This algorithm checks whether two initial vectors are language equivalent by building a language equivalence relation on vectors over the semiring and pruning the relation by congruence closure, i.e. pairs of vectors that are already in the congruence closure of previously found pairs of vectors, are discarded.

One semiring, where language equivalence is undecidable is the tropical semiring over the natural numbers $(\mathbb{N}_0 \cup \{\infty\}, \mathsf{min}, +, \infty, 0)$, as shown by Krob [16]. However, the universality problem is decidable in this case. The universality problem asks, for any given number $n \in \mathbb{N}_0$, whether the weight of all words from a given starting vector is bounded by n.

All algorithms for these problems require a method for solving linear equations over the semiring.

PAWS also analyses a second type of systems: conditional transition systems (CTS) [4, 1]. CTS are defined over a finite partial order of conditions. Each transition is assigned to a downwards closed set of conditions under which the transition may be taken. Before execution, one condition is fixed and all transitions that carry the respective condition remain active and all other transitions remain inactive. Afterwards, the CTS evolves just like a traditional labelled transition system. At any point though, a change of conditions can occur by going down in the order. If the condition is changed, so do the active transitions and additional transitions may become available. Note that due to the requirement that all transitions carry a downwards closed set of conditions, only additional transitions can appear, no transition can be deactivated by performing a change in conditions. Therefore this change of conditions can be considered as an upgrade of the system. Also note the strong relation to complete distributive lattices: Every finite partial order gives rise to a lattice by considering all downwards closed sets of elements ordered by inclusion. And vice versa, every finite distributive lattice can be represented in this way [11].

Now consider the following CTS defined over the set of conditions $\{\varphi, \varphi'\}$, where $\varphi' \leq \varphi$.



Furthermore assume we are starting in state A and under the initial condition φ. Then only the transitions from B to B and from A to C are available. If we choose to make a step from A to C via the a-labelled edge, we cannot do any further steps, unless we first perform an upgrade to φ', which allows us to use the b-loop in state C.

For CTS we are interested in conditional bisimulation. Two states are conditionally bisimilar, if there exists a conditional bisimulation relating the states. A conditional bisimulation is family of traditional bisimulations R_φ, one for each condition φ, on the respective underlying transition systems. For two conditions $\varphi' \leq \varphi$ it must hold that $R_{\varphi'} \supseteq R_\varphi$, which intuitively means that if two states are bisimilar under φ, they must also be bisimilar under every smaller condition φ'. Furthermore, the standard transfer property for bisimulations must be satisfied.

In [4] we have shown how to model a small adaptive routing protocol as CTS.

Summarizing, the problems PAWS solves and the corresponding algorithms and semirings are displayed in the following table:

Problem	Algorithm	Semiring	Model
Language equivalence (all pairs)	*Language Equ.(Complete)*	any semiring	WA
Language equivalence (initial vectors)	*Language Equ.(Up-To)*	l-monoids, lattices	WA
Universality Problem	*Universality*	Tropical Semiring	WA
Conditional bisimilarity	*CTS Bisimilarity*	finite lattice	CTS

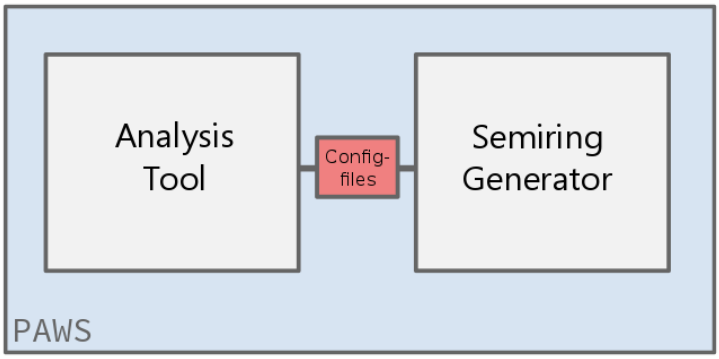
3 Design and Usage

In this section, we give an overview of some design decisions and the usage of the tool. First, we mention the basic structure of the tool and then discuss some of the problems and math-related features of the tool.

Furthermore we explain how to work with PAWS.

3.1 Design

PAWS is a Windows tool offering a complete graphical interface, developed in Microsoft's Visual Studio using C#. The program is divided into two autonomous components:



Both components are designed according to the MVC (Model View Controller) pattern. In the sequel, we will discern these two program parts, as their interaction is rather limited, allowing them to be considered as separate concerns.

3.1.1 The Semiring Generator

The development of the semiring generator started in a master's thesis [17]. It supports five different generation processes, which, for clarity, are equipped with five separate input forms. The semiring generator supports three fully automated cases:

- Direct products
- Fields of fractions
- Field extensions for $\mathbb{Q}$

Furthermore, there are two options to generate code based on user implementations:

- l-monoids
- Arbitrary semirings

Note that generation of finite lattices and modulo semirings is less involved and is done directly within the analysis component.

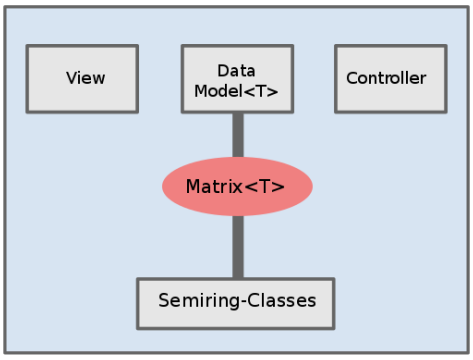
For the generation of source code we use *CodeDOM* of the *.NET* Framework, which enables code generation based on object graphs. The five processes are implemented via one superclass and three derived classes. The superclass contains all methods for creating if-statements, for-loops or useful combinations of these, based on predefined patterns. Except for the direct product generator, every class uses the constructor generating methods of the superclass. Most of the differences between the classes are reflected in the methods for generating the binary operators. Concerning methods for solving linear equations we are offering two templates: a naive implementation of the Gaussian algorithm and one method for l-monoids, based on the residuum operator [9].

With code generation, one always has to face the question of how to give the program, specifically the analysis component of PAWS, access to the newly generated classes. We decided to use the *Microsoft.Build.Execution* namespace for updating the analysis tool. This decision avoids creating multiple DLL files, which would be the case with a pure reflection-based solution to the problem. However, reflection is used to solve another difficulty. Due to the combination of different semirings or data types as elements of a new semiring, a dynamic approach is required to enable the automated generation of constructors with a string parameter, specifying the semiring value.

PAWS also manages the names of the semiring classes in individual text files. First, this prevents a user from overwriting an already fixed class name. Another advantage is that by using *System.Activator*, an instance can be created dynamically during runtime without knowing the class name at the level of program design. Hence, both components use consistent config files, which is ensured by updating the corresponding config file if one of the program parts creates or deletes a semiring. But when deleting a semiring that has already been used to create and store an automaton or transition system, conflicts may occur with the serialised objects and thus with the user's storage files. Therefore, deletion of semirings must be dealt with separately. A semiring can only be deleted or modified if it has not been used in the previous session within an automaton.

3.1.2 The Analysis Tool

As already mentioned, for both program parts MVC is used to implement the user interface. The main focus of PAWS is to give the user the tools to define an automaton in order to be able to subsequently analyse it with the supported algorithms. In order to implement this as dynamically as possible, we have opted for a generic implementation. Therefore, the class $Matrix\langle T\rangle$ (where T is the generic type of the semiring), which implements automata in a matrix representation, is the core of the tool's architecture.



The main argument for the generic approach is that the algorithms for analysis of an automaton are based on the basic operators of a semiring. A further method is needed to solve systems of linear equations, which is also defined for each semiring within PAWS. It is therefore recommended to use a generic class that includes all the methods of analysis, which in turn dynamically call the corresponding operators or methods of the currently used semiring. This dynamic approach is thus combined with reflection.

Therefore, the management of automata created by the user requires a generic implementation as well. The model for the matrices is therefore also generic and thus it makes sense to manage several semiring models by the controller.

Because of the generic approach, automatically checking the correctness of the user input proves to be problematic when the user has generated his or her own semiring and has implemented a string

constructor to read semiring elements from input strings without input verification. In this case PAWS can not check whether the input is well-formed, this has to be taken care of in the user-implemented method. Such semirings can not be further used to generate other semirings.

A further design decision is that one can create finite semirings directly within the analysis component of PAWS. The reason why we chose to not move this option to the generator is that for finite lattices and modulo semirings over the integers no new classes have to be generated nor is there any need for new source code at all. In this case, it is sufficient to configure template classes for the corresponding semirings via static variables that contain the required information about the semiring. The operators are designed to behave according to the configuration of the class. In such cases, the analysis program must also access the configuration files in order to make the extensions known to the semiring generator.

As an additional feature, we have also integrated GraphViz[2] into the tool, as it allows visualization of weighted automata, if desired by the user.

3.2 Usage

We will discuss the usage of the tool separately for the two individual components of PAWS, hence this section contains subsections giving details about the following two components.

- The *semiring generator* to build and provide the required semirings over which automata can be defined. This generator is used to generate semirings that cannot be obtained in a fully automated way and supports some automatic generations.
- The *analysis tool* that allows the user to choose a previously generated semiring, one of the semirings that come built-in with the PAWS or to build a lattice and then to define automata in a matrix representation over those lattices. Matrices are then interpreted as weighted automata or conditional transition systems (CTS) and can be used to compute language equivalence for weighted automata with two different approaches, decide the threshold problem for weighted automata over the tropical semiring of natural numbers or to compute the greatest bisimilarity of a CTS.

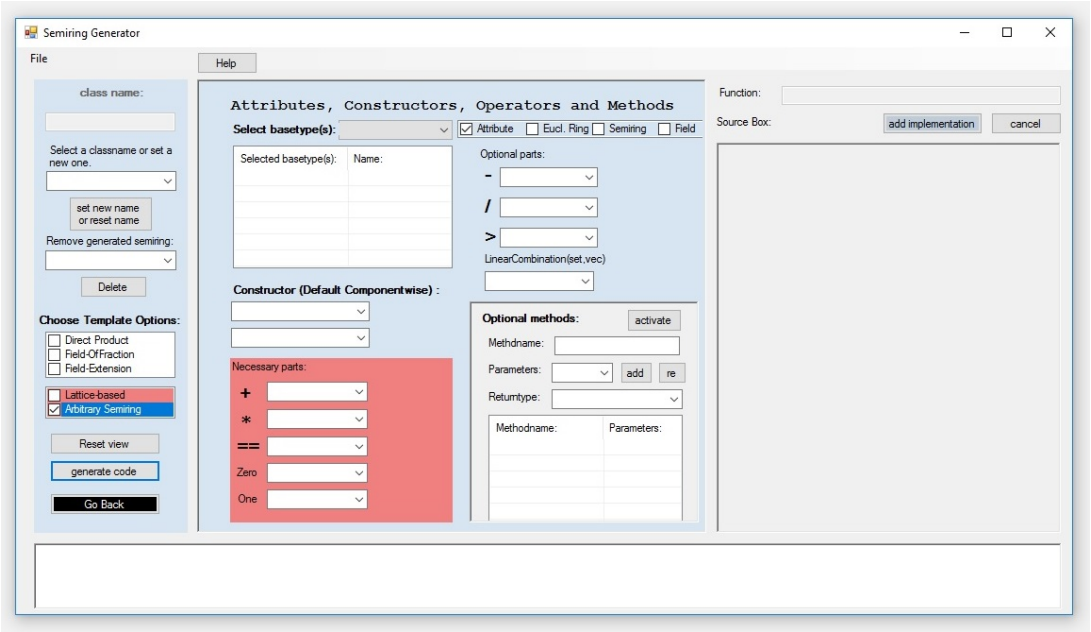
3.2.1 The Semiring Generator

The semiring generator is used to generate the semirings under consideration. In order for a semiring to be usable within the context of PAWS, the structure needs to define the following components:

- A universe which contains all elements of the semiring. All predefined datatypes from C#, as well as combinations of them can serve as universes.
- An addition operator $+$ of the semiring.
- A multiplication operator $*$ of the semiring.
- One() and Zero() methods, which return the units of addition and multiplication.
- A method for solving linear equations over the semiring.

While the first four components are necessary to define a semiring in a mathematical context either way, the procedure to solve linear equations is an additional requirement – one that PAWS aims at reducing as much as possible – but in order to provide the greatest amount of flexibility possible, the tool offers the option to define a procedure to solve linear equations from scratch.

[2] `www.graphviz.org`



3.2.2 The Analysis Tool

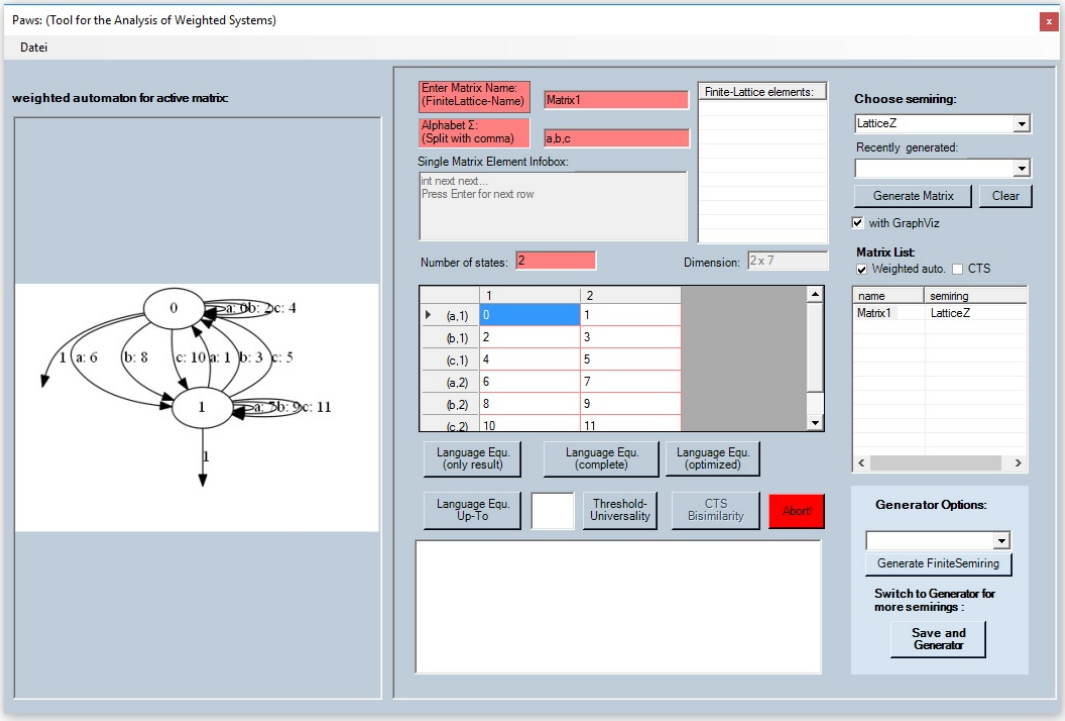
The analysis component is designed to offer generic algorithms applicable to numerous predefined or user-defined semirings. Some of the algorithms can however only be used with specific (types) of semirings. The most general algorithm is the language equivalence check, for which all semirings are eligible. Conditional Transition Systems are only defined over lattices, therefore, the bisimulation check is limited to lattice structures. However, the user still has the choice between two different ways of dealing with lattices: representing elements of the lattice as downwards closed sets of irreducibles via the Birkhoff duality [11], applicable to all finite distributive lattices, or representing them using binary decision diagrams (BDDs). The BDD variant is more restrictive and mainly designed for the application to CTS. Here, the irreducibles are required to be full conjunctions of features from a base set of features, ordered by the presence of distinct upgrade features. Lastly, the threshold check can only be performed over a single semiring, the tropical semiring over natural numbers.

The general workflow of the analysis tool is as follows:

- ▷ Choose a semiring
- ▷ Generate a matrix over this semiring, representing a weighted automaton or a conditional transition system

  Alternatively: Choose the matrix from a list of matrices that have been generated previously
- ▷ Start the algorithm and provide – if necessary – additional input

Additional input comes in two forms: starting vectors and the threshold to be checked against in case of the threshold algorithm. Depending on the semiring of choice, questions regarding language equivalence might not be decidable, leading to non-termination of the corresponding procedure in PAWS. In order to deal with this problem and to allow abortion of an overlong computation, the actual computation is delegated to a separate thread that can at any time be aborted by clicking a red button labelled “Abort”. In that case all intermediate results are discarded.



Note that only the two language equivalence-based algorithms can run into non-termination issues. For the CTS bisimulation check, as well as the threshold problem on the tropical semiring of natural numbers, termination is always guaranteed. However, the runtime of CTS bisimulation check can be doubly exponential in the number of features under consideration – because the lattice is the set of all possible configurations, which in turn are all possible conjunctions over the features. Using the BDD-based implementation of lattices – which is particularly suited to the needs of CTS modelling – this explosion is mitigated in many cases, but it can not be ruled out completely. On the other hand, the BDD-based modelling only allows for special lattices to be modelled, i.e. those that arise as the lattices constructed from a set of features and upgrade features, whereas the variant called *FiniteLattice* allows for arbitrary (finite, distributive) lattices to be represented. In this case, lattices are represented via the partial order of irreducible elements, using Birkhoff's representation theorem [11].

Here we do not give any runtime results, but refer the reader to [4, 5] where we describe several case studies and list runtime results.

4 GUI Overview of PAWS

In this section, we demonstrate the handling of the various features and options that PAWS offers. First, we will present the semiring generator. In this case, we briefly explain a fully automated generation and the even more complex case, in which, apart from the constructor, the user has to specify the implementations on his or her own. We then illustrate the various possibilities to use the analysis component of PAWS in a short overview.

4.1 The Semiring Generator

First, we consider a fully automated generation demonstrated by the direct product input-mask (Figure 1 and 2). Then, in Figure 3 the console informs the user that the generated source code is compilable. In case the user specifies some code on his or her own, the console will display suitable compilation error messages. Figures 4 and 5 depict the use of the input mask for user-dependent implementations.

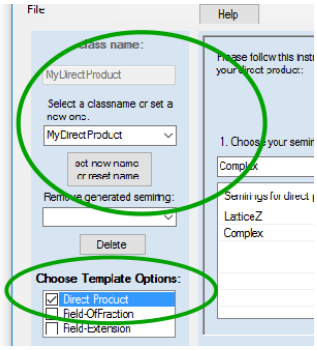


Figure 1: Lower ellipse: Choose an input mask. Top ellipse: Specify the name of the new semiring class.

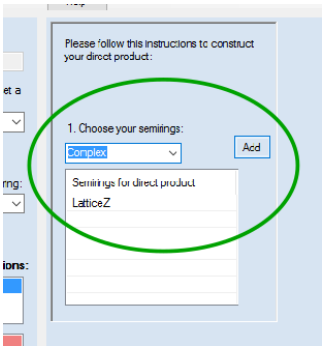


Figure 2: Specify a direct product, add semirings as member fields to the class.

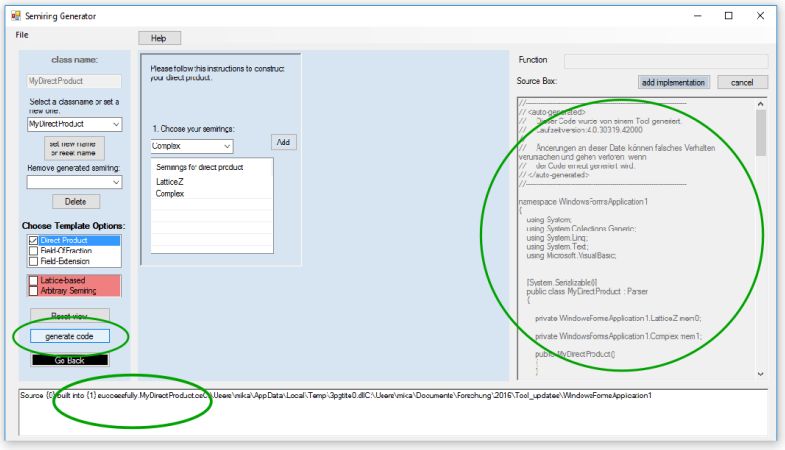


Figure 3: Pressing the button will generate the code visible in the right-hand text-box. The console at the bottom informs the user, whether the source code was compiled and successfully added to the analysis component of PAWS.

4.2 The Analysis Tool

In this section the use of the PAWS analysis component is presented in a brief overview. The illustrations serve to explain our intuition behind the design and the use of PAWS.

In Figure 6 and 7 the first steps for creating a weighted automaton are illustrated. After generating a matrix (Figure 8 and 9), the user can choose one of the available algorithms and wait until the result of the computation is displayed inside the text area (Figure 10).

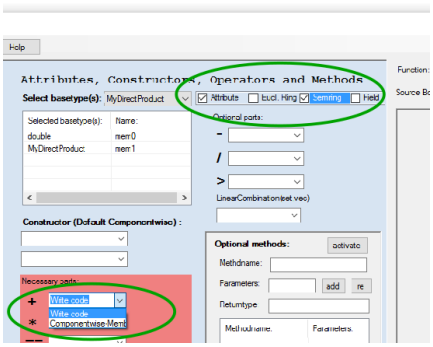


Figure 4: Top ellipse: Specify the type of the class. Bottom ellipse: Select *write code* to implement the addition (+).

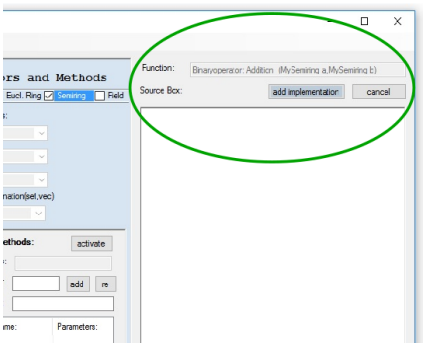


Figure 5: The text box for entering source code will be enabled after selecting *write code* as shown in Figure 4.

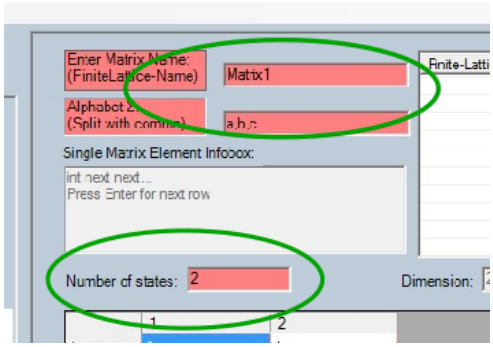


Figure 6: Top ellipse: First determine the name. Bottom ellipse: Type in the number of states.

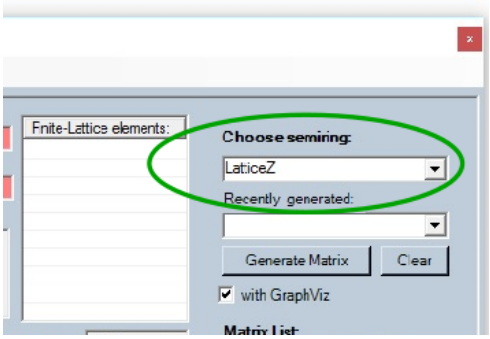


Figure 7: Choose the semiring.

With the PAWS analysis component, besides automata, also finite semirings can be generated and stored for further semiring generation as well as for the analysis.

5 Conclusion, Future Work and Related Work

We have seen that PAWS is a flexible tool to analyse the behaviour of weighted automata and conditional transition systems. The generic approach allows for adding new semirings with a varying degree of support by the tool itself.

Concerning related approaches, we are not aware of analysis tools for language equivalence and the threshold problem for weighted automata.

For the problem of generating semirings dynamically, there exists previous work for solving fixpoint equations over semirings by Esparza, Luttenberger and Schlund [13]. In [13] FPSOLVE is described, a C++ template based tool for solving fixpoint equations over semirings. That is, the tool has a different application scenario than ours. However, the tools share similarities since in FPSOLVE the user also has the possibility to generate new semirings. For this, only the addition, multiplication and Kleene star must be implemented. However, a string constructor must also be specified without automatic support and the main method must be adjusted with the corresponding command-line. PAWS is designed to enable the

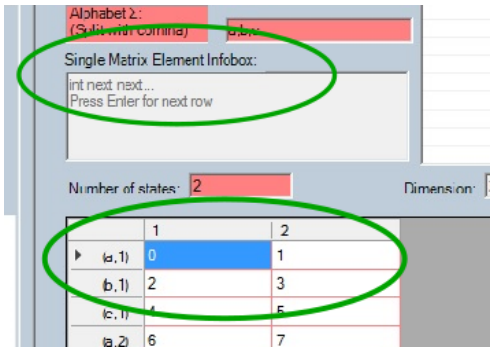


Figure 8: After a semiring has been selected, a description of the expected input is shown above the input area for the matrix.

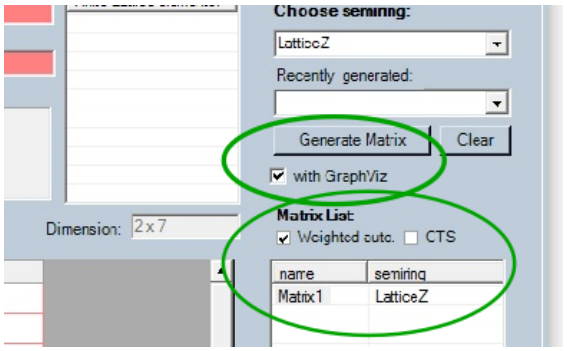


Figure 9: By pressing the button *Generate Matrix*, the matrix is generated in the tool and listed according to its form (Weighted automaton or CTS).

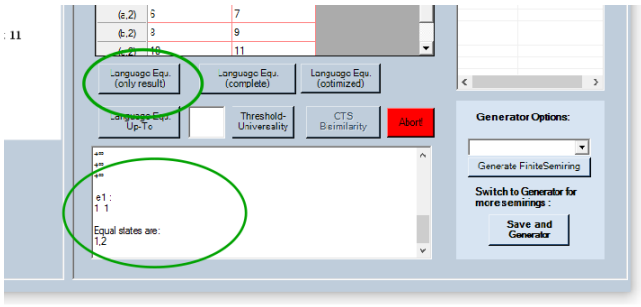


Figure 10: After starting one of the supported algorithms for a transition system, the analysis result will be displayed in the bottom text area.

generation of new semirings for solving linear equations using a graphical user interface and does not change already existing code, which is not part of a semiring class.

In contrast to this, work has already been done on an analysis tool for featured transition systems – which are basically CTS without a notion of upgrades – to analyse software product lines wrt. simulation. In their work [8], Cordy et al. have implemented their model using BDDs as well, yielding a similar speed up as our own approach. The significant differences here lie in the notion of behaviour, since Cordy et al. have focused on simulation relations, whereas we are concerned with bisimulations. Furthermore we capture a notion of upgrade and thus support partial orders of products instead of just abritrary sets of products.

We intend to develop PAWS further in several ways. We are looking for new classes of semirings where solutions of linear equations can be effectively computed, in order to equip those semirings with an improved support from PAWS. Furthermore, we are interested in analysing more extensive case studies, where we will use PAWS to conduct all required analyses.

References

[1] Jiří Adámek, Filippo Bonchi, Mathias Hülsbusch, Barbara König, Stefan Milius & Alexandra Silva (2012): *A Coalgebraic Perspective on Minimization and Determinization*. In: *Proc. of FOSSACS '12*, Springer, pp. 58–73. LNCS/ARCoSS 7213.

[2] Shaull Almagor, Udi Boker & Orna Kupferman (2011): *What's Decidable about Weighted Automata?* In: *Proc. of ATVA '11*, Springer, pp. 482–491. LNCS 6996.

[3] Mariel-Pierre Béal, Slyvain Lombardy & Jacques Sakarovitch (2006): *Conjugacy and Equivalence of Weighted Automata and Functional Transducers*. In: *Prof. of CSR '06*, Springer, pp. 58–69. LNCS 3967.

[4] Harsh Beohar, Barbara König, Sebastian Küpper & Alexandra Silva (2016): *Conditional Transition Systems: A Model for Software Product Lines with Upgrades*. Unpublished, available from `http://www.ti.inf.uni-due.de/fileadmin/public/koenig/cts.pdf`.

[5] Filippo Bonchi, Barbara König & Sebastian Küpper (2017): *Up-To Techniques for Weighted Systems*. In: *Proc. of TACAS '17*, Springer. LNCS, to appear.

[6] Filippo Bonchi & Damien Pous (2013): *Checking NFA equivalence with bisimulations up to congruence*. In: *Proc. of POPL '13*, ACM, pp. 457–468.

[7] Michele Boreale (2009): *Weighted bisimulation in linear algebraic form*. In: *Proc. of CONCUR '09*, Springer, pp. 163–177. LNCS 5710.

[8] Maxime Cordy, Andreas Classen, Gilles Perrouin, Pierre-Yves Schobbens, Patrick Heymans & Axel Legay (2012): *Simulation-based abstractions for software product-line model checking*. In: *Prof. of ICSE '12*, pp. 672–682.

[9] Raymond A. Cuninghame-Green (1979): *Minimax algebra*. Lecture Notes in Economics and Mathematical Systems, Springer-Verlag.

[10] Abhijit Das & C. E. Veni Madhavan (2009): *Public-Key Cryptography: Theory and Practice*. Pearson Education. pp. 295-296.

[11] Brian A. Davey & Hilary A. Priestley (2002): *Introduction to lattices and order*. Cambridge University Press.

[12] Manfred Droste, Werner Kuich & Heiko Vogler, editors (2009): *Handbook of Weighted Automata*. Springer.

[13] Javier Esparza, Michael Luttenberger & Maximilian Schlund (2014): *FPsolve: A Generic Solver for Fixpoint Equations over Semirings*. In: *Proc. of CIAA '14*, Springer, pp. 1–15. LNCS 8587.

[14] Stefan Kiefer, Andrzej S. Murawski, Joel Ouaknine, Bjoern Wachter & James Worrell (2011): *Language Equivalence for Probabilistic Automata*. In: *Proc. of CAV '11*, Springer, pp. 526–540. LNCS 6806.

[15] Barbara König & Sebastian Küpper (2016): *A generalized partition refinement algorithm, instantiated to language equivalence checking for weighted automata*. *Soft Computing*, pp. 1–18, doi:10.1007/s00500-016-2363-z. Available at `http://dx.doi.org/10.1007/s00500-016-2363-z`.

[16] Daniel Krob (1994): *The equality problem for rational series with multiplicities in the tropical semiring is undecidable*. *International Journal of Algebra and Computation* 4(3), pp. 405–425.

[17] Christine Mika (2015): *Ein generisches Werkzeug für Sprachäquivalenz bei gewichteten Automaten*. Master's thesis, Universität Duisburg-Essen.

[18] Marcel-Paul Schützenberger (1961): *On the Definition of a Family of Automata*. *Information and Control* 4(2–3), pp. 245–270.

Mining for Safety using Interactive Trace Analysis

Stephan Brandauer
Uppsala University
stephan.brandauer@it.uu.se

Tobias Wrigstad
Uppsala University
tobias.wrigstad@it.uu.se

This paper presents the results of a trace-based study of object and reference properties on a subset of the DaCapo benchmark suite with the intent to uncover facts about programs that can be leveraged by type systems, compilers and run-times. In particular, we focus on aliasing, and immutability, based on their recent application in the literature.

To facilitate analyses like this one, we previously created Spencer (`http://spencer-t.racing`, [8]), a web based tool and API that hosts dynamic trace data and enables researchers to query and analyse the data. In this paper we only use data that are openly accessible via Spencer's API – all code written for this paper (not counting Spencer itself) amounts to 13 lines of bash and 260 lines of python to plot the results.

We find that while Java allows aliasing and mutation by default, objects are often unique, unique on the heap, immutable, or stack-bound – 97.7% of objects fulfill at least one of these properties. Furthermore, uniqueness and immutability, or their absence, are class-properties, not object-properties: *e.g.,*, it is surprisingly rare for classes to produce both immutable and mutable instances.

Although we use a different, more fine-grained, methodology, our findings confirm prior results.

1 Introduction

In this paper, we study the object graphs that make up object-oriented programs to uncover common properties about the object structures and how object aliasing and mutation are used. Our main motivation for this work is the wealth of work on controlling and managing object aliasing, *e.g.,*various proposals for ownership [10, 13] and uniqueness [6, 7], and various forms of immutability [18, 22]. With this study, we wish to understand how "programs in the wild" (where such properties are not enforced) compare to such proposals (which impose restrictions), and to motivate their existence. We seek to answer questions such as "how many objects are aliased?", or "are immutable objects used for longer periods of times than non-immutable objects?" There is a substantial amount of programming languages work on abstractions like uniqueness, immutability, and combinations thereof [13, 10, 2, 1, 15].

This is not the first paper to ask these or similar questions, or to study the shape or structure of the heap (see *e.g.,*[16, 17, 22, 20, 9, 14, 18]) and in some respects, this paper reproduces results studied by other authors using different, *and arguably more fine-grained*, techniques. Whereas prior work approaches these questions using static analysis or heap snapshotting, we employ a tracing approach in which we are able to study the life-cycle of all individual objects in a running program. In particular, our traces include all reads and writes to local variables, which are ignored by previous dynamic analyses. This avoids the conservative over-approximation built into static analysis, and avoids false positives due to incomplete data when snapshotting. Thus, to the best of our knowledge, we are the first to attempt to answer such broad research questions with such high-resolution data.

The data sets (the traces in this paper, combined, weigh in at 680GB) are hosted by SPENCER (`http://spencer-t.racing`, [8]), a web based tool that lets users query dynamic program traces using their web browser. All results in this paper are produced using Spencer's data.

Submitted to:
QAPL'17

	Name	Objects	Log	
1.	`luindex`	81,158	5.8GB	37.906.637
2.	`pmd`	131,462	2.7GB	18.107.255
3.	`fop`	521,789	10GB	63.957.143
4.	`batik`	526,945	21GB	134.864.425
5.	`xalan`	1,133,391	48GB	302.083.030
6.	`lusearch`	1,212,743	61GB	380.164.919
7.	`sunflow`	2,419,900	91GB	569.648.600
8.	`h2`	6,655,852	207GB	1.468.550.379
9.	`avrora`	932,085	236GB	1.514.972.478
	Total:	13,615,325	≈680GB	4,490,254,866

Table 1: Currently loaded benchmarks, a similar list can be found in the tool: `http://spencer-t.racing/datasets`.

The main contributions of this paper are the results from analyses of program traces to find evidence of immutability, and uniqueness from 9 programs from the DaCapo benchmark suite and a comparison with and discussion of results from prior work. We use a more fine-grained approach than previous studies by considering all program events, and stack variables.

2 Methodology

We employ a trace-based method to study the behaviour of objects. Our corpus of programs is the DaCapo benchmark suite release 9.12 [3], limiting ourselves to 9 programs[1] because of the volume of data involved. This suite contains a wide range of workloads, including simulations of micro controllers (`avrora`), a static analysis tool (`pmd`), an in-memory database (`h2`), and others. Table 1 lists the benchmarks we are studying. Tracing allows us to track not only how aliases to each object are created, but where they are stored, and how they are used at a level which previous dynamic analyses do not do. For the 9 programs in our study, we recorded $\approx 4.5 \cdot 10^9$ events such as object creation, field read, field write, etc. To facilitate studies like this one, we created Spencer, a web service that hosts large program traces, and provides a user interface and API to query these datasets. All data used in this paper (and more) can be accessed openly, and we will provide appropriate links in the paper.

Spencer uses a JVMTI agent loaded into the a stock JVM that listens to events in the running program. Because of limitations in these events (*e.g.,*JVMTI does not allow events on stack variables), we also instrument the loaded Java byte code on the fly to emit additional events[2]. This is done on the fly with *no need to manipulate program sources*, and automatically includes any loaded libraries in the analysis. At the time the analysis was made, the tooling infrastructure consisted of more than 10.000 LOC of a mix of C++, Java, Scala, and Javascript. These files are loaded into a relational database (PostgresSQL) and hosted by a web server. The web server provides a user interface for users to interactively explore data sets (Figure 1), and also a JSON API (`http://spencer-t.racing/doc/api`). For both web based UI and API, the concept of a *query* is central: a query is a selection operation on the set of objects in a trace – it analyses the whole trace, and returns, as its result a set of objects. Queries can be combined by using query

[1] `avrora`, `batik`, `fop`, `h2`, `luindex`, `lusearch`, `pmd`, `sunflow`, and `xalan`.

[2] Our events: object creation; method entry and exit; field loads and stores; variable loads and stores.

combinators, and the server will cache results that it computed in the database to improve performance.

2.1 Queries

Queries being selections, the queries that a user submits are literally translated to SQL. To run a query, it is enough to access the URL http://spencer-t.racing/query/⟨datasetname⟩/⟨query⟩, the page also contains links to the corresponding API call that will just return the selected objects in JSON format at the bottom.

This paper is not mainly about Spencer; but we will give a brief explanation of the queries[3] that we will will use here:

Query	*Explanation*
ImmutableObj()	This query selects all objects that were modified only in their constructor, but never after.
UniqueObj()	This query selects all objects that had at most one reference from fields or (stack-) variables at one time, but never more.
StackBoundObj()	This query selects all objects that are never reachable from any field.
HeapUniqueObj()	This query selects all objects that had at most one reference from fields at one time, but any number of references from variables.
HeapDeeply(q)	If q selects objects from a trace, then HeapDeeply(q) selects all the objects that are selected by q from which, following only fields, only objects in q are reachable.
Or($q_1 \dots q_N$)	This query selects all objects that are selected by at least one of the inner queries.

3 The Properties of Interest in our Study

We now describe the properties of programs, individual objects and references that are the subjects of our study. For each property, we explain it briefly; outline why it is important and how it is used in the programming language literature; and state the definitions of what we have studied in our traces in relation to the property. A discussion about each property is found in conjunction with the results.

3.1 Property 1: Uniqueness

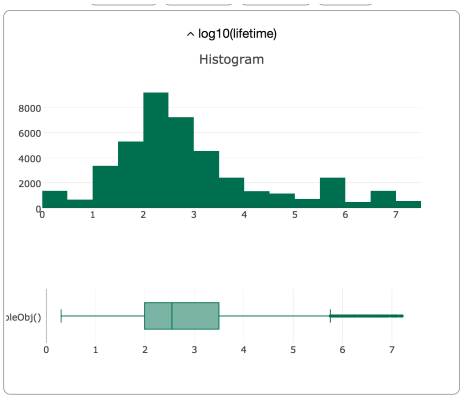
Unique references are references that have no aliases. Unique references simplify reasoning about software, both by programmers and tools. Verifying properties of an object is much easier in the absence of aliasing. When unique references go out of scope, the object they reference can be free'd. Many optimisations are unlocked in compilers due to alias-freedom.

Studying uniqueness through heap snapshots as done by Potanin et al. [20] is simple: count the in-degrees of incoming reference for all objects. However, snapshots have the problem that they are not aware of behaviour in between snapshots – when "no one is looking". We study two definitions of uniqueness:

Heap-Uniqueness A reference is heap-unique if there is, at any one time, at most one reference to it from a live object on the heap, with no restrictions on the number of field references.

[3] These queries are links in the PDF file. Clicking them will bring you to the user interface.

(a) The percentage of objects that are selected by a query, and a sample of them.



(b) Visualisation of the life time of selected objects (the difference between the event index of the last access to the object and the first).

Figure 1: The Spencer user interface as seen from a web browser.

Uniqueness A reference is unique if there is at most one reference to it from either variables or fields. Even though this definition seems to be very constraining, we shall see that a considerable amount of objects and fields fulfill it.

The properties are captured by the Spencer queries HeapUniqueObj()) and UniqueObj(), respectively.

3.2 Property 2: Immutability

Immutable objects are objects that will not change. A classic example of immutable objects in the Java world are strings and boxed primitives such as `Integer` and `Boolean`. Immutability is a powerful property and gives similar reasoning power as uniqueness: a value will not change under foot. The recent years have seen several designs of type systems for Java-like object-oriented programming languages with the aim of simplifying concurrent programming that use some form of immutable object to share data without risking data-races, *e.g.,*[5, 11, 4, 12, 19]. Java does not support immutable objects except for the ability to declare a field as final (a final field has to be assigned to in the object's constructor, and it can never be assigned to outside of the object's contstructor). This is a limited support, *e.g.,* cannot express construction of cyclic immutable structures, or initialisation that is distributed over several parts of a program.

We explore the presence of immutability in three forms:

Shallow-Immutable An object is *shallow immutable* if its fields are never written to – except in its constructor. In Java, this could be handled by final fields, unless the initialisation of the object is complicated or requires some form of delay.

Deeply-Immutable An object is *deeply immutable* if it is immutable and all its fields are deeply immutable. Most immutability constructs that appear in the literature rely on deep immutability.

The properties are captured by the Spencer queries ImmutableObj()) and Deeplu(ImmutableObj()), respectively.

4 Approximating Pseudo Static Properties from Trace Data

Our analysis works on dynamic program traces. We follow an object through the program trace and analyse whether or not the property of interest holds for that object – f.ex., whether the object is immutable. Each analysis partitions the set of known objects into those that satisfy immutability and those that do not. To approximate invariants that could be enforced statically – in particular captured by type system-like annotations as in most works referred to above – we search for patterns among variables, fields and classes:

Field Analysis: For each field F of each class C, we collect all objects that any C-instance ever referred to from the field, yielding the set $O_{C:F}$.

Class Analysis: For each class C, we collect all of its instances, yielding the set O_C.

These sets approximate static properties – "*pseudo static properties*" – in our analysis. For example, if a field F of the class C only stores references to immutable objects (in other words, all objects in $O_{C:F}$ satisfy the immutability property), we hypothesise that there is an invariant in the program that guarantees that all references that could ever be stored in the variable are immutable, too (Section 5.1.1, Section 5.2.1)

We cover field (using the $O_{C:F}$ sets in Section 5.1.1 and Section 5.2.1), and classes (using the O_C sets in Section 5.1.2 and Section 5.2.2). This reasoning is, of course, unsound (it may produce false positives[4]) and should be taken in context with results of sound static analyses (in Section 7) – that will, on the other hand, produce false negatives.

5 Results

We now discuss the outcome of applying our analyses to traces from our program corpus. Since the programs come out of the DaCapo benchmark suite, each program comes with pre-set instructions for how to run it on representative data. Many studies of *e.g.,* performance have been carried out on these programs with the exact same input.

N.B: Results in this section that are not annotated with the name of a specific benchmark are to be read as being a result that summarises the results of all benchmarks.

5.1 Uniqueness and Heap-Uniqueness

Our studies find that 24% of all objects satisfy **Uniqueness** and 45.5% of all objects satisfy **Heap-Uniqueness**. This suggests that aliasing is more commonly happening on the stack and that many objects are either flat or tree-shaped (in fact, the query). The results are visible in Figure 2. Especially interesting is the fact that 97.7% of all objects are "safe" where to be safe means that they are either unique, heap-unique, stack-bound, immutable, or deeply immutable.

In a study from 2004, Potanin et al. [20] find that 13.6% of objects had more than one field pointing to them. We get a similar result: by running the query Or(StackBoundObj() HeapUniqueObj()) – it returns 89.5% on average[5], leaving $\approx$ 10.5% of objects with more than one field reference. Their methodology is different from ours in that they use snapshots of heaps, but not stacks. We are able to show that uniqueness is much lower if stack references are also analysed. Our measuring methodology also works with a continuous view of the heap whereas Potanin's might overlook aliasing that happens in between snapshots.

[4]For example, statically capturing a property can be complicated by value-based overloading. For example, if there can be an assignment from some not always immutable set $O_{C':F'}$ to the variables in $O_{C:M:F}$ when some guard holds true, which guarantees the particular object's immutability.

[5]See `http://spencer-t.racing/json/percentage/pmd/Or(StackBoundObj()%20HeapUniqueObj())`, modify

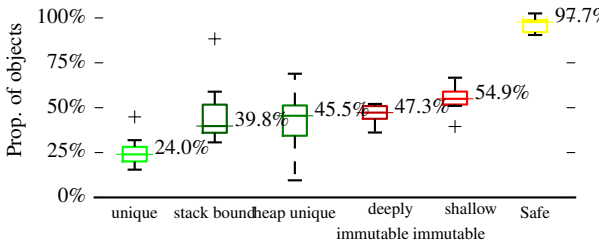


Figure 2: Proportion of **Unique/Stackbound/Heap-Unique/DeepImmutable/Immutable/Safe** objects. A single object can be in several categories, objects that fulfill any definition are classified as "**Safe**". Labels denote the median percentages for all benchmarks.

5.1.1 Unique, Stack-bound, and Heap-Unique Fields

The percentage of heap-unique objects, as shown in Figure 2 varies wildly across different programs. This might imply that heap-uniqueness as a language abstraction is doomed to work only in some niches. However, when we look at the explain pseudo static objects that fields contain, and count those fields that always (and those that never) contain heap-unique objects, we see clear patterns emerge. Figure 3a-e show histograms. Each field goes in the bin according to the percentage of objects it referred to that were selected by the respective query. The distributions are clearly bi-modal – there is lots of fields that rarely contain objects selected, and there is lots of fields that often contain the objects selected by the query.

As programmers, knowing that `x.f` is immutable *most of the time* is of course not very helpful. Invariants are. Our next question is therefore to ask whether there might be static invariants guaranteeing that a property holds *always* or *never* for a given field. This question is what Figure 3f answers: it shows, in green the proportion of fields that contained *only* objects with the given property, and in red the proportion of fields that *never* contained objects with the given property.

A striking property of Figure 3f is that so few fields contain mixes of, say, heap-unique and unique objects. This suggests that invariants about sharing (or its absence) are as common in the wild as invariants about mutability.

5.1.2 Unique, Stack-bound, and Heap-Unique Classes

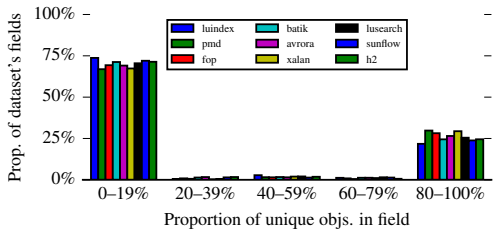
Our results in Figure 4 show[6] that the properties under consideration are predominantly pseudo-static: $\approx 61\%$ of all classes either always or never produce instances that are unique (12% always, and 49% never), $\approx 63\%$ of fields are always or never heap-unique (27% always and 36% never). We don't mention stack-bound objects in these field statistics, as – per definition – no fields ever refer to stack-bound objects.

This results are encouraging for creators of unique type systems as it suggests that in most cases, *a single declaration-site annotation* will be enough to capture uniqueness, rather than annotations on types at *use-site*. Since most programs have far fewer declarations than uses.

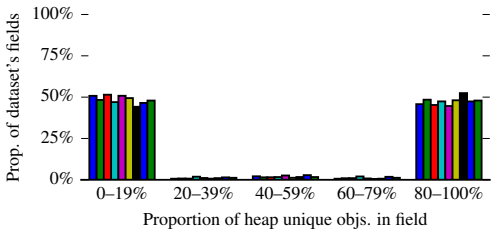
---

the URL for other datasets.

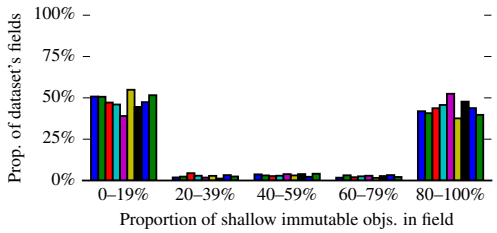
[6]Schema for raw data: `http://spencer-t.racing/json/classpercentage/pmd/Deeply(ImmutableObj())`, and similar for other data sets and queries.



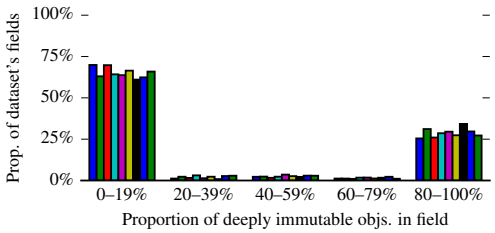
(a) Unique.



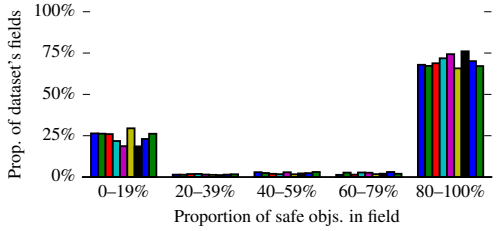
(b) Heap unique.



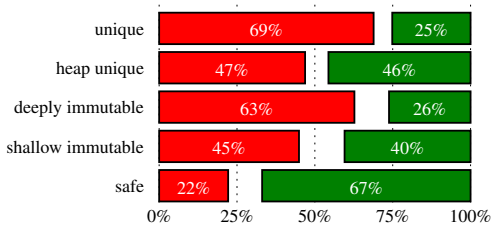
(c) Shallow immutable.



(d) Deeply immutable.

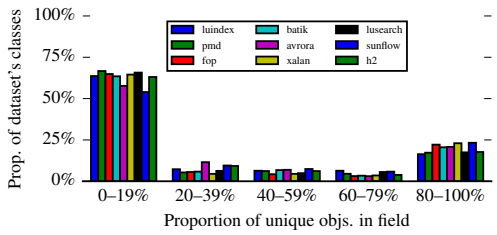


(e) Safe.

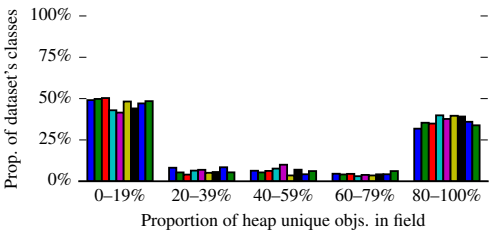


(f) Proportion of fields that contained only objects with the given property (green), not a single one with the given property (red), or other (proportion reported as average of all datasets).

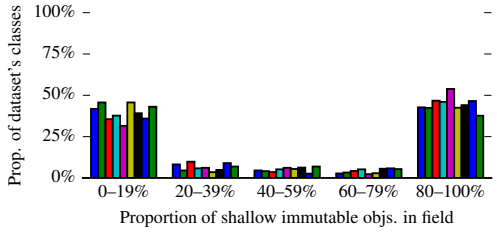
Figure 3: Subfigures a-e: Per syntactic field, out of all the objects it referred to, the percentage of objects with the given property. Fields with a high proportion of objects with the property go to the right-most bin, fields with a low proportion go to the left-most bin. Since benchmarks have different sizes, they also access different numbers of fields in total. To account for this incidental detail, we normalise all bars such that the bar height shows the proportion of all fields *in one benchmark* – in other words, the bars for one color will always add up to 100%.



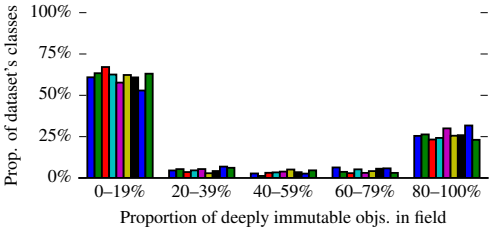
(a) Unique.



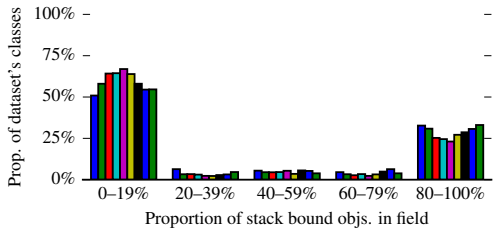
(b) Heap unique.



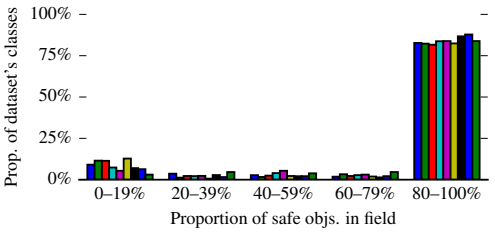
(c) Shallow immutable.



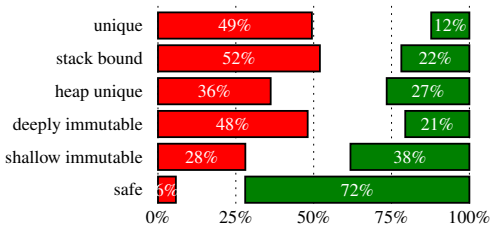
(d) Deeply immutable.



(e) Stack-bound.



(f) Safe.



(g) This shows, for each property, the average across data sets of the proportion of classes that *only* (green, right)/*never* (red, left) produced instances that had the property. We infer, unsoundly, that there exists an invariant that guarantees the property holds statically. Classes with less than 10 instances are ignored.

Figure 4: Subfigures a-f: Per class, out of all the instances this class had, proportion of objects that had the given property. Classes with a high proportion of instances having the property go to the right-most bin; classes with a low proportion of instances having the property go to the left-most bin.

5.2 Immutability

Our analysis in Fig. 2 shows that 54.9% of all objects are shallow-immutable, and 47.9% are deeply immutable. Especially the latter number is encouraging, as deep immutability is a very powerful property when reasoning about code.

5.2.1 Immutable Fields

Our analysis finds that it is quite common for *fields* to refer to deeply immutable, and immutable objects. In Figure 3, we see that 26% and 40% of fields refer to only deeply or shallow immutable objects respectively. The (unsound) conclusion, again, is that up to 26%/40% of fields might be annotated with such qualifiers in a language that provides them. On the contrary, 63%/45% of fields *never* contain deeply/shallow immutable objects.

5.2.2 Immutable Classes

As shown in Figure 4, shallow and deep immutability (or their absence), just like uniqueness, is often a property of the class, not something that varies with individual objects. A third (38%) of all classes produce only immutable instances and half of classes never produce stack-bound objects.

5.3 Summary

In summary, we find that *aliasing is more common on the heap* (Fig. 2): uniqueness is more rare than heap-uniqueness. Figure 3f shows that it is very common for fields to have pseudo-static invariants – there seems to be a lot of knowledge in programmers' minds that is not captured by Java semantics that does not allow distinguishing between the green, the red, or the white part of the bars.

Previous dynamic studies have focused their attention on the objects in benchmarks – essentially providing information like what we do in Fig. 2. What is especially interesting is that the variation in some metrics is quite high (especially heap uniqueness and stack-boundedness), yet where we innovate – by looking at pseudo static results – we find that this variance *does not* seem to carry over to fields and classes: Fig. 3a-f and Fig. 4a-e clearly show that all programs under consideration show very similar behaviour. This is good news for researchers who want to build, or implement language abstractions concerning the properties under consideration.

We believe that program correctness, program understanding, and program performance all benefit from clear expression of mutability and aliasing invariants. The reason is that, when reasoning about a program conservatively, language semantics that allow aliasing and mutability are *hindering*. In nine programs in our study, the freedom given by default by Java's static semantics is rarely needed in practise. Consequently, we argue that instead of enabling, it is hindering, by complicating both reasoning and aspects of the deep run-time. This disconnect between freedoms given and freedoms used is overviewed below:

Semantics	*vs.*	**Programs**
Deep immutability is not a language concern	*vs.*	Half of all objects, a quarter of all fields, and a fifth of all classes only have deeply immutable instances (Fig. 2, Fig. 3f, Fig. 4g)
A field's contents can always be aliased	*vs.*	Almost half of fields are always heap-unique, almost half are never heap-unique Fig. 3f.
Objects may live forever	*vs.*	39.8% of objects (median), and 22% of classes are stack-bound (Fig. 2, Fig. 4g).

5.3.1 Uniqueness

Heap uniqueness in fields is much more common than deep immutability, and so is it in classes. Heap uniqueness is an interesting property – it says that there is one "main alias" and several stack aliases that all have limited life time. It can help the garbage collector, or help optimise memory layouts (heap unique fields do not need to be represented as pointer).

For heap-uniqueness, half of all fields could be annotated unique. Thus, we find no strong evidence that Java's design decision in this respect is well-founded. Making fields unique rather than shared by default would increase static safety, could potentially help the garbage collector (the referent of a unique variable can be de-allocated when the variable goes out of scope) and improve performance (the referent of a unique variable could be on the stack). For designers of future object-oriented programming languages, this is encouraging.

5.3.2 Immutability

With respect to annotations on fields, shallow immutability and deep immutability are roughly as common as **Heap-Unique** and **Unique**, respectively – as shown in Figure 3. Simple immutability semantics – where an object is fixed right after creation – seems to be less useful in practise, *e.g.,* because of delayed initialisation. There have been proposals for Java [21] but, to our knowledge, none have yet been included in a mainstream language[7]. From our results, these patterns that these proposals capture are common.

6 Threats to Validity

Dynamic Analysis The single biggest threat to validity of the results obtained is the fact that we use program traces as input, not static analysis. Dynamic analysis can soundly prove some properties, but not others – For instance, a static analysis can easily prove that a class is *immutable*, but not easily prove that a class is *mutable* (there might be mutating code that is unreachable). Dynamic analysis can easily prove that a class is mutable, but not that it's immutable. As Spencer already stores the classes that where used during an execution, it might be possible to add static analysis facilities in the future – then, an analysis could produce both upper and lower bounds for its results.

Our traces are obtained from single runs of each program, meaning we are not able to detect variations in the same program that stem from non-determinism. Over-interpreting our results may perceive static invariants where there are, in fact, none. This is why our results, should be taken as *research hypotheses*, rather than fact.

Non-Instrumented Classes We are unable to instrument all classes because they are loaded early in the bootstrap process of the JVM, meaning these classes are not covered in our analysis. We mitigate this risk by recording most classes that are being loaded and transforming them as soon as it is safe to do so (at the start of the application). This means that – to the best of our knowledge – all application code and all data structures of the standard library is instrumented.

Native Methods Native methods are rare, but still may cause distorted results. Most notably, we do not get events from methods in the `java.util.Arrays` utility class, such as `copyOf`. That method creates a new array and initialises it to contain the same values or references as an existing array. Our analysis will miss the assignments to the array's cells and therefore classify the newly created copied instance as immutable (assuming it is never changed again), instead of stationary, as it

[7] Scala case classes with lazy fields might qualify

would have been classified with a fully instrumented `copyOf` implementation. Since `copyOf` has semantics similar to a constructor – it creates and initialises an array – we do not correct for this case. However, it may be the case that we miss other important side effects or sharing caused by native implementations. We plan to identify the most common native methods systematically and call equivalent mock implementations in Java (that are instrumented) instead.

Benchmarks are Not Representative Our findings are extracted from 9 programs in the well-known and well-studied DaCapo benchmark suite [3]. While the domains of the programs are very different, it cannot be excluded that our benchmarks are not representative of a larger class of Java programs. We have observed very stable behaviour, and it is interesting to grow the set of datasets, especially to ones implemented in different languages. We'd expect to see more

Bugs in Tracing or Analysis Because of the large amount of code involved in obtaining these traces and analysing them, it is likely that there is some bug somewhere. Due to the nature of tracing large applications, verification is very hard except for very small programs. To mitigate this risk – or at least make sure bugs are found, rather than stay hidden forever – our data sets are hosted publicly. We went a long way to make the data convenient to analyse, in part because we want others to be able to fact check our claims.

7 Related Work

A number of studies on aliasing in object-oriented or imperative programming exist in the literature. In contrast to this work, these studies either employ static analysis, or use a snapshot-based approach to collect data at run-time and excludes stack variables. Even though the stack is bound to be relatively small at any given instant, most objects are referenced from the stack at some point. The difference between **Unique** and **Heap-Unique** hints at the impact of considering/ignoring stack references. Static analysis is by nature conservative, meaning the results from static analysis is likely to include many false negatives, *e.g.,* because an analysis is unable to reason about branching in the running program. On the other hand, static analysis-based approaches are a naturally good fit for determining whether certain static information could be propagated through the code.

Snapshot-based approaches are similar in spirit to this work, but rely on sampling instead of tracing, leading to "lower resolution". Following an object through its entire life-cycle through sampling suffers from false positives (*e.g.,* snapshots before and after a violation of a property P for object o will falsely report $P(o)$ holds) and, according to our methodology, false negatives (a snapshot cannot discover that one alias is effectively buried [6]). An obvious up-side of sampling over our approach is the reduced complexity and improved speed of gathering data.

Hackett and Aiken use static analysis on C programs [14]. They find, like we do for Java, that most fields hold unaliased objects, but are unable to reason about the proportion of unaliased object at run-time for particular program runs. Similar to our findings, but for structs (which importantly lack a **this** pointer), how an object is aliased is a declaration-site property (per struct) not a use-site property (per "object"). Following their static analysis approach is less feasible in Java programs because of the additional problems that must be solved, such as dealing with dynamic dispatch and dynamic code generation.

Unkel and Lam [22] use static analysis on Java benchmarks and open source programs to detect the number of stationary fields. Nelson et. al later study the same property using dynamic analysis [18]. The find the number of stationary fields to be in the range of 55–82% in a variety of programs (the static analysis giving the lower bound and the dynamic analysis giving the upper bound). We measure a stronger property of stationary *objects*, which are objects with only stationary fields. Our findings are in line with

the aforementioned results: 73.4% (median) of all objects are stationary, and 72.6% (median) of all classes only produce stationary objects. This suggests that the stationary fields measured by Nelson et. al tend to cluster, rather than being scattered across all classes.

Chis et al. analyse heap snapshots, focusing on memory bloat in Java programs and identifies common problems that are specific to Java programs [9]. Mitchell et al. [16], summarises heap snapshots in ways that programmers may comprehend with a different goal than ours – to identify memory bloat.

Potanin et al. [20] analyse heap snapshots of Java programs and report among other things on uniqueness (on the heap) and ownership. Their analysis of uniqueness only considers pointers on the heap and finds that 87% of fields are unaliased. Our findings are similar but importantly reached through a different methodology.

8 Conclusion and Future Work

We have presented a trace-based analysis of 9 programs from the DaCapo benchmark suite studying uniqueness, stack-boundedness, and immutability. The ultimate goal of this work is capturing the de-facto properties of real-world programs, with a minimal effort on the behalf of the programmer. Properties such as uniqueness, immutability and their cousin, encapsulation, unlock compile-time and run-time optimisations and can avoid catering to pathological cases that rarely show up in practise, but cannot generally be ruled out.

The results we obtain suggest that a significant amount of static invariants relating to aliasing and immutability exists in unaltered Java programs.

In future work, we want to move the Spencer service forward – by adding more data, implementing more analyses, and doing more analyses like this one. We have preliminary evidence that tracking move semantics might be very interesting – by that, we mean tracking objects that may well be aliased, but where only the *newest* alias is used. A common way to implement unique references in practice (C++, Rust use similar constructs) is to consume them after reading them by setting the read variable or field to `null`, but in unmodified Java programs, explicitly nullifying fields is of course rare. Another research direction is to add static analysis to Spencer. We currently already store all classes that are loaded (even classes that were dynamically generated) in the data base. Mixing static and dynamic analysis techniques would potentially give ranges upper and lower bounds on reported results.

References

[1] Paulo Sérgio Almeida (1997): *Balloon Types: Controlling Sharing of State in Data Types*, pp. 32–59. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/BFb0053373.

[2] Paulo Sérgio Almeida (1997): *Balloon Types: Controlling Sharing of State in Data Types*. In Mehmet Akşit & Satoshi Matsuoka, editors: *ECOOP'97 — Object-Oriented Programming*, *Lecture Notes in Computer Science* 1241, Springer Berlin Heidelberg, pp. 32–59.

[3] S. M. Blackburn, R. Garner, C. Hoffman, A. M. Khan, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage & B. Wiedermann (2006): *The DaCapo Benchmarks: Java Benchmarking Development and Analysis*. In: *OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-Oriented Programing, Systems, Languages, and Applications*, ACM Press, New York, NY, USA, pp. 169–190, doi:http://doi.acm.org/10.1145/1167473.1167488.

[4] Bard Bloom, John Field, Nathaniel Nystrom, Johan Östlund, Gregor Richards, Rok Strniša, Jan Vitek & Tobias Wrigstad (2009): *Thorn: robust, concurrent, extensible scripting on the JVM*. In: *ACM SIGPLAN Notices*, 44, ACM, pp. 117–136.

[5] Chandrasekhar Boyapati & Martin Rinard (2001): *A parameterized type system for race-free Java programs*. In: *ACM SIGPLAN Notices*, 36, ACM, pp. 56–69.

[6] John Boyland (2001): *Alias burying: Unique variables without destructive reads*. *Softw., Pract. Exper.* 31(6), pp. 533–553.

[7] John Boyland, James Noble & William Retert (2001): *Capabilities for sharing*. In: *ECOOP 2001—Object-Oriented Programming*, Springer, pp. 2–27.

[8] Stephan Brandauer & Tobias Wrigstad (2017): *Spencer: Interactive Heap Analysis for the Masses*. *under submission*.

[9] Adriana E. Chis, Nick Mitchell, Edith Schonberg, Gary Sevitsky, Patrick O'Sullivan, Trevor Parsons & John Murphy (2011): *Patterns of Memory Inefficiency*. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6813 LNCS, pp. 383–407, doi:10.1007/978-3-642-22655-7_18.

[10] Dave Clarke & Tobias Wrigstad (2003): *External Uniqueness Is Unique Enough*. In Luca Cardelli, editor: *ECOOP 2003 – Object-Oriented Programming*, *Lecture Notes in Computer Science* 2743, Springer Berlin Heidelberg, pp. 176–200.

[11] Dave Clarke, Tobias Wrigstad, Johan Östlund & Einar Johnsen (2008): *Minimal ownership for active objects*. *Programming Languages and Systems*, pp. 139–154.

[12] Sylvan Clebsch, Sophia Drossopoulou, Sebastian Blessing & Andy McNeil (2015): *Deny Capabilities for Safe, Fast Actors*. In: *AGERE15*. Available at `http://www.doc.ic.ac.uk/~scd/fast-cheap-AGERE.pdf`.

[13] Colin S. Gordon, Matthew J. Parkinson, Jared Parsons, Aleks Bromfield & Joe Duffy (2012): *Uniqueness and Reference Immutability for Safe Parallelism*. *SIGPLAN Not.* 47(10), pp. 21–40.

[14] Brian Hackett & Alex Aiken (2006): *How is Aliasing Used in Systems Software?* In: *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, SIGSOFT '06/FSE-14, ACM, New York, NY, USA, pp. 69–80, doi:10.1145/1181775.1181785.

[15] Philipp Haller & Martin Odersky (2010): *Capabilities for Uniqueness and Borrowing*. *24th European Conference on Object-Oriented Programming (ECOOP 2010)* (June), pp. 354–378, doi:10.1007/978-3-642-14107-2_17.

[16] Nick Mitchell (2006): *The Runtime Structure of Object Ownership*. *ECOOP 2006–Object-Oriented Programming*, pp. 74–98, doi:10.1007/11785477_5.

[17] Nick Mitchell, Edith Schonberg & Gary Sevitsky (2009): *Making Sense of Large Heaps*. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5653 LNCS, pp. 77–97, doi:10.1007/978-3-642-03013-0_5.

[18] S Nelson, D J Pearce & J Noble (2013): *Profiling Field Initialisation in Java*. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7687 LNCS, pp. 292–307.

[19] Johan Östlund (2016): *Language Constructs for Safe Parallel Programming on Multi-cores*. Ph.D. thesis, Department of Information Technology, Uppsala University.

[20] Alex Potanin, James Noble & Robert Biddle (2004): *Checking Ownership and Confinement*. *Concurrency Computation Practice and Experience* 16(7), pp. 671–687, doi:10.1002/cpe.799.

[21] Alexander J Summers & Peter Mueller (2011): *Freedom before commitment: a lightweight type system for object initialisation*. In: *ACM SIGPLAN Notices*, 46, ACM, pp. 1013–1032.

[22] Christopher Unkel & Monica S. Lam (2008): *Automatic Inference of Stationary Fields: A Generalization of Java's Final Fields*. In: *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08, ACM, New York, NY, USA, pp. 183–195, doi:10.1145/1328438.1328463.

Logical Characterization of Trace Metrics

Valentina Castiglioni
University of Insubria (IT)
v.castiglioni2@uninsubria.it

Simone Tini
University of Insubria (IT)
simone.tini@uninsubria.it

In this paper we continue our research line on logical characterizations of behavioral metrics obtained from the definition of a metric over the set of logical properties of interest. This time we provide a characterization of both strong and weak trace metric on nondeterministic probabilistic processes, based on a minimal boolean logic $\mathbb{L}$ which we prove to be powerful enough to characterize strong and weak probabilistic trace equivalence. Moreover, we also prove that our characterization approach can be restated in terms of a more classic probabilistic $\mathbb{L}$-model checking problem.

1 Introduction

Behavioral equivalences and modal logics have been successfully employed for the specification and verification of communicating concurrent systems, henceforth processes. The former ones provide a simple and elegant tool for comparing the observable behavior of processes. The latter ones allow for an immediate expression of the desired properties of processes. Since the work of [19] on the Hennessy-Milner logic (HML), these two approaches are connected by means of *logical characterizations* of behavioral equivalences: two processes are behaviorally equivalent if and only if they satisfy the same formulae in the logic. Hence, the characterization of an equivalence subsumes both the fact that the logic is as expressive as the equivalence and the fact that the equivalence preserves the logical properties of processes.

It is common agreement that when also quantitative properties of processes are taken into account a metric semantics is favored over behavioral equivalences, [1, 3, 7, 10, 12, 15–17, 23, 24]. since the latter ones are too sensible to small variations in the probabilistic properties of processes. Therefore, the interest in logical characterizations of the so called *behavioral metrics*, namely the quantitative analogues of equivalences that quantify how far the behavior of two processes is apart, is constantly growing.

In this paper we propose a logical characterization of the *strong* and *weak* variants of the *trace metric* [28] for nondeterministic probabilistic processes (PTSs [27]). To this aim we follow the approach of [8] in which a logical characterization of the bisimilarity metric is provided. We introduce two boolean logics $\mathbb{L}$ and $\mathbb{L}_{\mathrm{w}}$, providing a probabilistic choice operator capturing the probability weights that a process assigns to arbitrary traces, which we prove to characterize resp. the *strong* and *weak probabilistic trace equivalences* of [26]. Such a characterization is obtained by introducing the novel notion of *mimicking formulae of resolutions*, i.e. formulae capturing, for each possible resolution of nondeterminism for a process, all the executable traces as well as the probability weights assigned to them. Then we introduce the notions of *distance between formulae* in $\mathbb{L}$ and $\mathbb{L}_{\mathrm{w}}$ which are 1-bounded (pseudo)metrics assigning to each pair of formulae a suitable quantitative analogue of their syntactic disparities. These lift to metrics over processes, called resp. $\mathbb{L}$-*distance* and $\mathbb{L}_{\mathrm{w}}$-*distance*, corresponding to the Hausdorff lifting of the distance between formulae to the sets of formulae satisfied by the two processes. We prove that our $\mathbb{L}$-distance and $\mathbb{L}_{\mathrm{w}}$-distance correspond resp. to the strong and weak trace metric.

An important feature of our characterization method is that, although it is firmly based on the mimicking formulae of resolutions, it does not actually depend on how these resolutions of nondeterminism are

To appear in EPTCS.

obtained from processes. For instance, in this paper we consider resolutions obtained via a *deterministic scheduler* [4,26], but our approach would not be different when applied to *randomized resolutions* [4,26].

Our approach differs from the ones proposed in the literature in that in general logics equipped with a real-valued semantics are used for the characterization, which is then expressed as

$$d(s,t) = \sup_{\varphi \in L} |[\varphi](s) - [\varphi](t)| \tag{1}$$

where d is the behavioral metric of interest, L is the considered logic and $[\varphi](s)$ denotes the value of the formula φ at process s accordingly to the real-valued semantics [1, 2, 12–14]. In [3] it is proved that the trace metric on Markov Chains (MCs) can be characterized in terms of the probabilistic LTL-model checking problem. Roughly speaking, a characterization as in (1) is obtained from the boolean logic LTL by assigning a real-valued semantics to it, defined by exploiting the probabilistic properties of the MC: the value of a formula $\varphi \in$ LTL at state s is given by the probability of s to execute a run satisfying φ. In this paper we show that we can obtain a similar result by means of our distance between formulae. More precisely, we provide an alternative characterization of the trace metric on PTSs $\mathbf{d}_T$ in terms of the probabilistic $\mathbb{L}$-model checking problem. In detail, we define a real-valued semantics for $\mathbb{L}$ by assigning to each formula $\Psi \in \mathbb{L}$ at process s the value $[\Psi](s)$ corresponding to the minimal distance between Ψ and any formula satisfied by s. Thus we could use this real-valued semantics to verify whether process s behaves within an allowed tolerance wrt. to the specification given by the formula Ψ. Then, by exploiting some properties of the Hausdorff metric, we will be able to conclude that $\mathbf{d}_T(s,t) = \sup_{\Psi \in \mathbb{L}} | [\Psi](s) - [\Psi](t) |$ thus giving that the verification of any $\mathbb{L}$-formula in s cannot differ from its verification in t for more than $\mathbf{d}_T(s,t)$ which, in turn, constitutes the maximal observable error in the approximation of s with t.

We can summarize our contributions as follows:

1. Logical characterization of both strong and weak trace metric: we define a distance on the class of formulae $\mathbb{L}$ (resp. $\mathbb{L}_{\mathrm{w}}$) and we prove that the strong (resp. weak) trace metric between two processes equals the syntactic distance between the sets of formulae satisfied by them.
2. Logical characterization of strong trace metric in terms of a probabilistic $\mathbb{L}$-model checking problem: by means of the distance between formulae we equip $\mathbb{L}$ with a real-valued semantics and we use it to establish a characterization of the trace metric as in (1).
3. Logical characterization of both strong and weak probabilistic trace equivalence: by exploiting the notion of mimicking formula, we prove that two processes are strong (resp. weak) trace equivalent if and only if they satisfy the same (resp. syntactically equivalent) formulae in $\mathbb{L}$ (resp. $\mathbb{L}_{\mathrm{w}}$).

2 Background

2.1 Nondeterministic probabilistic transition systems

Nondeterministic probabilistic transition systems [27] combine LTSs [22] and discrete time Markov chains [18, 29], allowing us to model reactive behavior, nondeterminism and probability.

As state space we take a set $\mathbf{S}$, whose elements are called *processes*. We let $s,t,\ldots$ range over $\mathbf{S}$. Probability distributions over $\mathbf{S}$ are mappings $\pi\colon \mathbf{S} \to [0,1]$ with $\sum_{s\in\mathbf{S}} \pi(s) = 1$ that assign to each $s \in \mathbf{S}$ its probability $\pi(s)$. By $\Delta(\mathbf{S})$ we denote the set of all distributions over $\mathbf{S}$. We let $\pi,\pi',\ldots$ range over $\Delta(\mathbf{S})$. For $\pi \in \Delta(\mathbf{S})$, we denote by $\mathsf{supp}(\pi)$ the support of π, namely $\mathsf{supp}(\pi) = \{s \in \mathbf{S} \mid \pi(s) > 0\}$. We consider only distributions with *finite* support. For $s \in \mathbf{S}$ we denote by δ_s the *Dirac distribution* defined

by $\delta_s(s) = 1$ and $\delta_s(t) = 0$ for $s \neq t$. The convex combination $\sum_{i\in I} p_i\pi_i$ of a family $\{\pi_i\}_{i\in I}$ of distributions $\pi_i \in \Delta(\mathbf{S})$ with $p_i \in (0,1]$ and $\sum_{i\in I} p_i = 1$ is defined by $(\sum_{i\in I} p_i\pi_i)(s) = \sum_{i\in I}(p_i\pi_i(s))$ for all $s \in \mathbf{S}$.

Definition 1 (PTS, [27])**.** A *nondeterministic probabilistic labeled transition system (PTS)* is a triple $(\mathbf{S}, \mathscr{A}, \rightarrow)$, where: (i) $\mathbf{S}$ is a countable set of processes, (ii) $\mathscr{A}$ is a countable set of *actions*, and (iii) $\rightarrow \subseteq \mathbf{S} \times \mathscr{A} \times \Delta(\mathbf{S})$ is a *transition relation*.

We call $(s,a,\pi) \in \rightarrow$ a *transition*, and we write $s \xrightarrow{a} \pi$ for $(s,a,\pi) \in \rightarrow$. We write $s \xrightarrow{a}$ if there is a distribution $\pi \in \Delta(\mathbf{S})$ with $s \xrightarrow{a} \pi$, and $s \not\xrightarrow{a}$ otherwise. Let $\mathrm{init}(s) = \{a \in \mathscr{A} \mid s \xrightarrow{a}\}$ denote the set of the actions that can be performed by s. Let $\mathrm{der}(s,a) = \{\pi \in \Delta(\mathbf{S}) \mid s \xrightarrow{a} \pi\}$ denote the set of the distributions reachable from s through action a. Let $\mathrm{dpt}(s)$ denote the *depth* of s, namely the maximal number of sequenced transitions that can be performed from s, defined by $\mathrm{dpt}(s) = 0$, if $\mathrm{init}(s) = \emptyset$, and $\mathrm{dpt}(s) = 1 + \sup_{a\in\mathrm{init}(s),\pi\in\mathrm{der}(s,a),t\in\mathsf{supp}(\pi)} \mathrm{dpt}(t)$, otherwise. We say that a process $s \in \mathbf{S}$ is *image-finite* if for all actions $a \in \mathrm{init}(s)$ the set $\mathrm{der}(s,a)$ is finite [20], and that s has *finite depth* if $\mathrm{dpt}(s)$ is finite. Finally, we denote as *finite* the image-finite processes with finite depth. In this paper we consider only processes that are finite.

Throughout the paper we will introduce some equivalence relations on traces and on modal formulae. To deal with the equivalence of probability distributions over these elements, we need to introduce the notion of *lifting* of a relation.

Definition 2. Let X be any set. Consider a relation $\mathscr{R} \subseteq X \times X$. Then the *lifting* of $\mathscr{R}$ is the relation $\mathscr{R}^\dagger \subseteq \Delta(X) \times \Delta(X)$ with $\pi \mathscr{R}^\dagger \pi'$ if whenever $\pi = \sum_{i\in I} p_i\delta_{x_i}$ then $\pi' = \sum_{i\in I, j_i\in J_i} p_i p_{j_i} \delta_{y_{j_i}}$ with $\sum_{j_i\in J_i} p_{j_i} = p_i$ and $x_i \mathscr{R} y_{j_i}$ for all $j_i \in J_i$.

Moreover, we can lift relations to relations over sets. Given a relation $\mathscr{R} \subseteq X \times Y$, we say that two subsets $X' \subseteq X, Y' \subseteq Y$ are in relation $\mathscr{R}$, notation $X' \mathscr{R} Y'$, iff (i) for each $x \in X'$ there is an $y \in Y'$ with $x \mathscr{R} y$, and (ii) for each $y \in Y'$ there is an $x \in X'$ with $x \mathscr{R} y$.

2.2 Strong probabilistic trace equivalence

A probabilistic trace equivalence is a relation over $\mathbf{S}$ that equates processes $s,t \in \mathbf{S}$ if for all resolutions of nondeterminism they can mimic each other's sequences of transitions with the same probability.

Definition 3 (Computation, [4])**.** Let $P = (\mathbf{S}, \mathscr{A}, \rightarrow)$ be a PTS and $s,s' \in \mathbf{S}$. We say that $c := s_0 \overset{a_1}{\twoheadrightarrow} s_1 \overset{a_2}{\twoheadrightarrow} s_2 \ldots s_{n-1} \overset{a_n}{\twoheadrightarrow} s_n$ is a *computation* of P of length n from $s = s_0$ to $s' = s_n$ iff for all $i = 1,\ldots,n$ there exists a transition $s_{i-1} \xrightarrow{a_i} \pi_i$ in P such that $s_i \in \mathsf{supp}(\pi_i)$, with $\pi_i(s_i)$ being the *execution probability* of step $s_{i-1} \overset{a_i}{\twoheadrightarrow} s_i$ conditioned on the selection of transition $s_{i-1} \xrightarrow{a_i} \pi_i$ of P at s_{i-1}. We denote by $\Pr(c) = \prod_{i=1}^n \pi_i(s_i)$ the product of the execution probabilities of the steps in c.

Let $s,s',s'' \in \mathbf{S}$. Given any computation $c' = s' \overset{a_1}{\twoheadrightarrow} s_1 \overset{a_2}{\twoheadrightarrow} \ldots \overset{a_n}{\twoheadrightarrow} s''$ from s' to s'', we write $c = s \overset{a}{\twoheadrightarrow} c'$ if $c = s \overset{a}{\twoheadrightarrow} s' \overset{a_1}{\twoheadrightarrow} \ldots \overset{a_n}{\twoheadrightarrow} s''$ is a computation from s to s''.

Given a process $s \in \mathbf{S}$, we say that c is a computation from s if there is a process s' such that c is a computation from s to s'. We denote by $\mathscr{C}(s)$ the set of computations from s. We say that a computation c from process s is *maximal* if it is not a proper prefix of any other computation from s. We denote by $\mathscr{C}_{\max}(s) \subseteq \mathscr{C}(s)$ the subset of the maximal computations from s. Since we are considering finite processes, all computations are guaranteed to be of finite length.

We denote by $\mathscr{A}^\star$ the set of sequences of actions in $\mathscr{A}$ and we call *trace* any element $\alpha \in \mathscr{A}^\star$. We use the special symbol $\varepsilon \notin \mathscr{A}$ to denote the empty trace. We say that a computation is *compatible* with the trace $\alpha \in \mathscr{A}^\star$ iff the sequence of actions labeling the computation steps is equal to α. We

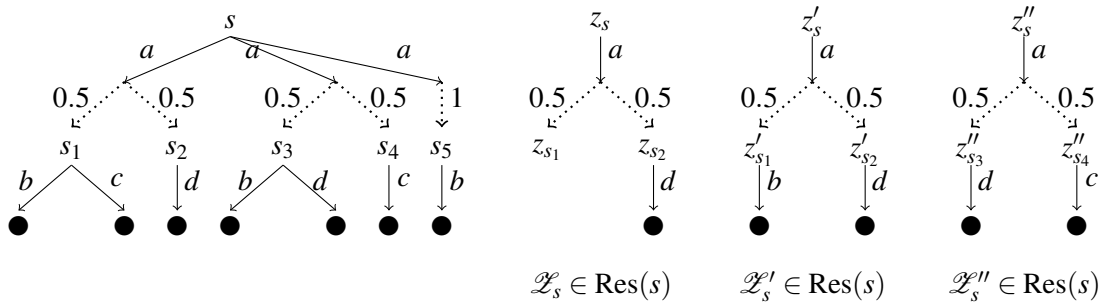


Figure 1: An example of three distinct resolutions for process s. Black circles stand for the probability distribution δ_{nil}, with nil process that cannot execute any action.

denote by $\mathscr{C}(s,\alpha) \subseteq \mathscr{C}(s)$ the set of computations of process s which are compatible with trace α and by $\mathscr{C}_{\max}(s,\alpha) \subseteq \mathscr{C}(s,\alpha)$ we denote the set of maximal computations of s that are compatible with α. Then, given any $\mathscr{C} \subseteq \mathscr{C}(s)$, we define $\Pr(\mathscr{C}) = \sum_{c\in\mathscr{C}} \Pr(c)$.

Definition 4. Let $s \in \mathbf{S}$ and consider any $c \in \mathscr{C}(s)$. We denote by $\mathrm{Tr}(c) \in \mathscr{A}^\star$ the trace to which c is compatible. We extend this notion to sets by letting $\mathrm{Tr}(\mathscr{C}') = \{\mathrm{Tr}(c) \mid c \in \mathscr{C}'\}$ for any $\mathscr{C}' \subseteq \mathscr{C}(s)$. We say that $\mathrm{Tr}(\mathscr{C}(s))$ is the *set of traces* of s and $\mathrm{Tr}(\mathscr{C}_{\max}(s))$ is the *set of maximal traces* of s.

To establish trace equivalence we need first to deal with nondeterministic choices of processes. To this aim, we consider all possible resolutions of nondeterminism one by one. Using the notation of [4], our resolutions correspond to the resolutions obtained via a *deterministic scheduler* (see Fig. 1 for an example).

Definition 5 (Resolution, [4])**.** Let $P = (\mathbf{S}, \mathscr{A}, \rightarrow)$ be a PTS and $s \in \mathbf{S}$. We say that a PTS $\mathscr{Z} = (Z, \mathscr{A}, \rightarrow_{\mathscr{Z}})$ is a *resolution* for s iff there exists a state correspondence function $\mathrm{corr}_{\mathscr{Z}}\colon Z \to \mathbf{S}$ such that $s = \mathrm{corr}_{\mathscr{Z}}(z_s)$ for some $z_s \in Z$, called the *initial state* of $\mathscr{Z}$, and moreover it holds that:

- $z_s \notin \mathsf{supp}(\pi)$ for any $\pi \in \bigcup_{z\in Z, a\in\mathscr{A}} \mathrm{der}(z,a)$.
- Each $z \in Z \setminus \{z_s\}$ is such that $z \in \mathsf{supp}(\pi)$ for some $\pi \in \bigcup_{z'\in Z\setminus\{z\}, a\in\mathscr{A}} \mathrm{der}(z',a)$.
- Whenever $z \xrightarrow{a}_{\mathscr{Z}} \pi$, then $\mathrm{corr}_{\mathscr{Z}}(z) \xrightarrow{a} \pi'$ with $\pi(z') = \pi'(\mathrm{corr}_{\mathscr{Z}}(z'))$ for all $z' \in Z$.
- Whenever $z \xrightarrow{a_1}_{\mathscr{Z}} \pi_1$ and $z \xrightarrow{a_2}_{\mathscr{Z}} \pi_2$ then $a_1 = a_2$ and $\pi_1 = \pi_2$.

We let $\mathrm{Res}(s)$ be the set of resolutions for s and $\mathrm{Res}(\mathbf{S}) = \bigcup_{s\in\mathbf{S}} \mathrm{Res}(s)$ be the set of all resolutions on $\mathbf{S}$.

Strong probabilistic trace equivalence equates two processes if their resolutions can be matched so that they assign the same probability to all traces.

Definition 6 (Strong probabilistic trace equivalence, [4, 26])**.** Let $P = (\mathbf{S}, \mathscr{A}, \rightarrow)$ be a PTS. We say that $s,t \in \mathbf{S}$ are *strong probabilistic trace equivalent*, notation $s \approx_{\text{st}} t$, iff it holds that:

- For each resolution $\mathscr{Z}_s \in \mathrm{Res}(s)$ of s there is a resolution $\mathscr{Z}_t \in \mathrm{Res}(t)$ of t such that for all traces $\alpha \in \mathscr{A}^\star$ we have $\Pr(\mathscr{C}(z_s,\alpha)) = \Pr(\mathscr{C}(z_t,\alpha))$.
- For each resolution $\mathscr{Z}_t \in \mathrm{Res}(t)$ of t there is a resolution $\mathscr{Z}_s \in \mathrm{Res}(s)$ of s such that for all traces $\alpha \in \mathscr{A}^\star$ we have $\Pr(\mathscr{C}(z_t,\alpha)) = \Pr(\mathscr{C}(z_s,\alpha))$.

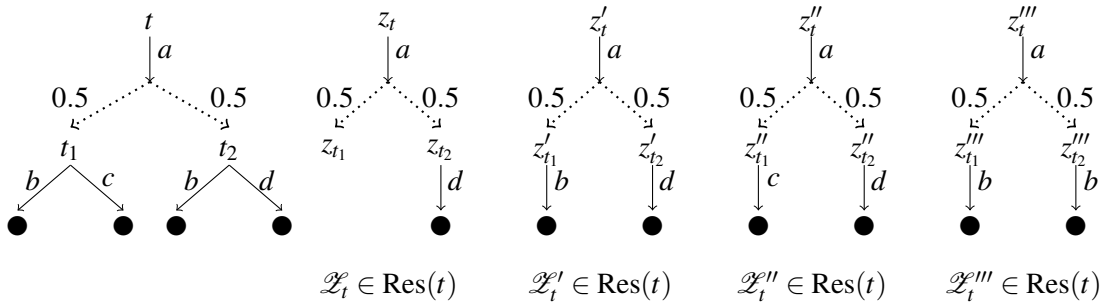


Figure 2: Process t is strong trace equivalent to process s in Fig. 1

***Example* 1.** Consider process s in Fig. 1 and process t in Fig. 2. We have that $s \approx_{\mathrm{st}} t$. Briefly, it is immediate to check that the three resolutions $\mathscr{Z}_s, \mathscr{Z}'_s, \mathscr{Z}''_s \in \mathrm{Res}(s)$ in Fig. 1 are matched resp. by the three resolutions $\mathscr{Z}_t, \mathscr{Z}'_t, \mathscr{Z}''_t \in \mathrm{Res}(t)$ in Fig. 2. Moreover, for all other resolutions, we notice that accordingly to the chosen resolutions for processes t_1 and t_2, process s can always match their traces and related probabilities by selecting the proper a-branch. In particular, resolution $\mathscr{Z}'''_t \in \mathrm{Res}(t)$ in Fig. 2 is matched by the resolution for s corresponding to the rightmost a-branch.

2.3 Weak probabilistic trace equivalence

We extend the set of actions $\mathscr{A}$ to the set $\mathscr{A}_\tau$ containing also the silent action τ. We let $\mathfrak{a}$ range over $\mathscr{A}_\tau$.

Usually, traces are not distinguished by any occurrence of τ in them [28]. Hence, we introduce the notion of *equivalence of traces*.

Definition 7 (Equivalence of traces)**.** The relation of *equivalence of traces* $\equiv_{\mathrm{w}} \subseteq \mathscr{A}_\tau^\star \times \mathscr{A}_\tau^\star$ is the smallest equivalence relation satisfying 1. $\varepsilon \equiv_{\mathrm{w}} \varepsilon$ and 2. given $\alpha = \mathfrak{a}_1\alpha'$, $\beta = \mathfrak{a}_2\beta'$ we have $\alpha \equiv_{\mathrm{w}} \beta$ iff

- either $\mathfrak{a}_1 = \tau$ and $\alpha' \equiv_{\mathrm{w}} \beta$,
- or $\mathfrak{a}_2 = \tau$ and $\alpha \equiv_{\mathrm{w}} \beta'$
- or $\mathfrak{a}_1 = \mathfrak{a}_2$ and $\alpha' \equiv_{\mathrm{w}} \beta'$.

For each trace $\alpha \in \mathscr{A}_\tau^\star$, we denote by $[\alpha]_{\mathrm{w}}$ the equivalence class of α with respect to $\equiv_{\mathrm{w}}$, namely $[\alpha]_{\mathrm{w}} = \{\beta \in \mathscr{A}_\tau^\star \mid \beta \equiv_{\mathrm{w}} \alpha\}$. Moreover, for each computation c, we let $\mathrm{Tr}_{\mathrm{w}}(c) = [\mathrm{Tr}(c)]_{\mathrm{w}}$.

Given any process $s \in \mathbf{S}$ and any trace $\alpha \in \mathscr{A}_\tau^\star$, we say that a computation $c \in \mathscr{C}(s)$ is in $\mathscr{C}^{\mathrm{w}}(s,\alpha)$ iff $\mathrm{Tr}(c) \equiv_{\mathrm{w}} \alpha$ and c is not a proper prefix of any other computation in $\mathscr{C}^{\mathrm{w}}(s,\alpha)$. This is to avoid to count multiple times the same execution probabilities in the evaluation of $\Pr(\mathscr{C}^{\mathrm{w}}(s,\alpha))$.

Definition 8 (Weak probabilistic trace equivalence)**.** Let $P = (\mathbf{S}, \mathscr{A}, \rightarrow)$ be a PTS. We say that $s,t \in \mathbf{S}$ are *weak probabilistic trace equivalent*, notation $s \approx_{\mathrm{wt}} t$, iff it holds that:

- For each resolution $\mathscr{Z}_s \in \mathrm{Res}(s)$ of s there is a resolution $\mathscr{Z}_t \in \mathrm{Res}(t)$ of t such that for all traces $\alpha \in \mathscr{A}^\star$ we have $\Pr(\mathscr{C}^{\mathrm{w}}(z_s,\alpha)) = \Pr(\mathscr{C}^{\mathrm{w}}(z_t,\alpha))$.
- For each resolution $\mathscr{Z}_t \in \mathrm{Res}(t)$ of t there is a resolution $\mathscr{Z}_s \in \mathrm{Res}(s)$ of s such that for all traces $\alpha \in \mathscr{A}^\star$ we have $\Pr(\mathscr{C}^{\mathrm{w}}(z_t,\alpha)) = \Pr(\mathscr{C}^{\mathrm{w}}(z_s,\alpha))$.

3 Trace metrics

In this section we introduce the quantitative analogues of strong and weak probabilistic trace equivalence, namely the *strong* and *weak trace metric*, resp., which are 1-bounded pseudometrics that quantify how much the behavior of two processes is apart wrt. the strong (resp. weak) probabilistic trace semantics. Our metrics are a revised version of the trace metric proposed in [28]. Briefly, in [28] there is a distinction between the notions of *path* and *trace*: any $\alpha \in \mathscr{A}_\tau^\star$ is called path and the trace related to a path is obtained by deleting any occurrence of τ from it. The metric in [28] is then defined only on traces and it has inspired our strong trace metric. In the present paper we distinguish between the strong and the weak case and we regain the results in [28] by our equivalence of traces: the weak trace metric coincides with the strong one on the quotient space wrt. $\equiv_{\mathrm{w}}$.

3.1 The Kantorovich and Hausdorff lifting functionals

In the literature we can find several examples of behavioral metrics on systems with probability and nondeterminism (see among others [1,5,6,10,12,28]). In this paper we follow the approach of [6,10,28] in which two kind of metrics are combined to obtain a metric on the system. The *Kantorovich metric* [21] quantifies the disparity between the probabilistic properties of processes and it is defined by means of the notion of *matching*. For any set X, a matching for distributions $\pi, \pi' \in \Delta(X)$ is a distribution over the product space $\mathfrak{w} \in \Delta(X \times X)$ with π and π' as left and right marginal resp., namely $\sum_{y \in X} \mathfrak{w}(x,y) = \pi(x)$ and $\sum_{x \in X} \mathfrak{w}(x,y) = \pi'(y)$ for all $x, y \in X$. Let $\mathfrak{W}(\pi, \pi')$ denote the set of all matchings for π, π'.

Definition 9 (Kantorovich metric, [21])**.** Let $d \colon X \times X \to [0,1]$ be a 1-bounded metric. The *Kantorovich lifting* of d is the 1-bounded metric $\mathbf{K}(d) \colon \Delta(X) \times \Delta(X) \to [0,1]$ defined for all $\pi, \pi' \in \Delta(X)$ by

$$\mathbf{K}(d)(\pi, \pi') = \min_{\mathfrak{w} \in \mathfrak{W}(\pi, \pi')} \sum_{x,y \in X} \mathfrak{w}(x,y) \cdot d(x,y).$$

We remark that since we are considering only probability distributions with finite support, the minimum over $\mathfrak{W}(\pi, \pi')$ is well defined for all $\pi, \pi' \in \Delta(X)$.

The *Hausdorff metric* allows us to lift any distance over probability distributions to a distance over sets of probability distributions.

Definition 10 (Hausdorff metric)**.** Let $\hat{d} \colon \Delta(X) \times \Delta(X) \to [0,1]$ be a 1-bounded metric. The *Hausdorff lifting* of $\hat{d}$ is the 1-bounded metric $\mathbf{H}(\hat{d}) \colon \mathscr{P}(\Delta(X)) \times \mathscr{P}(\Delta(X)) \to [0,1]$ defined by

$$\mathbf{H}(\hat{d})(\Pi_1, \Pi_2) = \max \left\{ \sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} \hat{d}(\pi_1, \pi_2), \sup_{\pi_2 \in \Pi_2} \inf_{\pi_1 \in \Pi_1} \hat{d}(\pi_2, \pi_1) \right\}$$

for all $\Pi_1, \Pi_2 \subseteq \Delta(X)$, where $\inf \emptyset = 1$, $\sup \emptyset = 0$.

Hence, given two processes $s, t \in \mathbf{S}$, the idea is to quantify the distance between each pair of their resolutions by exploiting the Kantorovich metric, which quantifies the disparities in the probabilities of the two processes to execute the same traces. Then, we lift this distance on resolutions to a distance between s and t by means of the Hausdorff metric. Intuitively, as each resolution captures a different set of nondeterministic choices of a process, we use the Hausdorff metric to compare the possible choices of the two processes and to match them in order to obtain the minimal distance.

3.2 Strong trace metric

To define the strong trace metric we start from a distance between traces, defined as the discrete metric over traces: two traces are at distance 1 if they are distinct, otherwise the distance is set to 0. Differently from [28] we do not consider any discount on the distance between traces. Trace equivalences, and thus metrics, are usually employed when the observations on the system cannot be done in a step-by-step fashion, but only the total behavior of the system can be observed. Hence, a step-wise discount does not fit in this setting. However, the discount would not introduce any technical issue.

Definition 11 (Distance between traces)**.** The *distance between traces* $d_T : \mathscr{A}^\star \times \mathscr{A}^\star \to [0,1]$ is defined for any pair of traces $\alpha, \beta \in \mathscr{A}^\star$ by

$$d_T(\alpha,\beta) = \begin{cases} 0 & \text{if } \alpha = \beta \\ 1 & \text{otherwise.} \end{cases}$$

Following [28] we aim to lift the distance d_T to a distance between resolutions by means of the Kantorovich lifting functionalwhich, we recall, is defined on probability distributions. As shown in the following example, we are not guaranteed that the function $\Pr(\mathscr{C}(\_,\_))$ defines a probability distribution on the set of traces of a resolution.

***Example* 2.** Consider process t and the resolution $\mathscr{Z}_r \in \mathrm{Res}(t)$ for it, represented in Fig. 2. We can distinguish three computations for z_t:

$$\begin{aligned} c_1 &= z_t \xrightarrow{a} z_{t_1} \\ c_2 &= z_t \xrightarrow{a} z_{t_2} \\ c_3 &= z_t \xrightarrow{a} z_{t_2} \xrightarrow{d} \text{nil}. \end{aligned}$$

Clearly, $\mathrm{Tr}(\mathscr{C}(z_t)) = \{a, ad\}$. Then we have

$$\begin{aligned} \Pr(\mathscr{C}(z_t, a)) &= \textstyle\sum_{c \in \mathscr{C}(z_t,a)} \Pr(c) = \Pr(c_1) + \Pr(c_2) = 1 \\ \Pr(\mathscr{C}(z_t, ad)) &= \textstyle\sum_{c \in \mathscr{C}(z_t,ad)} \Pr(c) = \Pr(c_3) = 0.5 \end{aligned}$$

from which we gather

$$\sum_{\alpha \in \mathrm{Tr}(\mathscr{C}(z_t))} \Pr(\mathscr{C}(z_t,\alpha)) = \Pr(\mathscr{C}(z_t,a)) + \Pr(\mathscr{C}(z_t,ad)) = 1 + 0.5 > 1.$$

However, as shown in the following lemma, if we consider only maximal computations we obtain a probability distribution over traces.

Lemma 1. *Consider any resolution $\mathscr{Z} \in \mathrm{Res}(\mathbf{S})$ with initial state z. We have that $\sum_{c \in \mathscr{C}_{\max}(z)} \Pr(c) = 1$.*

Proof. We proceed by induction over the depth of z.

The base case $\mathrm{dpt}(z) = 0$ is immediate since we have that $\mathscr{C}(z) = \{\varepsilon\}$ and $\Pr(\varepsilon) = 1$.

Consider now the inductive step $\mathrm{dpt}(z) > 0$. Assume, wlog., that $z \xrightarrow{a}_{\mathscr{Z}} \pi$. Therefore, each trace $c \in \mathscr{C}_{\max}(z)$ will be of the form $c = z \xrightarrow{a} c'$ for some $c' \in \mathscr{C}_{\max}(z')$ for any $z' \in \mathsf{supp}(\pi)$ and moreover for such a trace c it holds that $\Pr(c) = \pi(z')\Pr(c')$. Thus we have

$$\begin{aligned} \textstyle\sum_{c \in \mathscr{C}_{\max}(z)} \Pr(c) &= \textstyle\sum_{\substack{z' \in \mathsf{supp}(\pi) \\ c' \in \mathscr{C}_{\max}(z')}} \pi(z')\Pr(c') \\ &= \textstyle\sum_{z' \in \mathsf{supp}(\pi)} \pi(z') \Big(\sum_{c' \in \mathscr{C}_{\max}(z')} \Pr(c') \Big) \\ &= \textstyle\sum_{z' \in \mathsf{supp}(\pi)} \pi(z') \cdot 1 \qquad \text{(by induction over } \mathrm{dpt}(z') < \mathrm{dpt}(z)) \\ &= 1. \end{aligned}$$

□

Definition 12 (Trace distribution)**.** Consider any resolution $\mathscr{Z} \in \mathrm{Res}(\mathbf{S})$, with initial state z. We define the *trace distribution* of $\mathscr{Z}$ as the function $\mathscr{T}_{\mathscr{Z}} \colon \mathscr{A}^\star \to [0,1]$ defined for each $\alpha \in \mathscr{A}^\star$ by

$$\mathscr{T}_{\mathscr{Z}}(\alpha) = \Pr(\mathscr{C}_{\max}(z,\alpha)).$$

Notice that only maximal computations are in the support of $\mathscr{T}_{\mathscr{Z}}$. This guarantees that $\mathscr{T}_{\mathscr{Z}}$ is a distribution.

Lemma 2. *Consider any resolution $\mathscr{Z} \in \mathrm{Res}(\mathbf{S})$, with initial state z. Then the trace distribution $\mathscr{T}_{\mathscr{Z}}$ of $\mathscr{Z}$ is a probability distribution over $\mathscr{A}^\star$.*

Proof. By definition and by Lemma 1 we have that for each $\alpha \in \mathscr{A}^\star$

$$0 \leq \Pr(\mathscr{C}_{\max}(z,\alpha)) = \sum_{c \in \mathscr{C}_{\max}(z,\alpha)} \Pr(c) \leq \sum_{c \in \mathscr{C}_{\max}(z)} \Pr(c) = 1$$

Hence, we are guaranteed that $\mathscr{T}_{\mathscr{Z}}(\alpha) \in [0,1]$ for each $\alpha \in \mathscr{A}^\star$. Thus, to prove the thesis we simply need to show that $\sum_{\alpha \in \mathscr{A}^\star} \mathscr{T}_{\mathscr{Z}}(\alpha) = 1$. We have that

$$\begin{array}{rcl}
\sum_{\alpha \in \mathscr{A}^\star} \mathscr{T}_{\mathscr{Z}}(\alpha) = & \sum_{\alpha \in \mathscr{A}^\star} \Pr(\mathscr{C}_{\max}(z,\alpha)) \\
= & \sum_{\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))} \Pr(\mathscr{C}_{\max}(z,\alpha)) \\
= & \sum_{\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z)), c \in \mathscr{C}_{\max}(z,\alpha)} \Pr(c) \\
= & \sum_{c \in \bigcup_{\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))} \mathscr{C}_{\max}(z,\alpha)} \Pr(c) \\
= & \sum_{c \in \mathscr{C}_{\max}(z)} \Pr(c) \\
= & 1
\end{array}$$

where

- the second equality follows from the fact that by definition $\Pr(\mathscr{C}_{\max}(z,\alpha)) = 0$ for each $\alpha \notin \mathrm{Tr}(\mathscr{C}_{\max}(z))$;
- the fourth equality follows from the fact that each maximal computation of z belongs to a set $\mathscr{C}_{\max}(z,\alpha)$ for at most one trace α, namely $\bigcup_{\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))} \mathscr{C}_{\max}(z,\alpha)$ is a disjoint union (and therefore no probability weight is counted more than once);
- the fifth equality follows by the fact that the disjoint union $\bigcup_{\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))} \mathscr{C}_{\max}(z,\alpha)$ is a partition of $\mathscr{C}_{\max}(z)$;
- the sixth equality follows by Lemma 1.

□

We remark that function $\mathscr{T}_{\_}$ plays the role of the *trace distribution* introduced in [26]. Formally, in [26] the trace distribution for a resolution is defined as the probability space built over its set of traces. Here, we simply identify it with the probability distribution defined on the probability space. In this setting, two resolutions are said to be *trace distribution equivalent* if they have the same trace distribution and thus two processes are trace equivalent if their resolutions are pairwise equivalent.

Lemma 3. *Consider any resolution $\mathscr{Z} \in \mathrm{Res}(\mathbf{S})$ with initial state z. Consider any trace $\alpha \in \mathscr{A}^\star$. Then $\Pr(\mathscr{C}(z,\alpha)) = \sum_{c \in P_{\max}(z,\alpha)} \Pr(c)$, where $P_{\max}(z,\alpha)$ is the set of maximal computations from z having a prefix which is compatible with α.*

Proof. For simplicity let us distinguish two cases.

1. $\Pr(\mathscr{C}(z,\alpha)) = 0$. This implies that there is no computation from z which is compatible with α. Clearly, this gives that there can not be any maximal computation from z having a prefix compatible with α, namely $P_{\max}(z,\alpha) = \emptyset$. Thus we have $\sum_{c \in P_{\max}(z,\alpha)} \Pr(c) = 0$ from which the thesis follows.
2. $\Pr(\mathscr{C}(z,\alpha)) > 0$. In this case, we proceed by induction over $|\alpha|$.
   - Base case $|\alpha| = 0$, namely $\alpha = \varepsilon$. The only computation compatible with α is the empty computation for which it holds that $\Pr(\mathscr{C}(z,\alpha)) = 1$. Since the empty computation is a prefix for all computations from z we have that $P_{\max}(z,\alpha) = \mathscr{C}_{\max}(z)$. By Lemma 1 we have that $\sum_{c \in \mathscr{C}_{\max}(z)} \Pr(c) = 1$ and thus the thesis follows.
   - Inductive step $|\alpha| > 0$. Assume wlog that the only transition inferable for z in $\mathscr{Z}$ is $z \xrightarrow{a}_{\mathscr{Z}} \pi$. Hence $\alpha = a\alpha'$ for some $\alpha' \in \mathscr{A}^\star$, with $|\alpha'| < |\alpha|$. Then we have

$$\begin{array}{rcll} \Pr(\mathscr{C}(z,\alpha)) = & \sum_{z' \in \mathsf{supp}(\pi)} \pi(z') \Pr(\mathscr{C}(z',\alpha')) & \\ = & \sum_{z' \in \mathsf{supp}(\pi)} \left(\pi(z') \cdot \sum_{c' \in P_{\max}(z',\alpha')} \Pr(c') \right) & \text{(by induction over } |\alpha'|) \\ = & \sum_{z' \in \mathsf{supp}(\pi),\, c' \in P_{\max}(z',\alpha')} \pi(z') \Pr(c') & \\ = & \sum_{c \in P_{\max}(z,a\alpha')} \Pr(c) & \end{array}$$

   where the last equality follows by considering that

$$P_{\max}(z,a\alpha') = \left\{ c \mid c = z \xrightarrow{a}_{\mathscr{Z}} c' \text{ and } c' \in \bigcup_{z' \in \mathsf{supp}(\pi)} P_{\max}(z',\alpha') \right\}.$$

□

Proposition 1. *For any pair of resolutions $\mathscr{Z}_1, \mathscr{Z}_2 \in \mathrm{Res}(\mathbf{S})$, with initial states z_1, z_2 resp., we have that $\mathscr{T}_{\mathscr{Z}_1} = \mathscr{T}_{\mathscr{Z}_2}$ iff $\Pr(\mathscr{C}(z_1,\alpha)) = \Pr(\mathscr{C}(z_2,\alpha))$ for all traces $\alpha \in \mathscr{A}^\star$.*

Proof. The thesis follows by applying the same arguments used it the proof of Theorem 2 below. □

Hence, we can now follow [28] to define the *trace metric*.

Definition 13 (Trace distance between resolutions)**.** The *trace distance between resolutions* $D_T : \mathrm{Res}(\mathbf{S}) \times \mathrm{Res}(\mathbf{S}) \to [0,1]$ is defined for any $\mathscr{Z}_1, \mathscr{Z}_2 \in \mathrm{Res}(\mathbf{S})$ by

$$D_T(\mathscr{Z}_1, \mathscr{Z}_2) = \mathbf{K}(d_T)(\mathscr{T}_{\mathscr{Z}_1}, \mathscr{T}_{\mathscr{Z}_2}).$$

Proposition 2 ([28, Proposition 2])**.** *The kernel of D_T is strong trace distribution equivalence of resolutions.*

To deal with nondeterministic choices, we lift the distance over deterministic resolutions to a pseudometric over processes by means of the Hausdorff lifting functional.

Definition 14 (Strong trace metric)**.** *Strong trace metric* $\mathbf{d}_T : \mathbf{S} \times \mathbf{S} \to [0,1]$ is defined for all $s,t \in \mathbf{S}$ as

$$\mathbf{d}_T(s,t) = \mathbf{H}(D_T)(\mathrm{Res}(s), \mathrm{Res}(t)).$$

Proposition 3 ([28, Proposition 3])**.** *The kernel of $\mathbf{d}_T$ is probabilistic strong trace equivalence.*

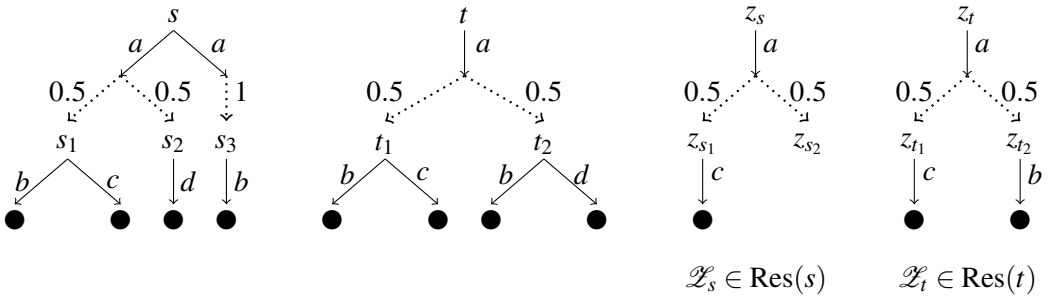


Figure 3: Processes s,t are such that $s \not\approx_{\mathrm{st}} t$ and $\mathbf{d}_T(s,t) = 0.5$.

***Example* 3.** Consider processes s,t in Fig. 3. We have that $s \not\approx_{\mathrm{st}} t$. Notice that none of the resolutions for s can exhibit both traces ab and ac. Thus, whenever we chose resolution $\mathscr{Z}_t \in \mathrm{Res}(t)$ in Fig. 3 for t, then there is no resolution for s that can match $\mathscr{Z}_t$ on all traces.

Let us evaluate the trace distance between s and t. Since resolution $\mathscr{Z}_t$ for t distinguishes the two processes, we start by evaluating its distance from the resolutions for s. Consider the resolution $\mathscr{Z}_s \in \mathrm{Res}(s)$ in Fig. 3. By Def. 12, we have

$$\mathscr{T}_{\mathscr{Z}_s} = 0.5\delta_{ac} + 0.5\delta_a \qquad \mathscr{T}_{\mathscr{Z}_t} = 0.5\delta_{ac} + 0.5\delta_{ab}.$$

Clearly, $d_T(ac,ac) = 0$ and $d_T(ac,a) = d_T(ac,ab) = d_T(a,ab) = 1$. Thus, by Def 13 we have

$$\begin{array}{rcl} D_T(\mathscr{Z}_s,\mathscr{Z}_t) = & \mathbf{K}(d_T)(\mathscr{T}_{\mathscr{Z}_s},\mathscr{T}_{\mathscr{Z}_t}) \\ = & \min_{\mathfrak{w}\in\mathfrak{W}(\mathscr{T}_{\mathscr{Z}_s},\mathscr{T}_{\mathscr{Z}_t})} \sum_{\alpha\in\mathsf{supp}(\mathscr{T}_{\mathscr{Z}_s}),\beta\in\mathsf{supp}(\mathscr{T}_{\mathscr{Z}_t})} \mathfrak{w}(\alpha,\beta)\cdot d_T(\alpha,\beta) \\ = & 0.5\cdot d_T(ac,ac) + 0.5\cdot d_T(a,ab) \\ = & 0.5 \end{array}$$

where to minimize the distance we have matched the two occurrences of the trace ac. By similar calculations, one can easily obtain that

$$0.5 = D_T(\mathscr{Z}_t,\mathscr{Z}_s) = \sup_{\mathscr{Z}_2\in\mathrm{Res}(t)} \inf_{\mathscr{Z}_1\in\mathrm{Res}(s)} D_T(\mathscr{Z}_2,\mathscr{Z}_1).$$

Moreover, it is immediate to check that whichever resolution for s we choose, there is always a resolution for t which is at trace distance 0 from it, namely

$$0 = \sup_{\mathscr{Z}_1\in\mathrm{Res}(s)} \inf_{\mathscr{Z}_2\in\mathrm{Res}(t)} D_T(\mathscr{Z}_1,\mathscr{Z}_2).$$

Therefore, we can conclude that

$$\mathbf{d}_T(s,t) = \mathbf{H}(D_T)(\mathrm{Res}(s),\mathrm{Res}(t)) = \max\{0, 0.5\} = 0.5$$

3.3 Weak trace metric

To obtain the quantitative analogue of the weak trace equivalence, it is enough to adapt the notion of distance between traces (Definition 11) to the weak context. The idea is that since silent steps cannot be observed, then they should not count on the trace distance. Thus we introduce the notion of *weak distance between traces* which is a 1-bounded pseudometric over $\mathscr{A}_\tau^\star$ having $\equiv_{\mathrm{w}}$ as kernel.

Definition 15 (Weak distance between traces)**.** The *weak distance between traces* $d_T^{\mathrm{w}} \colon \mathscr{A}_\tau^\star \times \mathscr{A}_\tau^\star \to [0,1]$ is defined for any pair of traces $\alpha, \beta \in \mathscr{A}_\tau^\star$ by

$$d_T^{\mathrm{w}}(\alpha,\beta) = \begin{cases} 0 & \text{if } \alpha \equiv_{\mathrm{w}} \beta \\ 1 & \text{otherwise.} \end{cases}$$

It is clear that d_T^{w} is a 1-bounded pseudometric whose kernel is the equivalence of traces.

By substituting d_T with d_T^{w} in Definition 13 we obtain the notion of *weak trace distance between resolutions*, denoted by the 1-bounded pseudometric D_T^{w}. By lifting the relation of equivalence of traces $\equiv_{\mathrm{w}}$ to an equivalence on probability distributions over traces $\equiv_{\mathrm{w}}^\dagger$, we obtain that the kernel of D_T^{w} is given by the lifted equivalence on trace distributions, namely by the weak trace distribution equivalence of resolutions. We can prove that our characterization of weak trace equivalence is equivalent to the one proposed in [26] in terms of trace distributions.

To simplify the reasoning in the upcoming proofs, let us define the weak version of the trace distribution given in Definition 12. The idea is that we want to define a probability distribution on the traces executable by a resolution up-to trace equivalence.

Definition 16. Let $s \in \mathbf{S}$ and consider any resolution $\mathscr{Z} \in \mathrm{Res}(\mathbf{S})$, with $z = \mathrm{corr}_{\mathscr{Z}}^{-1}(s)$. We define the *weak trace distribution* for $\mathscr{Z}$ as the function $\mathscr{T}_{\mathscr{Z}}^{\mathrm{w}} \colon \mathscr{A}_\tau^\star \to [0,1]$ defined by $\mathscr{T}_{\mathscr{Z}}^{\mathrm{w}}(\alpha) = \Pr(\mathscr{C}_{\max}^{\mathrm{w}}(z,\alpha))$.

Lemma 4. *For each $\mathscr{Z} \in \mathrm{Res}(\mathbf{S})$, the weak trace distribution $\mathscr{T}_{\mathscr{Z}}^{\mathrm{w}}$ is a probability distribution over $\mathscr{A}^\star$.*

Proof. The thesis follows by applying the same arguments used in the proof of Lemma 2 above. □

Remark 1. Notice that $\mathscr{T}_{\_}^{\mathrm{w}}$ is not a probability distribution over $\mathscr{A}_\tau^\star$. In fact it is enough to consider the simple resolution $\mathscr{Z}$ having z as initial state for which the only transition in $\mathscr{Z}$ is $c = z \xrightarrow{a}_{\mathscr{Z}} \delta_{\mathrm{nil}}$, namely z executes a and then with probability 1 it ends its execution. Clearly we have that $a \equiv_{\mathrm{w}} \tau^n a \tau^m$ for all $n,m \geq 0$. Let $\alpha_{n,m} = \tau^n a \tau^m$. Then by definition of weak trace distribution (Definition 16) we would have that $\mathscr{T}_{\mathscr{Z}}^{\mathrm{w}}(\alpha_{n,m}) = \Pr(\mathscr{C}_{\max}^{\mathrm{w}}(z,\alpha_{n,m})) = \Pr(c) = 1$, for all $n,m \geq 0$. Clearly this would imply that $\sum_{\alpha \in \mathscr{A}_\tau^\star} \mathscr{T}_{\mathscr{Z}}^{\mathrm{w}}(\alpha) = \sum_{n,m \geq 0} \mathscr{T}_{\mathscr{Z}}^{\mathrm{w}}(\alpha_{n,m}) > 1$.

However we remark hat $\mathscr{T}_{\mathscr{Z}}$ is a probability distribution over $\mathscr{A}_\tau^\star$ and thus D_T^{w} is well defined.

We aim to show now that there is a strong relation between the trace distribution for a resolution and its weak version: they are equivalent distributions.

Lemma 5. *For each $\mathscr{Z} \in \mathrm{Res}(\mathbf{S})$ we have that $\mathscr{T}_{\mathscr{Z}} \equiv_{\mathrm{w}}^\dagger \mathscr{T}_{\mathscr{Z}}^{\mathrm{w}}$.*

Proof. The thesis follows by applying the same arguments used in the proof of Lemma 8 below. □

Proposition 4. *For any pair of resolutions $\mathscr{Z}_1, \mathscr{Z}_2 \in \mathrm{Res}(\mathbf{S})$, with initial states z_1 and z_2 resp., we have that $\mathscr{T}_{\mathscr{Z}_1} \equiv_{\mathrm{w}}^\dagger \mathscr{T}_{\mathscr{Z}_2}$ iff $\Pr(\mathscr{C}^{\mathrm{w}}(z_1,\alpha)) = \Pr(\mathscr{C}^{\mathrm{w}}(z_2,\alpha))$ for all $\alpha \in \mathscr{A}^\star$.*

Proof. The thesis follows by the same arguments used in the proof of Theorem 4 below. □

Proposition 5. *The kernel of D_T^{w} is weak trace distribution equivalence of resolutions.*

Proof. The thesis follows by the same arguments used in the proof of Theorem 9 below. □

By substituting D_T with D_T^{w} in Definition 14 we obtain the notion of *weak trace metric*, denoted by the 1-bounded pseudometric $\mathbf{d}_T^{\mathrm{w}}$.

Definition 17 (Weak trace metric)**.** The *weak trace metric* $\mathbf{d}_T^{\mathrm{w}} \colon \mathbf{S} \times \mathbf{S} \to [0,1]$ is defined for all $s,t \in \mathbf{S}$ as

$$\mathbf{d}_T^{\mathrm{w}}(s,t) = \mathbf{H}(D_T^{\mathrm{w}})(\mathrm{Res}(s), \mathrm{Res}(t)).$$

The kernel of the weak trace metric is weak trace equivalence.

Proposition 6. *The kernel of* $\mathbf{d}_T^{\mathrm{w}}$ *is probabilistic weak trace equivalence.*

Proof. ($\Rightarrow$) Assume first that $\mathbf{d}_T^{\mathrm{w}}(s,t) = 0$. We aim to show that $s \approx_{\mathrm{wt}} t$. Since

- by definition $\mathbf{d}_T^{\mathrm{w}}(s,t) = \mathbf{H}(D_T^{\mathrm{w}})(\mathrm{Res}(s), \mathrm{Res}(t))$ and
- the kernel of D_T^{w} is $\equiv_{\mathrm{w}}^{\dagger}$ by Proposition 5

from $\mathbf{d}_T^{\mathrm{w}}(s,t) = 0$ we can infer that $\mathrm{Res}(s) \equiv_{\mathrm{w}}^{\dagger} \mathrm{Res}(t)$. Then, by Proposition 4 we can conclude that $s \approx_{\mathrm{wt}} t$.

($\Leftarrow$) Assume now that $s \approx_{\mathrm{wt}} t$. We aim to show that this implies that $\mathbf{d}_T^{\mathrm{w}}(s,t) = 0$. By Proposition 4 we have that $s \approx_{\mathrm{wt}} t$ implies that $\mathrm{Res}(s) \equiv_{\mathrm{w}}^{\dagger} \mathrm{Res}(t)$. Since the kernel of D_T^{w} is given by $\equiv_{\mathrm{w}}^{\dagger}$ (Proposition 5), we can infer

$$\mathbf{d}_T^{\mathrm{w}}(s,t) = \mathbf{H}(D_T^{\mathrm{w}})(\mathrm{Res}(s), \mathrm{Res}(t)) = 0.$$

□

4 Modal logics for traces

In this section we introduce the modal logics $\mathbb{L}$ and $\mathbb{L}_{\mathrm{w}}$ that will allow us to characterize resp. the strong trace equivalence and its weak version, as well as their quantitative counterparts. The classes $\mathbb{L}$ and $\mathbb{L}_{\mathrm{w}}$ are a simplified version of the modal logic $\mathscr{L}$ [11] which has been successfully employed in [8] to characterize the bisimilarity metric [6, 10, 12].

The logic $\mathbb{L}$ consists of two classes of formulae: the class $\mathbb{L}^{\mathrm{t}}$ of *trace formulae*, which are constituted by (finite) sequences of diamond operators and that will be used to represent traces, and the class $\mathbb{L}^{\mathrm{d}}$ of *trace distribution formulae*, which are defined as probability distributions over trace formulae and that will be used to capture the quantitative properties of resolutions, and thus of processes.

Definition 18 (Modal logic $\mathbb{L}$)**.** The classes of *trace distribution formulae* $\mathbb{L}^{\mathrm{d}}$ and *trace formulae* $\mathbb{L}^{\mathrm{t}}$ over $\mathscr{A}$ are defined by the following BNF-like grammar:

$$\mathbb{L}^{\mathrm{d}} \colon\ \Psi ::= \bigoplus_{i \in I} r_i \Phi_i \qquad\qquad \mathbb{L}^{\mathrm{t}} \colon\ \Phi ::= \top \mid \langle a \rangle \Phi$$

where: (i) Ψ ranges over $\mathbb{L}^{\mathrm{d}}$, (ii) Φ ranges over $\mathbb{L}^{\mathrm{t}}$, (iii) $a \in \mathscr{A}$, (iv) $I \neq \emptyset$ is a finite set of indexes, (v) the formulae Φ_i for $i \in I$ are pairwise distinct, namely $\Phi_i \neq \Phi_j$ for each $i, j \in I$ with $i \neq j$ and (vi) for all $i \in I$ we have $r_i \in (0,1]$ and $\sum_{i \in I} r_i = 1$.

To improve readability, we shall write $r_1\Phi_1 \oplus r_2\Phi$ for $\bigoplus_{i \in I} r_i \Phi_i$ with $I = \{1,2\}$ and Φ for $\bigoplus_{i \in I} r_i \Phi_i$ with $I = \{i\}$, $r_i = 1$ and $\Phi_i = \Phi$.

Definition 19 (Depth)**.** The *depth of trace distribution formulae* in $\mathbb{L}^{\mathrm{d}}$ is defined as $\mathrm{dpt}(\bigoplus_{i \in I} r_i \Phi_i) = \max_{i \in I} \mathrm{dpt}(\Phi_i)$ where the *depth of trace formulae* in $\mathbb{L}^{\mathrm{t}}$ is defined by induction on their structure as (i) $\mathrm{dpt}(\top) = 0$ and (ii) $\mathrm{dpt}(\langle a \rangle \Phi) = 1 + \mathrm{dpt}(\Phi)$.

Definition 20 (Semantics of $\mathbb{L}^{\mathrm{t}}$)**.** The *satisfaction relation* $\models \subseteq \mathscr{C} \times \mathbb{L}^{\mathrm{t}}$ is defined by structural induction over trace formulae in $\mathbb{L}^{\mathrm{t}}$ by

- $c \models \top$ always;
- $c \models \langle a \rangle \Phi$ iff $c = s \xrightarrow{a} c'$ for some computation c' such that $c' \models \Phi$.

We say that a computation c from a process s is *compatible* with the trace formula $\Phi \in \mathbb{L}^{\mathrm{t}}$, notation $c \in \mathscr{C}^{\mathrm{t}}(s,\Phi)$, if $c \models \Phi$ and $|c| = \mathrm{dpt}(\Phi)$.

Definition 21 (Semantics of $\mathbb{L}^{\mathrm{d}}$)**.** The *satisfaction relation* $\models \subseteq \mathbf{S} \times \mathbb{L}^{\mathrm{d}}$ is defined by

- $s \models \bigoplus_{i \in I} r_i \Phi_i$ iff there is a resolution $\mathscr{Z} \in \mathrm{Res}(s)$ with $z = \mathrm{corr}^{-1}_{\mathscr{Z}}(s)$ such that for each $i \in I$ we have $\Pr(\mathscr{C}^{\mathrm{t}}_{\max}(z,\Phi_i)) = r_i$.

We let $\mathbb{L}(s)$ denote the set of formulae satisfied by process $s \in \mathbf{S}$, namely $\mathbb{L}(s) = \{\Psi \in \mathbb{L}^{\mathrm{d}} \mid s \models \Psi\}$.

***Example* 4.** Consider process t in Fig. 3. It is easy to verify that $t \models 0.5\langle a \rangle \langle c \rangle \top \oplus 0.5 \langle a \rangle \langle b \rangle \top$. In fact, if we consider the resolution $\mathscr{Z}_t \in \mathrm{Res}(t)$ in the same figure, we have that the computation $c_1 = z_t \xrightarrow{a} z_{t_1} \xrightarrow{c} \mathrm{nil}$ is compatible with the trace formula $\langle a \rangle \langle c \rangle \top$ and that the computation $c_2 = z_t \xrightarrow{a} z_{t_2} \xrightarrow{b} \mathrm{nil}$ is compatible with the trace formula $\langle a \rangle \langle b \rangle \top$. Moreover, we have $\Pr(\mathscr{C}^{\mathrm{t}}_{\max}(z_t, \langle a \rangle \langle c \rangle \top)) = 0.5$ and $\Pr(\mathscr{C}^{\mathrm{t}}_{\max}(z_t, \langle a \rangle \langle b \rangle \top)) = 0.5$.

The modal logic $\mathbb{L}_{\mathrm{w}}$ differs from $\mathbb{L}$ solely in the labels of the diamonds in $\mathbb{L}^{\mathrm{t}}_{\mathrm{w}}$ which range over $\mathscr{A}_\tau$ in place of $\mathscr{A}$. Hence, syntax and semantics of $\mathbb{L}_{\mathrm{w}}$ directly follow from Definition 18 and Defs. 20-21, resp.

We let $\mathbb{L}_{\mathrm{w}}(s)$ denote the set of formlae satisfied by process $s \in \mathbf{S}$, namely $\mathbb{L}_{\mathrm{w}}(s) = \{\Psi \in \mathbb{L}^{\mathrm{d}}_{\mathrm{w}} \mid s \models \Psi\}$.

We introduce the $\mathbb{L}_{\mathrm{w}}$-*equivalence* which extends the equivalence of traces $\equiv_{\mathrm{w}}$ to trace formulae.

Definition 22 ($\mathbb{L}_{\mathrm{w}}$-equivalence of formulae)**.** The relation of $\mathbb{L}_{\mathrm{w}}$-*equivalence of trace formulae* $\equiv_{\mathrm{w}} \subseteq \mathbb{L}^{\mathrm{t}}_{\mathrm{w}} \times \mathbb{L}^{\mathrm{t}}_{\mathrm{w}}$ is the smallest equivalence relation satisfying (i) $\top \equiv_{\mathrm{w}} \top$ and (ii) $\langle \mathfrak{a}_1 \rangle \Phi_1 \equiv_{\mathrm{w}} \langle \mathfrak{a}_2 \rangle \Phi_2$ iff

- either $\mathfrak{a}_1 = \tau$ and $\Phi_1 \equiv_{\mathrm{w}} \langle \mathfrak{a}_2 \rangle \Phi_2$,
- or $\mathfrak{a}_2 = \tau$ and $\langle \mathfrak{a}_1 \rangle \Phi_1 \equiv_{\mathrm{w}} \Phi_2$
- or $\mathfrak{a}_1 = \mathfrak{a}_2$ and $\Phi_1 \equiv_{\mathrm{w}} \Phi_2$.

Then, the relation of $\mathbb{L}_{\mathrm{w}}$-*equivalence of trace distribution formulae* $\equiv^{\dagger}_{\mathrm{w}} \subseteq \mathbb{L}^{\mathrm{d}}_{\mathrm{w}} \times \mathbb{L}^{\mathrm{d}}_{\mathrm{w}}$ is obtained by lifting $\equiv_{\mathrm{w}}$ to a relation on probability distributions over trace formulae.

Remark 2. Clearly we have $\mathbb{L}_{\mathrm{w}/\equiv_{\mathrm{w}}} = \mathbb{L}$, namely the notion of $\equiv_{\mathrm{w}}$ coincides with the equality of formulae when restricted to $(\mathbb{L}^{\mathrm{d}} \times \mathbb{L}^{\mathrm{d}}) \cup (\mathbb{L}^{\mathrm{t}} \times \mathbb{L}^{\mathrm{t}})$. Given any $\Psi_1, \Psi_2 \in \mathbb{L}^{\mathrm{d}}$, we say that $\Psi_1 = \Psi_2$ if they express the same probability distribution over trace formulae.

Notice that we are using the same symbol $\equiv_{\mathrm{w}}$ to denote both the equivalence of traces and $\mathbb{L}_{\mathrm{w}}$-equivalence. The meaning will always be clear from the context.

5 Logical characterization of relations

In this section we present the characterization of strong (resp. weak) trace equivalence by means of $\mathbb{L}$ (resp. $\mathbb{L}_{\mathrm{w}}$) (Theorem 3 and Theorem 5). Following [8], we introduce the notion of *mimicking formula* of a resolution as a formula expressing the trace distribution for that resolution. Mimicking formulae characterize the (weak) trace distribution equivalence of resolutions: two resolutions are (weak) trace distribution equivalent iff their mimicking formulae are equal (resp. $\mathbb{L}_{\mathrm{w}}$-equivalent) (Theorem 2 and Theorem 4).

The *mimicking formula* of a resolution $\mathscr{Z} \in \mathrm{Res}(\mathbf{S})$ is defined as a trace distribution formula assigning a positive weight only to the maximal traces of $\mathscr{Z}$. Hence, we need to identify each maximal trace of $\mathscr{Z}$ with a proper trace formula. This is achieved through the notion of *tracing formula* of a trace.

Definition 23 (Tracing formula)**.** Given any trace $\alpha \in \mathscr{A}^\star$ we define the *tracing formula* of α, notation $\Phi_\alpha \in \mathbb{L}^{\mathsf{t}}$, inductively on the structure of α as follows:

$$\Phi_\alpha = \begin{cases} \top & \text{if } \alpha = \varepsilon \\ \langle a \rangle \Phi_{\alpha'} & \text{if } \alpha = a\alpha', \alpha' \in \mathscr{A}^\star. \end{cases}$$

Lemma 6. *Let $s \in \mathbf{S}$ and $\alpha \in \mathscr{A}^\star$. For each $c \in \mathscr{C}(s)$ we have $\mathrm{Tr}(c) = \alpha$ iff $c \models \Phi_\alpha$ and $|c| = \mathrm{dpt}(\Phi_\alpha)$.*

Proof. ($\Rightarrow$) Assume first that $\mathrm{Tr}(c) = \alpha$. We aim to show that this implies that $|c| = \mathrm{dpt}(\Phi_\alpha)$ and $c \models \Phi_\alpha$. To this aim we proceed by induction over $|c|$.

- Base case $|c| = 0$, namely c is the empty computation. Since $\alpha = \mathrm{Tr}(c)$, this gives that $\alpha = \varepsilon$ and therefore, by Def. 23, $\Phi_\varepsilon = \top$. Then from Def. 19 we gather $\mathrm{dpt}(\Phi_\alpha) = 0 = |c|$ and by Def. 20 we are guaranteed that $c \models \Phi_\varepsilon$.
- Inductive step $|c| > 0$. Assume wlog that $c = s \xrightarrow{a} c'$. In particular this implies that $|c'| < |c|$. Therefore, from $\alpha = \mathrm{Tr}(c)$ we get that α must be of the form $\alpha = a\alpha'$ for $\alpha' = \mathrm{Tr}(c')$. By Def. 23, $\alpha = a\alpha'$ implies $\Phi_\alpha = \langle a \rangle \Phi_{\alpha'}$. From $\alpha' = \mathrm{Tr}(c')$ and the inductive hypothesis over $|c'|$ we get that $\mathrm{dpt}(\Phi_{\alpha'}) = |c'|$ and $c' \models \Phi_{\alpha'}$. This, taken together with $c = s \xrightarrow{a} c'$ gives $c \models \Phi_\alpha$. Moreover, we have
$$\mathrm{dpt}(\Phi_\alpha) = \mathrm{dpt}(\Phi_{\alpha'}) + 1 = |c'| + 1 = |c|$$
thus concluding the proof.

($\Leftarrow$) Assume now that $|c| = \mathrm{dpt}(\Phi_\alpha)$ and $c \models \Phi_\alpha$. We aim to show that this implies that $\mathrm{Tr}(c) = \alpha$, namely that c is compatible with α. From $c \models \Phi_\alpha$ and the definition of tracing formula (Definition 23) we gather that the sequence of the labels of the first $\mathrm{dpt}(\Phi_\alpha)$ execution steps of c matches α. Moreover, $|c| = \mathrm{dpt}(\Phi_\alpha)$ implies that those steps are actually the only execution steps for c. Therefore we can immediately conclude that $\mathrm{Tr}(c) = \alpha$. □

We remark that a computation c is compatible with Φ_α iff c and α satisfy previous Lemma 6.

Definition 24 (Mimicking formula)**.** Consider any resolution $\mathscr{Z} \in \mathrm{Res}(\mathbf{S})$ with initial state z. We define the *mimicking formula* of $\mathscr{Z}$, notation $\Psi_{\mathscr{Z}}$, as

$$\Psi_{\mathscr{Z}} = \bigoplus_{\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))} \Pr(\mathscr{C}_{\max}(z, \alpha)) \Phi_\alpha$$

where, for each $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))$, the formula Φ_α is the tracing formula of α.

Lemma 7. *For any resolution $\mathscr{Z} \in \mathrm{Res}(\mathbf{S})$, the mimicking formula of $\mathscr{Z}$ is a well defined trace distribution formula.*

Proof. By definition of mimicking formula (Definition 24) we have

$$\Psi_{\mathscr{Z}} = \bigoplus_{\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))} \Pr(\mathscr{C}_{\max}(z, \alpha)) \Phi_\alpha$$

where for each $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))$ the formula Φ_α is the tracing formula of trace α.

Hence, to prove that $\Psi_{\mathscr{Z}}$ is a well defined trace distribution formula we simply need to show that

$$\sum_{\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))} \Pr(\mathscr{C}_{\max}(z, \alpha)) = 1$$

which follows by Lemma 2 by noticing that $\sum_{\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))} \Pr(\mathscr{C}_{\max}(z, \alpha)) = \sum_{\alpha \in \mathscr{A}^\star} \Pr(\mathscr{C}_{\max}(z, \alpha))$. □

***Example* 5.** Consider the resolutions $\mathscr{Z}_s \in \mathrm{Res}(s)$ and $\mathscr{Z}_t \in \mathrm{Res}(t)$ for processes s and t, resp., in Fig. 3. The mimicking formulae for them are, resp.

$$\Psi_{\mathscr{Z}_s} = 0.5\langle a\rangle\langle c\rangle\top \oplus 0.5\langle a\rangle\top$$
$$\Psi_{\mathscr{Z}_t} = 0.5\langle a\rangle\langle c\rangle\top \oplus 0.5\langle a\rangle\langle b\rangle\top.$$

The following results give us a first insight on the characterizing power of mimicking formulae: given $s \in \mathbf{S}$, the set of the mimicking formulae of its resolutions constitutes the set of formulae satisfied by s.

Proposition 7. *Let $s \in \mathbf{S}$. For each $\mathscr{Z} \in \mathrm{Res}(s)$ it holds that $s \models \Psi_{\mathscr{Z}}$.*

Proof. Let $\mathscr{Z} \in \mathrm{Res}(s)$, with $z = \mathrm{corr}^{-1}_{\mathscr{Z}}(s)$. Hence, by definition of mimicking formula (Definition 24) we have that

$$\Psi_{\mathscr{Z}} = \bigoplus_{\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))} \Pr(\mathscr{C}_{\max}(z,\alpha))\Phi_\alpha$$

where, for each $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))$ we have that Φ_α is the tracing formula of α. We need to show that $s \models \Psi_{\mathscr{Z}}$, namely we need to exhibit a resolution $\bar{\mathscr{Z}} \in \mathrm{Res}(s)$, with $\bar{z} = \mathrm{corr}^{-1}_{\bar{\mathscr{Z}}}(s)$, s.t. for each $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))$ we have that $\Pr(\mathscr{C}^{\mathrm{t}}(\bar{z},\Phi_\alpha)) = \Pr(\mathscr{C}_{\max}(z,\alpha))$. We aim to show that $\mathscr{Z}$ is such a resolution, namely that for each $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))$ we have

$$\Pr(\mathscr{C}^{\mathrm{t}}_{\max}(z,\Phi_\alpha)) = \Pr(\mathscr{C}_{\max}(z,\alpha)).$$

Let $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))$. By definition we have

$$\begin{aligned}
\mathscr{C}^{\mathrm{t}}_{\max}(z,\Phi_\alpha) &= \{c \in \mathscr{C}_{\max}(z) \mid c \models \Phi_\alpha \wedge |c| = \mathrm{dpt}(\Phi_\alpha)\} \\
&= \{c \in \mathscr{C}_{\max}(z) \mid \mathrm{Tr}(c) = \alpha\} && \text{(by Lemma 6)} \\
&= \mathscr{C}_{\max}(z,\alpha) && (\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))).
\end{aligned}$$

Thus, we can conclude that

$$\Pr(\mathscr{C}^{\mathrm{t}}_{\max}(z,\Phi_\alpha)) = \sum_{c \in \mathscr{C}^{\mathrm{t}}_{\max}(z,\Phi_\alpha)} \Pr(c) = \sum_{c \in \mathscr{C}_{\max}(z,\alpha)} \Pr(c) = \Pr(\mathscr{C}_{\max}(z,\alpha)).$$

□

Theorem 1. *Let $s \in \mathbf{S}$. We have that $\mathbb{L}(s) = \{1\top\} \cup \{\Psi_{\mathscr{Z}} \mid \mathscr{Z} \in \mathrm{Res}(s)\}$.*

Proof. From Proposition 7 and the definition of the relation $\models$ (Definition 21) we can immediately infer that $\{\Psi_{\mathscr{Z}} \mid \mathscr{Z} \in \mathrm{Res}(s)\} \subseteq \mathbb{L}(s)$. Moreover $1\top \in \mathbb{L}(s)$ is immediate. To conclude the proof we need to show that also the opposite inclusion holds, namely that $\mathbb{L}(s) \setminus \{1\top\} \subseteq \{\Psi_{\mathscr{Z}} \mid \mathscr{Z} \in \mathrm{Res}(s)\}$. To this aim, consider any $\Psi = \bigoplus_{i\in I} r_i\Phi_i$ and assume that $\Psi \in \mathbb{L}(s)$. We have to show that Ψ is the mimicking formula of some resolution for s. Since $s \models \Psi$, from Definition 21 we can infer that there is at least one resolution $\mathscr{Z} \in \mathrm{Res}(s)$ with $z = \mathrm{corr}^{-1}_{\mathscr{Z}}(s)$ s.t. for each $i \in I$ we have $\Pr(\mathscr{C}^{\mathrm{t}}_{\max}(z,\Phi_i)) = r_i$. We aim to prove that among the resolutions ensuring that $s \models \Psi$, there is a particular resolution $\mathscr{Z} \in \mathrm{Res}(s)$ s.t.

$$\Psi = \Psi_{\mathscr{Z}}. \tag{2}$$

First of all we recall that by definition of trace distribution formula (Definition 18), for each $i \in I$ we have $r_i > 0$ and moreover $\sum_{i\in I} r_i = 1$. By definition of $\mathscr{C}^{\mathrm{t}}$, we have that $c \in \mathscr{C}^{\mathrm{t}}_{\max}(z,\Phi_i)$ iff $c \models \Phi_i$ and $|c| =$

$\mathrm{dpt}(\Phi_i)$, which by Lemma 6 implies that $\Phi_i = \Phi_{\mathrm{Tr}(c)}$. Hence, let us consider the resolution $\mathscr{Z} \in \mathrm{Res}(s)$ s.t. for each $i \in I$ we have $\mathscr{C}^{\mathrm{t}}_{\max}(z,\Phi_i) \subseteq \mathscr{C}_{\max}(z)$, namely the resolution s.t. the computations compatible with the trace formulae Φ_i are all maximal. Notice that the existence of such a resolution is guaranteed by $s \models \Psi$. Since for each $c \in \mathscr{C}^{\mathrm{t}}_{\max}(z,\Phi_i)$ we have $c \in \mathscr{C}_{\max}(z)$, we can infer that $\mathrm{Tr}(c) \in \mathrm{Tr}(\mathscr{C}_{\max}(z))$, namely $\Phi_i = \Phi_\alpha$ for some $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))$. This gives that whenever $\Phi_i = \Phi_\alpha$, for some $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))$, then we can prove (as done in the proof of Proposition 7) that

$$\Pr(\mathscr{C}^{\mathrm{t}}_{\max}(z,\Phi_i)) = \Pr(\mathscr{C}_{\max}(z,\alpha)). \tag{3}$$

Furthermore, we have obtained that $\{\Phi_i \mid i \in I\} \subseteq \{\Phi_\alpha \mid \alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))\}$.

To prove Equation (2) we need to show that also the opposite inclusion holds. Assume by contradiction that there is at least one $\beta \in \mathrm{Tr}(\mathscr{C}_{\max}(z))$ s.t. there is no $i \in I$ with $\Phi_i = \Phi_\beta$. Then we would have

$$\begin{array}{lll}
1 = & \sum_{i\in I} r_i & \\
= & \sum_{i\in I} \Pr(\mathscr{C}^{\mathrm{t}}_{\max}(z,\Phi_i)) & \\
\leq & \sum_{\alpha\in \mathrm{Tr}(\mathscr{C}_{\max}(z))\setminus\{\beta\}} \Pr(\mathscr{C}_{\max}(z,\alpha)) & \text{(by Equation (3))} \\
< & \sum_{\alpha\in \mathrm{Tr}(\mathscr{C}_{\max}(z))} \Pr(\mathscr{C}_{\max}(z,\alpha)) & (\beta \in \mathrm{Tr}(\mathscr{C}_{\max}(z)) \text{ implies } \Pr(\mathscr{C}_{\max}(z,\beta)) > 0) \\
= & 1 & \text{(by Lemma 2)}
\end{array}$$

which is a contradiction. Hence we can conclude that $\{\Phi_i \mid i \in I\} = \{\Phi_\alpha \mid \alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))\}$ and thus, due to Equation (3), that Equation (2) holds. □

Remark 3. In Theorem 1, $1\top$ is not included in the set of mimicking formulae of resolutions merely for sake of presentation, as $1\top$ is the mimicking formula of the resolution for s in which no action is executed.

The following theorem states that two resolutions are trace distribution equivalent iff their mimicking formulae are the same.

Theorem 2. *Let $s,t \in \mathbf{S}$ and consider $\mathscr{Z}_s \in \mathrm{Res}(s)$, with $z_s = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(s)$, and $\mathscr{Z}_t \in \mathrm{Res}(t)$, with $z_t = \mathrm{corr}^{-1}_{\mathscr{Z}_t}(t)$. Then $\Psi_{\mathscr{Z}_s} = \Psi_{\mathscr{Z}_t}$ iff for all $\alpha \in \mathscr{A}^\star$ it holds that $\Pr(\mathscr{C}(z_s,\alpha)) = \Pr(\mathscr{C}(z_t,\alpha))$.*

Proof. ($\Rightarrow$) Assume first that $\Psi_{\mathscr{Z}_s} = \Psi_{\mathscr{Z}_t}$. We aim to show that this implies $\Pr(\mathscr{C}(z_s,\alpha)) = \Pr(\mathscr{C}(z_t,\alpha))$ for all $\alpha \in \mathscr{A}^\star$. By definition of mimicking formula (Definition 24) we have

$$\Psi_{\mathscr{Z}_s} = \bigoplus_{\alpha\in\mathrm{Tr}(\mathscr{C}_{\max}(z_s))} \Pr(\mathscr{C}_{\max}(z_s,\alpha))\Phi_\alpha$$

where for each $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z_s))$ the formula Φ_α is the tracing formula of α. Analogously

$$\Psi_{\mathscr{Z}_t} = \bigoplus_{\beta\in\mathrm{Tr}(\mathscr{C}_{\max}(z_t))} \Pr(\mathscr{C}_{\max}(z_t,\beta))\Phi_\beta$$

where for each $\beta \in \mathrm{Tr}(\mathscr{C}_{\max}(z_t))$ the formula Φ_β is the tracing formula of β.

Then from the assumption $\Psi_{\mathscr{Z}_s} = \Psi_{\mathscr{Z}_t}$ we gather

1. $\mathrm{Tr}(\mathscr{C}_{\max}(z_s)) = \mathrm{Tr}(\mathscr{C}_{\max}(z_t))$;
2. from previous item 1 we have that $\Pr(\mathscr{C}_{\max}(z_s,\alpha)) = \Pr(\mathscr{C}_{\max}(z_t,\alpha))$ for each $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z_s))$.

We notice that item 1 above implies the stronger relation

$$\mathrm{Tr}(\mathscr{C}(z_s)) = \mathrm{Tr}(\mathscr{C}(z_t)). \tag{4}$$

In fact each $\alpha \in \mathrm{Tr}(\mathscr{C}(z_s))$ is either a trace in $\mathrm{Tr}(\mathscr{C}_{\max}(z_s))$ or a proper prefix of a trace in that set. In both cases item 1 guarantees that each trace in $\mathrm{Tr}(\mathscr{C}(z_s))$ has a matching trace in $\mathrm{Tr}(\mathscr{C}(z_t))$ and viceversa.

Now, consider any $\alpha \in \mathscr{A}^\star$. We aim to show that $\Pr(\mathscr{C}(z_s,\alpha)) = \Pr(\mathscr{C}(z_t,\alpha))$. For simplicity of presentation, we can distinguish two cases.

- $\Pr(\mathscr{C}(z_s,\alpha)) = 0$. In this case we have that no computation from z_s is compatible with α, namely there is no computation from z_s for which the sequence of the labels of the execution steps matches α. More precisely, we have that $\alpha \notin \mathrm{Tr}(\mathscr{C}(z_s))$. Since by Equation (4) we have that $\mathrm{Tr}(\mathscr{C}(z_s)) = \mathrm{Tr}(\mathscr{C}(z_t))$ we can directly conclude that $\alpha \notin \mathrm{Tr}(\mathscr{C}(z_t))$, namely $\Pr(\mathscr{C}(z_t,\alpha)) = 0$.
- $\Pr(\mathscr{C}(z_s,\alpha)) > 0$. In this case we have that $\alpha \in \mathrm{Tr}(\mathscr{C}(z_s))$ and by Equation (4) we have that this implies that $\alpha \in \mathrm{Tr}(\mathscr{C}(z_t))$. Hence we are guaranteed that $\Pr(\mathscr{C}(z_t,\alpha)) > 0$. It remains to show that $\Pr(\mathscr{C}(z_s,\alpha)) = \Pr(\mathscr{C}(z_t,\alpha))$. We have

$$\begin{array}{rll}
\Pr(\mathscr{C}(z_s,\alpha)) = & \sum_{c\in P_{\max}(z_s,\alpha)} \Pr(c) & \text{(by Lemma 3)} \\
= & \sum_{\beta\in \mathrm{Tr}(P_{\max}(z_s,\alpha))} \Pr(\mathscr{C}_{\max}(z_s,\beta)) & \text{(by def. of } P_{\max}) \\
= & \sum_{\beta\in \mathrm{Tr}(P_{\max}(z_s,\alpha))} \Pr(\mathscr{C}_{\max}(z_t,\beta)) & (P_{\max}(z_s,\alpha) \subseteq \mathscr{C}_{\max}(z_s) \text{ and item 2}) \\
= & \sum_{\beta'\in \mathrm{Tr}(P_{\max}(z_t,\alpha))} \Pr(\mathscr{C}_{\max}(z_t,\beta')) & \text{(by Equation (4))} \\
= & \sum_{c'\in P_{\max}(z_t,\alpha)} \Pr(c') & \text{(by def. of } P_{\max}) \\
= & \Pr(\mathscr{C}(z_t,\alpha)) & \text{(by Lemma 3).}
\end{array}$$

($\Leftarrow$) Assume now that for all $\alpha \in \mathscr{A}^\star$ it holds that $\Pr(\mathscr{C}(z_s,\alpha)) = \Pr(\mathscr{C}(z_t,\alpha))$. We aim to show that this implies that $\Psi_{\mathscr{Z}_s} = \Psi_{\mathscr{Z}_t}$. By definition of mimicking formula (Definition 24) we have

$$\Psi_{\mathscr{Z}_s} = \bigoplus_{\alpha\in\mathrm{Tr}(\mathscr{C}_{\max}(z_s))} \Pr(\mathscr{C}_{\max}(z_s,\alpha))\Phi_\alpha$$
$$\Psi_{\mathscr{Z}_t} = \bigoplus_{\beta\in\mathrm{Tr}(\mathscr{C}_{\max}(z_t))} \Pr(\mathscr{C}_{\max}(z_t,\beta))\Phi_\beta.$$

Therefore, to prove $\Psi_{\mathscr{Z}_s} = \Psi_{\mathscr{Z}_t}$ we need to show that

$$\mathrm{Tr}(\mathscr{C}_{\max}(z_s)) = \mathrm{Tr}(\mathscr{C}_{\max}(z_t)) \tag{5}$$
$$\Pr(\mathscr{C}_{\max}(z_s,\alpha)) = \Pr(\mathscr{C}_{\max}(z_t,\alpha)) \text{ for each } \alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z_s)). \tag{6}$$

First of all we notice that $\Pr(\mathscr{C}(z_s,\alpha)) = \Pr(\mathscr{C}(z_t,\alpha))$ for each $\alpha \in \mathscr{A}^\star$ implies that $\mathrm{Tr}(\mathscr{C}(z_s)) = \mathrm{Tr}(\mathscr{C}(z_t))$. This is due to the fact that by definition, given any $\alpha \in \mathscr{A}^\star$, $\Pr(\mathscr{C}(z_s,\alpha)) > 0$ iff there is at least one computation $c \in \mathscr{C}(z_s)$ s.t. $\alpha = \mathrm{Tr}(c)$. Since $\Pr(\mathscr{C}(z_s,\alpha)) > 0$ implies $\Pr(\mathscr{C}(z_t,\alpha)) > 0$ we can infer that for each $\alpha \in \mathrm{Tr}(\mathscr{C}(z_s))$ there is at least one computation $c' \in \mathrm{Tr}(\mathscr{C}(z_t))$ s.t. $\alpha = \mathrm{Tr}(c')$, namely $\mathrm{Tr}(\mathscr{C}(z_s)) \subseteq \mathrm{Tr}(\mathscr{C}(z_t))$. As the same reasoning can be applied symmetrically to each $\alpha \in \mathrm{Tr}(\mathscr{C}(z_t))$, we can conclude that

$$\mathrm{Tr}(\mathscr{C}(z_s)) = \mathrm{Tr}(\mathscr{C}(z_t)). \tag{7}$$

Next we aim to show that a similar result holds even if we restrict our attention to maximal computations, that is we aim to prove Equation (5).

Let $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z_s))$. Notice that for this α we have $\mathscr{C}_{\max}(z_s,\alpha) \subseteq P_{\max}(z_s,\alpha)$. Then we have

$$\begin{array}{rll} \Pr(\mathscr{C}(z_s,\alpha)) = & \sum_{c\in P_{\max}(z_s,\alpha)} \Pr(c) & \text{(by Lemma 3)} \\ = & \sum_{c\in \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c) + \sum_{c'\in P_{\max}(z_s,\alpha)\setminus \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c'). & \end{array} \tag{8}$$

Moreover, by Lemma 3 it holds that $\Pr(\mathscr{C}(z_t,\alpha)) = \sum_{c''\in P_{\max}(z_t,\alpha)} \Pr(c'')$.

Therefore, from $\Pr(\mathscr{C}(z_s,\alpha)) = \Pr(\mathscr{C}(z_t,\alpha))$ we gather that

$$\sum_{c\in \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c) + \sum_{c'\in P_{\max}(z_s,\alpha)\setminus \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c') = \sum_{c''\in P_{\max}(z_t,\alpha)} \Pr(c''). \tag{9}$$

Assume by contradiction that $P_{\max}(z_t,\alpha) \cap \mathscr{C}_{\max}(z_t,\alpha) = \emptyset$, namely there is no maximal computation from z_t which is compatible with α. Then for each action $a \in \mathscr{A}$ consider the trace αa and define $\mathrm{Add}_{z_s}(\alpha) = \{a \in \mathscr{A} \mid \alpha a \in \mathrm{Tr}(\mathscr{C}(z_s))\}$. From Equation (7) we can directly infer that $\mathrm{Add}_{z_s}(\alpha) = \mathrm{Add}_{z_t}(\alpha)$. Moreover, since we are assuming that no maximal computation from z_t is compatible with α, we get

$$\bigcup_{a\in \mathrm{Add}_{z_s}(\alpha)} P_{\max}(z_s,\alpha a) = P_{\max}(z_s,\alpha) \setminus \mathscr{C}_{\max}(z_s,\alpha) \tag{10}$$

$$\bigcup_{a\in \mathrm{Add}_{z_t}(\alpha)} P_{\max}(z_t,\alpha a) = P_{\max}(z_t,\alpha) \tag{11}$$

where the unions are guaranteed to be disjoint (a single computation cannot be compatible with more than one trace αa). Furthermore, by Lemma 3 we have that for each $a \in \mathrm{Add}_{z_s}(\alpha)$

$$\begin{array}{c} \Pr(\mathscr{C}(z_s,\alpha a)) = \sum_{c_1\in P_{\max}(z_s,\alpha a)} \Pr(c_1) \\ \Pr(\mathscr{C}(z_t,\alpha a)) = \sum_{c_2\in P_{\max}(z_t,\alpha a)} \Pr(c_2) \end{array}$$

from which we get that for each $a \in \mathrm{Add}_{z_s}(\alpha)$ it holds that

$$\sum_{c_1\in P_{\max}(z_s,\alpha a)} \Pr(c_1) = \sum_{c_2\in P_{\max}(z_t,\alpha a)} \Pr(c_2). \tag{12}$$

Therefore we have that

$$\begin{array}{rll} & \sum_{c\in \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c) + \sum_{c'\in P_{\max}(z_s,\alpha)\setminus \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c') & \\ = & \sum_{c\in \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c) + \sum_{c'\in \bigcup_{a\in \mathrm{Add}_{z_s}(\alpha)} P_{\max}(z_s,\alpha a)} \Pr(c') & \text{(by Equation (10))} \\ = & \sum_{c\in \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c) + \sum_{a\in \mathrm{Add}_{z_s}(\alpha)} \left(\sum_{c'\in P_{\max}(z_s,\alpha a)} \Pr(c') \right) & \text{(disjoint union)} \\ = & \sum_{c\in \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c) + \sum_{a\in \mathrm{Add}_{z_s}(\alpha)} \left(\sum_{c''\in P_{\max}(z_t,\alpha a)} \Pr(c'') \right) & \text{(by Equation (12))} \\ = & \sum_{c\in \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c) + \sum_{c''\in \bigcup_{a\in \mathrm{Add}_{z_t}(\alpha)} P_{\max}(z_t,\alpha a)} \Pr(c'') & (\mathrm{Add}_{z_s}(\alpha) = \mathrm{Add}_{z_t}(\alpha) \text{ and disjoint union}) \\ = & \sum_{c\in \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c) + \sum_{c''\in P_{\max}(z_t,\alpha)} \Pr(c'') & \text{(by Equation (11)).} \end{array}$$

Thus we have obtained that

$$\sum_{c\in \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c) + \sum_{c'\in P_{\max}(z_s,\alpha)\setminus \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c') = \sum_{c\in \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c) + \sum_{c''\in P_{\max}(z_t,\alpha)} \Pr(c'')$$

which, since by the choice of α we have that $\sum_{c\in \mathscr{C}_{\max}(z_s,\alpha)} \Pr(c) > 0$, is in contradiction with Equation (9). Therefore, we have obtained that whenever $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z_s))$ then there is at least one maximal computation c from z_t s.t. $\alpha = \mathrm{Tr}(c)$, that is $\mathrm{Tr}(\mathscr{C}_{\max}(z_s)) \subseteq \mathrm{Tr}(\mathscr{C}_{\max}(z_t))$. Since the same reasoning can be applied symmetrically to each $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z_t))$ we gather that also $\mathrm{Tr}(\mathscr{C}_{\max}(z_t)) \subseteq \mathrm{Tr}(\mathscr{C}_{\max}(z_s))$ holds. The two inclusions give us Equation (5).

Finally, we aim to prove Equation (6). Let $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z_s))$. We can distinguish two cases.

- $|\alpha| = \mathrm{dpt}(z_s)$. First of all we notice that from Equation (7) and the assumption $\Pr(\mathscr{C}(z_s,\beta)) = \Pr(\mathscr{C}(z_t,\beta))$ for each $\beta \in \mathscr{A}^\star$, we can infer that $|\alpha| = \mathrm{dpt}(z_t)$. Hence, we have

$$\Pr(\mathscr{C}_{\max}(z_s,\alpha)) = \Pr(\mathscr{C}(z_s,\alpha)) = \Pr(\mathscr{C}(z_t,\alpha)) = \Pr(\mathscr{C}_{\max}(z_t,\alpha)).$$

- $|\alpha| < \mathrm{dpt}(z_s)$. Then we have

$$\begin{array}{ll}
& \Pr(\mathscr{C}_{\max}(z_s,\alpha)) \\
= & \sum_{c\in\mathscr{C}_{\max}(z_s,\alpha)} \Pr(c) \\
= & \sum_{c'\in P_{\max}(z_s,\alpha)} \Pr(c') - \sum_{c''\in P_{\max}(z_s,\alpha)\setminus\mathscr{C}_{\max}(z_s,\alpha)} \Pr(c'') \\
= & \Pr(\mathscr{C}(z_s,\alpha)) - \sum_{c''\in P_{\max}(z_s,\alpha)\setminus\mathscr{C}_{\max}(z_s,\alpha)} \Pr(c'') \\
= & \Pr(\mathscr{C}(z_t,\alpha)) - \sum_{c''\in P_{\max}(z_s,\alpha)\setminus\mathscr{C}_{\max}(z_s,\alpha)} \Pr(c'') \\
= & \sum_{c'''\in P_{\max}(z_t,\alpha)} \Pr(c''') - \sum_{c''\in P_{\max}(z_s,\alpha)\setminus\mathscr{C}_{\max}(z_s,\alpha)} \Pr(c'') \\
= & \sum_{c_1\in\mathscr{C}_{\max}(z_t,\alpha)} \Pr(c_1) + \sum_{c_2\in P_{\max}(z_t,\alpha)\setminus\mathscr{C}_{\max}(z_t,\alpha)} \Pr(c_2) - \sum_{c''\in P_{\max}(z_s,\alpha)\setminus\mathscr{C}_{\max}(z_s,\alpha)} \Pr(c'') \\
= & \sum_{c_1\in\mathscr{C}_{\max}(z_t,\alpha)} \Pr(c_1) + \sum_{c_2\in\bigcup_{b\in\mathrm{Add}_{z_t}(\alpha)} P_{\max}(z_t,\alpha b)} \Pr(c_2) - \sum_{c''\in\bigcup_{b\in\mathrm{Add}_{z_s}(\alpha)} P_{\max}(z_s,\alpha b)} \Pr(c'') \\
= & \sum_{c_1\in\mathscr{C}_{\max}(z_t,\alpha)} \Pr(c_1) + \sum_{b\in\mathrm{Add}_{z_t}(\alpha)} \left(\sum_{c_2\in P_{\max}(z_t,\alpha b)} \Pr(c_2)\right) - \sum_{b\in\mathrm{Add}_{z_s}(\alpha)} \left(\sum_{c''\in P_{\max}(z_s,\alpha b)} \Pr(c'')\right) \\
= & \sum_{c_1\in\mathscr{C}_{\max}(z_t,\alpha)} \Pr(c_1) + \sum_{b\in\mathrm{Add}_{z_t}(\alpha)} \Pr(\mathscr{C}(z_t,\alpha b)) - \sum_{b\in\mathrm{Add}_{z_s}(\alpha)} \Pr(\mathscr{C}(z_s,\alpha b)) \\
= & \sum_{c_1\in\mathscr{C}_{\max}(z_t,\alpha)} \Pr(c_1) \\
= & \Pr(\mathscr{C}_{\max}(z_t,\alpha))
\end{array}$$

  where
  - the second and the sixth steps follow by Equation (8);
  - the third, fifth and ninth steps follow by Lemma 3;
  - the fourth step follows by the initial assumption which guarantees that $\Pr(\mathscr{C}(z_s,\alpha)) = \Pr(\mathscr{C}(z_t,\alpha))$;
  - the seventh step follows by Equation (10);
  - the tenth step follows by $\mathrm{Add}_{z_s}(\alpha) = \mathrm{Add}_{z_t}(\alpha)$ (given by Equation (7)) and the initial assumption which guarantees that for each $b \in \mathrm{Add}_{z_s}(\alpha)$, $\Pr(\mathscr{C}(z_s,\alpha b)) = \Pr(\mathscr{C}(z_t,\alpha b))$.

□

Then we can derive the characterization result for the strong case: two processes s,t are strong trace equivalent iff they satisfy the same formulae in $\mathbb{L}$.

Theorem 3. *For all $s,t \in \mathbf{S}$ we have that $s \approx_{\mathrm{st}} t$ iff $\mathbb{L}(s) = \mathbb{L}(t)$.*

Proof. ($\Rightarrow$) Assume first that $s \approx_{\mathrm{st}} t$. We aim to sow that this implies that $\mathbb{L}(s) = \mathbb{L}(t)$. By Definition 6 $s \approx_{\mathrm{st}} t$ implies that

(i) for each resolution $\mathscr{Z}_s \in \mathrm{Res}(s)$, with $z_s = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(s)$, there is a resolution $\mathscr{Z}_t \in \mathrm{Res}(t)$, with $z_t = \mathrm{corr}^{-1}_{\mathscr{Z}_t}(t)$, s.t. for each $\alpha \in \mathscr{A}^\star$ we have $\Pr(\mathscr{C}(z_s,\alpha)) = \Pr(\mathscr{C}(z_t,\alpha))$;

(ii) for each resolution $\mathscr{Z}_t \in \mathrm{Res}(t)$, with $z_t = \mathrm{corr}^{-1}_{\mathscr{Z}_t}(t)$, there is a resolution $\mathscr{Z}_s \in \mathrm{Res}(s)$, with $z_s = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(s)$, s.t. for each $\alpha \in \mathscr{A}^\star$ we have $\Pr(\mathscr{C}(z_s,\alpha)) = \Pr(\mathscr{C}(z_t,\alpha))$.

Consider any $\mathscr{Z}_s \in \mathrm{Res}(s)$, with $z_s = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(s)$, and let $\mathscr{Z}_t \in \mathrm{Res}(t)$, with $z_t = \mathrm{corr}^{-1}_{\mathscr{Z}_t}(t)$, be any resolution of t satisfying item (i) above. By Theorem 2, $\Pr(\mathscr{C}(z_s,\alpha)) = \Pr(\mathscr{C}(z_t,\alpha))$ for all $\alpha \in \mathscr{A}^\star$ implies that $\Psi_{\mathscr{Z}_s} = \Psi_{\mathscr{Z}_t}$. More precisely, we have that

$$\text{for each } \mathscr{Z}_s \in \mathrm{Res}(s) \text{ there is } \mathscr{Z}_t \in \mathrm{Res}(t) \text{ s.t. } \Psi_{\mathscr{Z}_s} = \Psi_{\mathscr{Z}_t}. \tag{13}$$

Symmetrically, item (ii) above taken together with Theorem 2 gives that

$$\text{for each } \mathscr{Z}_t \in \mathrm{Res}(t) \text{ there is a } \mathscr{Z}_s \in \mathrm{Res}(s) \text{ s.t. } \Psi_{\mathscr{Z}_t} = \Psi_{\mathscr{Z}_s}. \tag{14}$$

Therefore, from Equations (13) and (14) we gather

$$\{\Psi_{\mathscr{Z}_s} \mid \mathscr{Z}_s \in \mathrm{Res}(s)\} = \{\Psi_{\mathscr{Z}_t} \mid \mathscr{Z}_t \in \mathrm{Res}(t)\}. \tag{15}$$

By Theorem 1 we have that $\mathbb{L}(s) = \{1\top\} \cup \{\Psi_{\mathscr{Z}_s} \mid \mathscr{Z}_s \in \mathrm{Res}(s)\}$ and similarly $\mathbb{L}(t) = \{1\top\} \cup \{\Psi_{\mathscr{Z}_t} \mid \mathscr{Z}_t \in \mathrm{Res}(t)\}$. Therefore, from Equation (15) we can conclude that $\mathbb{L}(s) = \mathbb{L}(t)$.

($\Leftarrow$) Assume now that $\mathbb{L}(s) = \mathbb{L}(t)$. We aim to show that this implies that $s \approx_{\mathrm{st}} t$. By Theorem 1 we have that $\mathbb{L}(s) = \{1\top\} \cup \{\Psi_{\mathscr{Z}_s} \mid \mathscr{Z}_s \in \mathrm{Res}(s)\}$ and analogously $\mathbb{L}(t) = \{1\top\} \cup \{\Psi_{\mathscr{Z}_t} \mid \mathscr{Z}_t \in \mathrm{Res}(t)\}$. Hence, from the assumption we can infer that $\{\Psi_{\mathscr{Z}_s} \mid \mathscr{Z}_s \in \mathrm{Res}(s)\} = \{\Psi_{\mathscr{Z}_t} \mid \mathscr{Z}_t \in \mathrm{Res}(t)\}$.

Clearly the equality between the two sets implies that

- for each $\mathscr{Z}_s \in \mathrm{Res}(s)$ there is a $\mathscr{Z}_t \in \mathrm{Res}(t)$ s.t. $\Psi_{\mathscr{Z}_s} = \Psi_{\mathscr{Z}_t}$ and
- for each $\mathscr{Z}_t \in \mathrm{Res}(t)$ there is a $\mathscr{Z}_s \in \mathrm{Res}(s)$ s.t. $\Psi_{\mathscr{Z}_t} = \Psi_{\mathscr{Z}_s}$.

By applying Theorem 2 to the two items above we obtain that

- for each resolution $\mathscr{Z}_s \in \mathrm{Res}(s)$, with $z_s = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(s)$, there is a resolution $\mathscr{Z}_t \in \mathrm{Res}(t)$, with $z_t = \mathrm{corr}^{-1}_{\mathscr{Z}_t}(t)$, s.t. for each $\alpha \in \mathscr{A}^\star$ we have $\Pr(\mathscr{C}(z_s, \alpha)) = \Pr(\mathscr{C}(z_t, \alpha))$;
- for each resolution $\mathscr{Z}_t \in \mathrm{Res}(t)$, with $z_t = \mathrm{corr}^{-1}_{\mathscr{Z}_t}(t)$, there is a resolution $\mathscr{Z}_s \in \mathrm{Res}(s)$, with $z_s = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(s)$, s.t. for each $\alpha \in \mathscr{A}^\star$ we have $\Pr(\mathscr{C}(z_s, \alpha)) = \Pr(\mathscr{C}(z_t, \alpha))$;

from which we can conclude that $s \approx_{\mathrm{st}} t$. □

The notions of *tracing formula* and *mimicking formula* and the related results Lemma 6, Lemma 7, Proposition 7 and Theorem 1 can be easily extended to the weak case by extending the set of traces $\mathscr{A}^\star$ to the set $\mathscr{A}^\star_\tau$.

The following theorem gives the characterization of weak trace distribution equivalence: two resolutions are weak trace distribution equivalent iff their mimicking formulae are $\mathbb{L}_{\mathrm{w}}$-equivalent.

To simplify the upcoming proofs, we introduce an alternative version of the *weak mimicking formula*, which captures the weak trace distribution (see Definition 16) of resolutions.

Definition 25. Consider any resolution $\mathscr{Z} \in \mathrm{Res}(\mathbf{S})$ with initial state z. We define the *weak mimicking formula* of $\mathscr{Z}$ as the trace distribution formula $\Psi^{\mathrm{w}}_{\mathscr{Z}}$ given by

$$\Psi^{\mathrm{w}}_{\mathscr{Z}} = \bigoplus_{\alpha \in \mathrm{Tr_w}(\mathscr{C}_{\max}(z))} \Pr(\mathscr{C}^{\mathrm{w}}_{\max}(z, \alpha))\Phi_\alpha$$

where, for each $\alpha \in \mathrm{Tr_w}(\mathscr{C}_{\max}(z))$, the formula Φ_α is the tracing formula of α.

Notice that from the definitions of $\mathscr{C}^{\mathrm{w}}_{\max}(\_,\_)$ and $\mathrm{Tr_w}(\_)$ we can infer that $\Psi^{\mathrm{w}}_{\_}$ represents a trace distribution formula over the quotient space of $\mathbb{L}_{\mathrm{w}}$ wrt. $\equiv_{\mathrm{w}}$, that is $\Psi^{\mathrm{w}}_{\_} \in \mathbb{L}^{\mathrm{d}}$.

Lemma 8. *For each $\mathscr{Z} \in \mathrm{Res}(\mathbf{S})$ it holds that $\Psi_{\mathscr{Z}} \equiv^{\dagger}_{\mathrm{w}} \Psi^{\mathrm{w}}_{\mathscr{Z}}$.*

Proof. Consider $\mathscr{Z} \in \mathrm{Res}(\mathbf{S})$ with initial state z. First of all we recall that by definition of mimicking formula (Definition 24) we have

$$\Psi_{\mathscr{Z}} = \bigoplus_{\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))} \Pr(\mathscr{C}_{\max}(z, \alpha))\Phi_\alpha$$

where for each $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))$, the formula Φ_α is the tracing formula of α. By definition of weak mimicking formula (Definition 25) we have

$$\Psi^{\mathrm{w}}_{\mathscr{Z}} = \bigoplus_{\beta \in \mathrm{Tr}_{\mathrm{w}}(\mathscr{C}_{\max}(z))} \Pr(\mathscr{C}^{\mathrm{w}}_{\max}(z,\beta))\Phi_\beta$$

where for each $\beta \in \mathrm{Tr}_{\mathrm{w}}(\mathscr{C}_{\max}(z))$, the formula Φ_β is the tracing formula of β. Moreover, we have that for each $\beta \in \mathrm{Tr}_{\mathrm{w}}(\mathscr{C}_{\max}(z))$

$$\begin{array}{rcl} \Pr(\mathscr{C}^{\mathrm{w}}_{\max}(z,\beta)) = & \sum_{c \in \mathscr{C}^{\mathrm{w}}_{\max}(z,\beta)} \Pr(c) \\ = & \sum_{c \in \mathscr{C}_{\max}(z) \text{ s.t. } \mathrm{Tr}(c) \equiv_{\mathrm{w}} \beta} \Pr(c) \\ = & \sum_{\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z)) \text{ s.t. } \alpha \equiv_{\mathrm{w}} \beta} \Pr(\mathscr{C}_{\max}(z,\alpha)). \end{array}$$

Furthermore, by definition of tracing formula (Definition 23) and of $\equiv_{\mathrm{w}}$ (Definition 7), it is immediate that $\alpha \equiv_{\mathrm{w}} \beta$ iff $\Phi_\alpha \equiv_{\mathrm{w}} \Phi_\beta$, for each $\alpha, \beta \in \mathscr{A}^\star$. For simplicity, we denote by α_β each $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))$ s.t. $\alpha \equiv_{\mathrm{w}} \beta$ for some $\beta \in \mathrm{Tr}_{\mathrm{w}}(\mathscr{C}_{\max}(z))$. Notice that by construction of $\mathrm{Tr}_{\mathrm{w}}(\_)$, no trace $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))$ can be equivalent to more than one $\beta \in \mathrm{Tr}_{\mathrm{w}}(\mathscr{C}_{\max}(z))$. Therefore, we have obtained that

$$\begin{array}{rcl} \Psi^{\mathrm{w}}_{\mathscr{Z}} = & \bigoplus_{\beta \in \mathrm{Tr}_{\mathrm{w}}(\mathscr{C}_{\max}(z))} \Pr(\mathscr{C}^{\mathrm{w}}_{\max}(z,\beta))\Phi_\beta \\ \equiv^{\dagger}_{\mathrm{w}} & \bigoplus_{\substack{\beta \in \mathrm{Tr}_{\mathrm{w}}(\mathscr{C}_{\max}(z)) \\ \alpha_\beta \in \mathrm{Tr}(\mathscr{C}_{\max}(z))}} \Pr(\mathscr{C}_{\max}(z,\alpha_\beta))\Phi_{\alpha_\beta} \\ \equiv^{\dagger}_{\mathrm{w}} & \bigoplus_{\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z))} \Pr(\mathscr{C}_{\max}(z,\alpha))\Phi_\alpha \\ = & \Psi_{\mathscr{Z}}. \end{array}$$

□

Theorem 4. *Let $s,t \in \mathbf{S}$ and consider $\mathscr{Z}_s \in \mathrm{Res}(s)$, with $z_s = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(s)$, and $\mathscr{Z}_t \in \mathrm{Res}(t)$, with $z_t = \mathrm{corr}^{-1}_{\mathscr{Z}_t}(t)$. Then $\Psi_{\mathscr{Z}_s} \equiv^{\dagger}_{\mathrm{w}} \Psi_{\mathscr{Z}_t}$ iff for all $\alpha \in \mathscr{A}^\star$ it holds that $\Pr(\mathscr{C}^{\mathrm{w}}(z_s,\alpha)) = \Pr(\mathscr{C}^{\mathrm{w}}(z_t,\alpha))$.*

Proof. ($\Rightarrow$) Assume first that $\Psi_{\mathscr{Z}_s} \equiv^{\dagger}_{\mathrm{w}} \Psi_{\mathscr{Z}_t}$. We aim to show that this implies $\Pr(\mathscr{C}^{\mathrm{w}}(z_s,\alpha)) = \Pr(\mathscr{C}^{\mathrm{w}}(z_t,\alpha))$ for all $\alpha \in \mathscr{A}^\star$. By Lemma 8 we have that

$$\Psi_{\mathscr{Z}_s} \equiv^{\dagger}_{\mathrm{w}} \Psi^{\mathrm{w}}_{\mathscr{Z}_s} \quad \text{and} \quad \Psi_{\mathscr{Z}_t} \equiv^{\dagger}_{\mathrm{w}} \Psi^{\mathrm{w}}_{\mathscr{Z}_t}.$$

Thus, $\Psi_{\mathscr{Z}_s} \equiv^{\dagger}_{\mathrm{w}} \Psi_{\mathscr{Z}_t}$ implies $\Psi^{\mathrm{w}}_{\mathscr{Z}_s} \equiv^{\dagger}_{\mathrm{w}} \Psi^{\mathrm{w}}_{\mathscr{Z}_t}$. Hence the prove the proof obligation, it is enough to prove that

$$\Psi^{\mathrm{w}}_{\mathscr{Z}_s} \equiv^{\dagger}_{\mathrm{w}} \Psi^{\mathrm{w}}_{\mathscr{Z}_t} \text{ implies } \Pr(\mathscr{C}^{\mathrm{w}}(z_s,\alpha)) = \Pr(\mathscr{C}^{\mathrm{w}}(z_t,\alpha)) \text{ for each } \alpha \in \mathscr{A}^\star. \tag{16}$$

From $\Psi^{\mathrm{w}}_{\mathscr{Z}_s} \equiv^{\dagger}_{\mathrm{w}} \Psi^{\mathrm{w}}_{\mathscr{Z}_t}$ we get that

$$\Psi^{\mathrm{w}}_{\mathscr{Z}_t} = \bigoplus_{\substack{\alpha \in \mathrm{Tr}_{\mathrm{w}}(\mathscr{C}_{\max}(z_s)) \\ \beta_\alpha \in \mathrm{Tr}_{\mathrm{w}}(\mathscr{C}_{\max}(z_t)) \cap [\alpha]_{\mathrm{w}}}} \Pr(\mathscr{C}^{\mathrm{w}}_{\max}(z_t,\beta_\alpha))\Phi_{\beta_\alpha}$$

with $\sum_{\beta_\alpha \in \mathrm{Tr}_{\mathrm{w}}(\mathscr{C}_{\max}(z_t)) \cap [\alpha]_{\mathrm{w}}} \Pr(\mathscr{C}^{\mathrm{w}}_{\max}(z_t,\beta_\alpha)) = \Pr(\mathscr{C}^{\mathrm{w}}_{\max}(z_s,\alpha))$ and $\Phi_{\beta_\alpha} \equiv_{\mathrm{w}} \Phi_\alpha$ for each $\beta_\alpha \in \mathrm{Tr}_{\mathrm{w}}(\mathscr{C}_{\max}(z_t)) \cap [\alpha]_{\mathrm{w}}, \alpha \in \mathrm{Tr}_{\mathrm{w}}(\mathscr{C}_{\max}(z_s))$.

We notice that by definition the elements of $\mathrm{Tr}_{\mathrm{w}}(\mathscr{C}_{\max}(z_t))$ represent distinct equivalence classes with respect to $\equiv_{\mathrm{w}}$. Thus we are guaranteed that for each $\alpha \in \mathrm{Tr}_{\mathrm{w}}(\mathscr{C}_{\max}(z_t)) \cap [\alpha]_{\mathrm{w}}$ contains a single trace β_α. Therefore, in this particular case, $\Psi^{\mathrm{w}}_{\mathscr{Z}_s} \equiv^{\dagger}_{\mathrm{w}} \Psi^{\mathrm{w}}_{\mathscr{Z}_t}$ is equivalent to say that $\Psi^{\mathrm{w}}_{\mathscr{Z}_s} = \Psi^{\mathrm{w}}_{\mathscr{Z}_t}$. Moreover, since the representative of the equivalence classes wrt $\equiv_{\mathrm{w}}$ can always be chosen in $\mathscr{A}^\star$, we can always construct

the sets $\mathrm{Tr_w}(\mathscr{C}_{\max}(z_s))$ and $\mathrm{Tr_w}(\mathscr{C}_{\max}(z_t))$ in such a way that $\mathrm{Tr_w}(\mathscr{C}_{\max}(z_s)) \cap \mathscr{A}^\star = \mathrm{Tr_w}(\mathscr{C}_{\max}(z_s))$ and $\mathrm{Tr_w}(\mathscr{C}_{\max}(z_t)) \cap \mathscr{A}^\star = \mathrm{Tr_w}(\mathscr{C}_{\max}(z_t))$. Hence, the same argumentations presented in the first part of the proof of Theorem 2 allow us to prove the proof obligation Equation (16).

($\Leftarrow$) Assume now that for all $\alpha \in \mathscr{A}^\star$ it holds that $\Pr(\mathscr{C}^{\mathrm{w}}(z_s,\alpha)) = \Pr(\mathscr{C}^{\mathrm{w}}(z_t,\alpha))$. We aim to show that this implies that $\Psi_{\mathscr{Z}_s} \equiv^{\dagger}_{\mathrm{w}} \Psi_{\mathscr{Z}_t}$. To this aim we show that the assumption $\Pr(\mathscr{C}^{\mathrm{w}}(z_s,\alpha)) = \Pr(\mathscr{C}^{\mathrm{w}}(z_t,\alpha))$ for all $\alpha \in \mathscr{A}^\star$ implies $\Psi^{\mathrm{w}}_{\mathscr{Z}_s} = \Psi^{\mathrm{w}}_{\mathscr{Z}_t}$. This follows from the same argumentations presented in the second part of the proof of Theorem 2. Then, since from Lemma 8 we have $\Psi_{\mathscr{Z}_s} \equiv^{\dagger}_{\mathrm{w}} \Psi^{\mathrm{w}}_{\mathscr{Z}_s}$ and $\Psi_{\mathscr{Z}_t} \equiv^{\dagger}_{\mathrm{w}} \Psi^{\mathrm{w}}_{\mathscr{Z}_t}$, we can conclude that $\Psi_{\mathscr{Z}_s} \equiv^{\dagger}_{\mathrm{w}} \Psi_{\mathscr{Z}_t}$ as required. □

Then we can derive the characterization result for the weak case: two processes s,t are weak trace equivalent iff they satisfy equivalent formulae in $\mathbb{L}_{\mathrm{w}}$.

Theorem 5. *For all $s,t \in \mathbf{S}$ we have that $s \approx_{\mathrm{wt}} t$ iff $\mathbb{L}_{\mathrm{w}}(s) \equiv^{\dagger}_{\mathrm{w}} \mathbb{L}_{\mathrm{w}}(t)$.*

Proof. ($\Rightarrow$) Assume first that $s \approx_{\mathrm{wt}} t$. We aim to sow that this implies that $\mathbb{L}_{\mathrm{w}}(s) \equiv^{\dagger}_{\mathrm{w}} \mathbb{L}_{\mathrm{w}}(t)$. By Definition 8 $s \approx_{\mathrm{wt}} t$ implies that

(i) for each resolution $\mathscr{Z}_s \in \mathrm{Res}(s)$, with $z_s = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(s)$, there is a resolution $\mathscr{Z}_t \in \mathrm{Res}(t)$, with $z_t = \mathrm{corr}^{-1}_{\mathscr{Z}_t}(t)$, s.t. for each $\alpha \in \mathscr{A}^\star$ we have $\Pr(\mathscr{C}^{\mathrm{w}}(z_s,\alpha)) = \Pr(\mathscr{C}^{\mathrm{w}}(z_t,\alpha))$;

(ii) for each resolution $\mathscr{Z}_t \in \mathrm{Res}(t)$, with $z_t = \mathrm{corr}^{-1}_{\mathscr{Z}_t}(t)$, there is a resolution $\mathscr{Z}_s \in \mathrm{Res}(s)$, with $z_s = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(s)$, s.t. for each $\alpha \in \mathscr{A}^\star$ we have $\Pr(\mathscr{C}^{\mathrm{w}}(z_s,\alpha)) = \Pr(\mathscr{C}^{\mathrm{w}}(z_t,\alpha))$.

Consider any $\mathscr{Z}_s \in \mathrm{Res}(s)$, with $z_s = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(s)$, and let $\mathscr{Z}_t \in \mathrm{Res}(t)$, with $z_t = \mathrm{corr}^{-1}_{\mathscr{Z}_t}(t)$, be any resolution of t satisfying item (i) above. By Theorem 4, $\Pr(\mathscr{C}^{\mathrm{w}}(z_s,\alpha)) = \Pr(\mathscr{C}^{\mathrm{w}}(z_t,\alpha))$ for all $\alpha \in \mathscr{A}^\star$ implies that $\Psi_{\mathscr{Z}_s} \equiv^{\dagger}_{\mathrm{w}} \Psi_{\mathscr{Z}_t}$. More precisely, we have that

$$\text{for each } \mathscr{Z}_s \in \mathrm{Res}(s) \text{ there is } \mathscr{Z}_t \in \mathrm{Res}(t) \text{ s.t. } \Psi_{\mathscr{Z}_s} \equiv^{\dagger}_{\mathrm{w}} \Psi_{\mathscr{Z}_t}. \tag{17}$$

Symmetrically, item (ii) above taken together with Theorem 4 gives that

$$\text{for each } \mathscr{Z}_t \in \mathrm{Res}(t) \text{ there is a } \mathscr{Z}_s \in \mathrm{Res}(s) \text{ s.t. } \Psi_{\mathscr{Z}_t} \equiv^{\dagger}_{\mathrm{w}} \Psi_{\mathscr{Z}_s}. \tag{18}$$

Therefore, from Equations (17) and (18) we gather

$$\{\Psi_{\mathscr{Z}_s} \mid \mathscr{Z}_s \in \mathrm{Res}(s)\} \equiv^{\dagger}_{\mathrm{w}} \{\Psi_{\mathscr{Z}_t} \mid \mathscr{Z}_t \in \mathrm{Res}(t)\}. \tag{19}$$

By Theorem 1 we have that $\mathbb{L}_{\mathrm{w}}(s) = \{1\top\} \cup \{\Psi_{\mathscr{Z}_s} \mid \mathscr{Z}_s \in \mathrm{Res}(s)\}$ and similarly $\mathbb{L}_{\mathrm{w}}(t) = \{1\top\} \cup \{\Psi_{\mathscr{Z}_t} \mid \mathscr{Z}_t \in \mathrm{Res}(t)\}$. Therefore, from Equation (19) we can conclude that $\mathbb{L}_{\mathrm{w}}(s) \equiv^{\dagger}_{\mathrm{w}} \mathbb{L}_{\mathrm{w}}(t)$.

($\Leftarrow$) Assume now that $\mathbb{L}_{\mathrm{w}}(s) \equiv^{\dagger}_{\mathrm{w}} \mathbb{L}_{\mathrm{w}}(t)$. We aim to show that this implies that $s \approx_{\mathrm{wt}} t$. By Theorem 1 we have that $\mathbb{L}_{\mathrm{w}}(s) = \{1\top\} \cup \{\Psi_{\mathscr{Z}_s} \mid \mathscr{Z}_s \in \mathrm{Res}(s)\}$ and analogously $\mathbb{L}_{\mathrm{w}}(t) = \{1\top\} \cup \{\Psi_{\mathscr{Z}_t} \mid \mathscr{Z}_t \in \mathrm{Res}(t)\}$. Hence, from the assumption we can infer that $\{\Psi_{\mathscr{Z}_s} \mid \mathscr{Z}_s \in \mathrm{Res}(s)\} \equiv^{\dagger}_{\mathrm{w}} \{\Psi_{\mathscr{Z}_t} \mid \mathscr{Z}_t \in \mathrm{Res}(t)\}$.

Clearly the equivalence between the two sets implies that

- for each $\mathscr{Z}_s \in \mathrm{Res}(s)$ there is a $\mathscr{Z}_t \in \mathrm{Res}(t)$ s.t. $\Psi_{\mathscr{Z}_s} \equiv^{\dagger}_{\mathrm{w}} \Psi_{\mathscr{Z}_t}$ and
- for each $\mathscr{Z}_t \in \mathrm{Res}(t)$ there is a $\mathscr{Z}_s \in \mathrm{Res}(s)$ s.t. $\Psi_{\mathscr{Z}_t} \equiv^{\dagger}_{\mathrm{w}} \Psi_{\mathscr{Z}_s}$.

By applying Theorem 4 to the two items above we obtain that

- for each resolution $\mathscr{Z}_s \in \mathrm{Res}(s)$, with $z_s = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(s)$, there is a resolution $\mathscr{Z}_t \in \mathrm{Res}(t)$, with $z_t = \mathrm{corr}^{-1}_{\mathscr{Z}_t}(t)$, s.t. for each $\alpha \in \mathscr{A}^\star$ we have $\Pr(\mathscr{C}^{\mathrm{w}}(z_s,\alpha)) = \Pr(\mathscr{C}^{\mathrm{w}}(z_t,\alpha))$;
- for each resolution $\mathscr{Z}_t \in \mathrm{Res}(t)$, with $z_t = \mathrm{corr}^{-1}_{\mathscr{Z}_t}(t)$, there is a resolution $\mathscr{Z}_s \in \mathrm{Res}(s)$, with $z_s = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(s)$, s.t. for each $\alpha \in \mathscr{A}^\star$ we have $\Pr(\mathscr{C}^{\mathrm{w}}(z_s,\alpha)) = \Pr(\mathscr{C}^{\mathrm{w}}(z_t,\alpha))$;

from which we can conclude that $s \approx_{\mathrm{wt}} t$. □

6 Logical characterization of trace metrics

In this section we present the logical characterization of strong and weak trace metric (resp. Theorem 8 and Theorem 11). We define a suitable distance on formulae in $\mathbb{L}$ (resp. $\mathbb{L}_{\mathrm{w}}$) and we characterize the strong (resp. weak) trace metric between processes as the distance between the sets of formulae satisfied by them.

6.1 $\mathbb{L}$-characterization of strong trace metric

Firstly, we need to define a distance on trace formulae.

Definition 26 (Distance on $\mathbb{L}^{\mathrm{t}}$)**.** The function $\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}\colon \mathbb{L}^{\mathrm{t}}\times\mathbb{L}^{\mathrm{t}}\to[0,1]$ is defined over $\mathbb{L}^{\mathrm{t}}$ as follows:

$$\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_1,\Phi_2)=\begin{cases}0 & \text{if } \Phi_1=\Phi_2\\ 1 & \text{otherwise.}\end{cases}$$

Proposition 8. *The function $\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}$ is a 1-bounded metric over $\mathbb{L}^{\mathrm{t}}$.*

Proof. The thesis follows by noticing that $\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}$ is the discrete metric over $\mathbb{L}^{\mathrm{t}}$. □

To define a distance over trace distribution formulae we see them as probability distribution over trace formulae and we define the distance over $\mathbb{L}^{\mathrm{d}}$ as the Kantorovich lifting of the metric $\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}$.

Definition 27 (Distance on $\mathbb{L}^{\mathrm{d}}$)**.** The function $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}\colon \mathbb{L}^{\mathrm{d}}\times\mathbb{L}^{\mathrm{d}}\to[0,1]$ is defined over $\mathbb{L}^{\mathrm{d}}$ as follows:

$$\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\Psi_2)=\mathbf{K}(\mathscr{D}^{\mathrm{t}}_{\mathbb{L}})(\Psi_1,\Psi_2).$$

Proposition 9. *The function $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}$ is a 1-bounded metric over $\mathbb{L}^{\mathrm{d}}$.*

Proof. First we prove that $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}$ is a metric over $\mathbb{L}^{\mathrm{d}}$, namely that

1. $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\Psi_2)=0$ iff $\Psi_1=\Psi_2$;
2. $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\Psi_2)=\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_2,\Psi_1)$;
3. $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\Psi_2)\le\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\Psi_3)+\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_3,\Psi_2)$.

Proof of item 1

($\Leftarrow$) Assume first that $\Psi_1=\Psi_2$. Then $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\Psi_2)=0$ immediately follows from Definition 27, since the Kantorovich metric is a pseudometric.

($\Rightarrow$) Assume now that $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\Psi_2)=0$. We aim to show that this implies that $\Psi_1=\Psi_2$. Assume wlog. that $\Psi_1=\bigoplus_{i\in I}r_i\Phi_i$ and that $\Psi_2=\bigoplus_{j\in J}r_j\Phi_j$. Then we have

$$\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\Psi_2)=\min_{\mathfrak{w}\in\mathfrak{W}(\Psi_1,\Psi_2)}\sum_{i\in I,j\in J}\mathfrak{w}(\Phi_i,\Phi_j)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_i,\Phi_j)\tag{20}$$

and the distance in Equation (20) is 0 if, given the optimal matching $\bar{\mathfrak{w}}$

$$\bar{\mathfrak{w}}(\Phi_i,\Phi_j)>0 \text{ iff } \mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_i,\Phi_j)=0.$$

By Proposition 8 we have that $\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_i,\Phi_j)=0$ iff $\Phi_i=\Phi_j$. In particular, let Φ_{j_i} be any formula in $\{\Phi_j\mid j\in J\}$ s.t. $\Phi_i=\Phi_{j_i}$. Since by Definition 18 the trace formulae Φ_i occurring in Ψ_1 are pairwise distinct and, analogously, the trace formulae Φ_j occurring in Ψ_2 are pairwise distinct, we gather that

$$r_i=\textstyle\sum_{j\in J}\bar{\mathfrak{w}}(\Phi_i,\Phi_j)=\sum_{j_i\in J}\bar{\mathfrak{w}}(\Phi_i,\Phi_{j_i})=\bar{\mathfrak{w}}(\Phi_i,\Phi_{j_i})$$
$$r_j=\textstyle\sum_{i\in I}\bar{\mathfrak{w}}(\Phi_i,\Phi_j)=\sum_{i_j\in I}\bar{\mathfrak{w}}(\Phi_{i_j},\Phi_j)=\bar{\mathfrak{w}}(\Phi_{i_j},\Phi_j).$$

Therefore we can infer that $\Psi_1 = \Psi_2$ as probability distributions over $\mathbb{L}^{\mathrm{t}}$.

Proof of item 2 Immediate from the discrete metric and the matching being both symmetric.

Proof of item 3 Assume wlog. that $\Psi_1 = \bigoplus_{i\in I} r_i\Phi_i$, $\Psi_2 = \bigoplus_{j\in J} r_j\Phi_j$ and $\Psi_3 = \bigoplus_{h\in H} r_h\Phi_h$. Let $\mathfrak{w}_{1,3} \in \mathfrak{W}(\Psi_1,\Psi_3)$ be an optimal matching for Ψ_1,Ψ_3, namely

$$\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\Psi_3) = \min_{\mathfrak{w}\in\mathfrak{W}(\Psi_1,\Psi_3)} \sum_{\substack{i\in I\\ h\in H}} \mathfrak{w}(\Phi_i,\Phi_h)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_i,\Phi_h) = \sum_{\substack{i\in I\\ h\in H}} \mathfrak{w}_{1,3}(\Phi_i,\Phi_h)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_i,\Phi_h)$$

and let $\mathfrak{w}_{2,3} \in \mathfrak{W}(\Psi_2,\Psi_3)$ be an optimal matching for Ψ_2,Ψ_3, that is

$$\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_2,\Psi_3) = \min_{\mathfrak{w}\in\mathfrak{W}(\Psi_2,\Psi_3)} \sum_{\substack{j\in J\\ h\in H}} \mathfrak{w}(\Phi_j,\Phi_h)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_j,\Phi_h) = \sum_{\substack{j\in J\\ h\in H}} \mathfrak{w}_{2,3}(\Phi_j,\Phi_h)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_j,\Phi_h).$$

Consider now the function $f\colon I\times J\times H \to [0,1]$ defined by

$$f(i,j,h) = \mathfrak{w}_{1,3}(\Phi_i,\Phi_h)\cdot\mathfrak{w}_{2,3}(\Phi_j,\Phi_h)\cdot\frac{1}{r_h}.$$

Then, we have $\sum_{j\in J} f(i,j,h) = \mathfrak{w}_{1,3}(\Phi_i,\Phi_h)$ namely the projection of f over the first and third components coincides with the optimal matching for Ψ_1,Ψ_3. Similarly, $\sum_{i\in I} f(i,j,h) = \mathfrak{w}_{2,3}(\Phi_j,\Phi_h)$ namely the projection of f over the second and third components coincides with the optimal matching for Ψ_2,Ψ_3. Moreover, it holds that $\sum_{j\in J,h\in H} f(i,j,h) = r_i$ and $\sum_{i\in I,h\in H} f(i,j,h) = r_j$, that is $f(i,j,h)$ is a matching in $\mathfrak{W}(\Psi_1,\Psi_2)$. Therefore,

$$\begin{array}{rcll}
\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\Psi_2) = & \min_{\mathfrak{w}\in\mathfrak{W}(\Psi_1,\Psi_2)} \sum_{i\in I,\, j\in J} \mathfrak{w}(\Phi_i,\Phi_j)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_i,\Phi_j) & \text{(by definition)}\\
\leq & \sum_{i\in I,\, j\in J,h\in H} f(i,j,h)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_i,\Phi_j) & \text{(by construction of } f)\\
\leq & \sum_{i\in I,\, j\in J,h\in H} f(i,j,h)\big(\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_i,\Phi_h) + \mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_j,\Phi_h)\big) & \text{(since } \mathscr{D}^{\mathrm{t}}_{\mathbb{L}} \text{ is a metric)}\\
= & \sum_{i\in I,\, j\in J,h\in H} f(i,j,h)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_i,\Phi_h) + & \\
 & \sum_{i\in I,\, j\in J,h\in H} f(i,j,h)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_j,\Phi_h) & \\
= & \sum_{i\in I,h\in H}\Big(\sum_{j\in J} f(i,j,h)\Big)\cdot\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_i,\Phi_h) + & \\
 & \sum_{j\in J,h\in H}\Big(\sum_{i\in I} f(i,j,h)\Big)\cdot\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_j,\Phi_h) & \\
= & \sum_{i\in I,h\in H} \mathfrak{w}_{1,3}(\Phi_i,\Phi_h)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_i,\Phi_h) + & \\
 & \sum_{j\in J,h\in H} \mathfrak{w}_{2,3}(\Phi_j,\Phi_h)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_j,\Phi_h) & \text{(by construction of } f)\\
= & \mathbf{K}(\mathscr{D}^{\mathrm{t}}_{\mathbb{L}})(\Psi_1,\Psi_3) + \mathbf{K}(\mathscr{D}^{\mathrm{t}}_{\mathbb{L}})(\Psi_3,\Psi_2) & \text{(by definition of } \mathfrak{w}_{1,3},\mathfrak{w}_{2,3})\\
= & \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\Psi_3) + \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_3,\Psi_2) & \text{(by definition).}
\end{array}$$

To conclude, we need to show that $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}$ is 1-bounded, namely that for each $\Psi_1,\Psi_2 \in \mathbb{L}^{\mathrm{d}}$ we have $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\Psi_2) \leq 1$. Assume wlog that $\Psi_1 = \bigoplus_{i\in I} r_i\Phi_i$ and $\Psi_2 = \bigoplus_{j\in J} r_j \in \Phi_j$. We have

$$\begin{array}{rcll}
\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\Psi_2) = & \min_{\mathfrak{w}\in\mathfrak{W}(\Psi_1,\Psi_2)} \sum_{i\in I,j\in J} \mathfrak{w}(\Phi_i,\Phi_j)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_i,\Phi_j) & \\
\leq & \sum_{i\in I,j\in J} \mathfrak{w}(\Phi_i,\Phi_j)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_i,\Phi_j) & \text{(for an arbitrary } \mathfrak{w})\\
\leq & \sum_{i\in I,j\in J} \mathfrak{w}(\Phi_i,\Phi_j) & (\mathscr{D}^{\mathrm{t}}_{\mathbb{L}} \text{ is either 1 or 0})\\
= & 1 & (\mathfrak{w} \text{ is probability distribution}).
\end{array}$$

□

***Example* 6.** Consider the trace distribution formulae $\Psi_1 = 0.6\langle a\rangle\langle b\rangle\top \oplus 0.4\langle a\rangle\langle c\rangle\top$ and $\Psi_2 = 0.7\langle a\rangle\langle c\rangle\top \oplus 0.3\langle a\rangle\langle b\rangle\top$. We have that

$$\begin{aligned}\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\Psi_2) = \quad & \min_{\mathfrak{w}\in\mathfrak{W}(\Psi_1,\Psi_2)} \sum_{\substack{\Phi\in\mathsf{supp}(\Psi_1)\\ \Phi'\in\mathsf{supp}(\Psi_2)}} \mathfrak{w}(\Phi,\Phi')\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi,\Phi') \\ \leq \quad & 0.3\cdot\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\langle a\rangle\langle b\rangle\top,\langle a\rangle\langle b\rangle\top) + 0.4\cdot\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\langle a\rangle\langle c\rangle\top,\langle a\rangle\langle c\rangle\top) + 0.3\cdot\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\langle a\rangle\langle b\rangle\top,\langle a\rangle\langle c\rangle\top) \\ = \quad & 0.3\cdot 0 + 0.4\cdot 0 + 0.3\cdot 1 \\ = \quad & 0.3\end{aligned}$$

Next result derives from our characterization of trace distribution equivalence of resolutions (Theorem 2).

Theorem 6. *The kernel of $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}$ is trace distribution equivalence of resolutions.*

Proof. Let $s,t\in\mathbf{S}$ and consider $\mathscr{Z}_s\in\mathrm{Res}(s)$, with $z_s = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(s)$, and $\mathscr{Z}_t\in\mathrm{Res}(t)$, with $z_t = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(t)$. By Theorem 2 we have that $z_s \approx_{\mathrm{st}} z_t$ iff $\Psi_{\mathscr{Z}_s} = \Psi_{\mathscr{Z}_t}$. Since by Proposition 9 $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}$ is a metric on $\mathbb{L}^{\mathrm{d}}$, we have that $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_{\mathscr{Z}_s},\Psi_{\mathscr{Z}_t}) = 0$ iff $\Psi_{\mathscr{Z}_s} = \Psi_{\mathscr{Z}_t}$. Thus we can conclude that

$$z_s \approx_{\mathrm{st}} z_t \quad \text{iff} \quad \Psi_{\mathscr{Z}_s} = \Psi_{\mathscr{Z}_t} \quad \text{iff} \quad \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_{\mathscr{Z}_s},\Psi_{\mathscr{Z}_t}) = 0.$$

□

We lift the distance over formulae to a distance over processes as the Hausdorff distance between the sets of formulae satisfied by them.

Definition 28. The $\mathbb{L}$-*distance* over processes $\mathscr{D}_{\mathbb{L}}\colon \mathbf{S}\times\mathbf{S}\to[0,1]$ is defined, for all $s,t\in\mathbf{S}$, by

$$\mathscr{D}_{\mathbb{L}}(s,t) = \mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\mathbb{L}(s),\mathbb{L}(t)).$$

Proposition 10. *The mapping $\mathscr{D}_{\mathbb{L}}$ is a* 1*-bounded pseudometric over* $\mathbf{S}$.

Proof. First we show that $\mathscr{D}_{\mathbb{L}}$ is a pseudometric over $\mathbf{S}$, namely that for each $s,t,u\in\mathbf{S}$

$$\mathscr{D}_{\mathbb{L}}(s,s) = 0 \tag{21}$$
$$\mathscr{D}_{\mathbb{L}}(s,t) = \mathscr{D}_{\mathbb{L}}(t,s) \tag{22}$$
$$\mathscr{D}_{\mathbb{L}}(s,t) \leq \mathscr{D}_{\mathbb{L}}(s,u) + \mathscr{D}_{\mathbb{L}}(u,t) \tag{23}$$

Equation (21) and Equation (22) are immediate from the definition of $\mathscr{D}_{\mathbb{L}}$ (Definition 28).

Let us prove Equation (23). Firstly, we notice that from the definition of Hausdorff distance we have

$$\mathscr{D}_{\mathbb{L}}(s,t) = \max\{\sup_{\Psi\in\mathbb{L}(s)}\inf_{\Psi'\in\mathbb{L}(t)}\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\Psi'),\ \sup_{\Psi'\in\mathbb{L}(t)}\inf_{\Psi\in\mathbb{L}(s)}\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\Psi')\}.$$

Thus, for all $s,t,u\in\mathbf{S}$ we can infer that

$$\sup_{\Psi\in\mathbb{L}(s)}\inf_{\Psi''\in\mathbb{L}(u)}\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\Psi'') \leq \mathscr{D}_{\mathbb{L}}(s,u) \tag{24}$$
$$\sup_{\Psi''\in\mathbb{L}(u)}\inf_{\Psi'\in\mathbb{L}(t)}\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi'',\Psi') \leq \mathscr{D}_{\mathbb{L}}(u,t). \tag{25}$$

As a first step, we aim to show that

$$\sup_{\Psi\in\mathbb{L}(s)}\inf_{\Psi'\in\mathbb{L}(t)}\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\Psi') \leq \mathscr{D}_{\mathbb{L}}(s,u) + \mathscr{D}_{\mathbb{L}}(u,t). \tag{26}$$

For sake of simplicity, we index formulae in $\mathbb{L}(s)$ by indexes in the set J, formulae in $\mathbb{L}(t)$ by indexes in set I and formulae in $\mathbb{L}(u)$ by indexes in H. By definition of infimum we have that for each $\varepsilon_1 > 0$

$$\text{for each } \Psi_j \in \mathbb{L}(s) \text{ there is a } \Psi_{h_j} \in \mathbb{L}(u) \text{ s.t. } \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_{h_j}) < \inf_{\Psi_h \in \mathbb{L}(u)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_h) + \varepsilon_1 \tag{27}$$

and analogously for each $\varepsilon_2 > 0$

$$\text{for each } \Psi_h \in \mathbb{L}(u) \text{ there is a } \Psi_{i_h} \in \mathbb{L}(t) \text{ s.t. } \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_h, \Psi_{i_h}) < \inf_{\Psi_i \in \mathbb{L}(t)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_h, \Psi_i) + \varepsilon_2. \tag{28}$$

In particular given $\Psi_j \in \mathbb{L}(s)$ let $\Psi_{h_j} \in \mathbb{L}(u)$ be the index realizing Equation (27), with respect to ε_1, and let $\Psi_{i_{h_j}} \in \mathbb{L}(t)$ be the index realizing Equation (28) with respect to Ψ_{h_j} and ε_2. Then we have

$$\begin{array}{lll}
& \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_{i_{h_j}}) & \\
\leq & \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_{h_j}) + \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_{h_j}, \Psi_{i_{h_j}}) & (\mathscr{D}^{\mathrm{d}}_{\mathbb{L}} \text{ is a metric}) \\
< & \left(\inf_{\Psi_h \in \mathbb{L}(u)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_h) + \varepsilon_1\right) + \left(\inf_{\Psi_i \in \mathbb{L}(t)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_{h_j}, \Psi_i) + \varepsilon_2\right) & (\text{by Eqs. 27 and 28}) \\
\leq & \left(\sup_{\Psi_j \in \mathbb{L}(s)} \inf_{\Psi_h \in \mathbb{L}(u)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_h) + \varepsilon_1\right) + \left(\sup_{\Psi_h \in \mathbb{L}(u)} \inf_{\Psi_i \in \mathbb{L}(t)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_h, \Psi_i) + \varepsilon_2\right) &
\end{array}$$

from which we gather

$$\inf_{\Psi_i \in \mathbb{L}(t)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_i) \leq \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_{i_{h_j}}) < \sup_{\Psi_j \in \mathbb{L}(s)} \inf_{\Psi_h \in \mathbb{L}(u)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_h) + \sup_{\Psi_h \in \mathbb{L}(u)} \inf_{\Psi_i \in \mathbb{L}(t)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_h, \Psi_i) + \varepsilon_1 + \varepsilon_2.$$

Thus, since j was arbitrary, we obtain

$$\sup_{\Psi_j \in \mathbb{L}(s)} \inf_{\Psi_i \in \mathbb{L}(t)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_i) \leq \sup_{\Psi_j \in \mathbb{L}(s)} \inf_{\Psi_h \in \mathbb{L}(u)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_h) + \sup_{\Psi_h \in \mathbb{L}(u)} \inf_{\Psi_i \in \mathbb{L}(t)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_h, \Psi_i) + \varepsilon_1 + \varepsilon_2$$

and since this relation holds for any ε_1 and ε_2 we can conclude that

$$\sup_{\Psi_j \in \mathbb{L}(s)} \inf_{\Psi_i \in \mathbb{L}(t)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_i) \leq \sup_{\Psi_j \in \mathbb{L}(s)} \inf_{\Psi_h \in \mathbb{L}(u)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_h) + \sup_{\Psi_h \in \mathbb{L}(u)} \inf_{\Psi_i \in \mathbb{L}(t)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_h, \Psi_i).$$

Then, by the inequalities in Equation (24) and Equation (25) we can conclude that

$$\sup_{\Psi_j \in \mathbb{L}(s)} \inf_{\Psi_i \in \mathbb{L}(t)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_i) \leq \mathscr{D}_{\mathbb{L}}(s,u) + \mathscr{D}_{\mathbb{L}}(u,t)$$

and thus Equation (26) holds. Switching the roles of s and t in the steps above allows us to infer

$$\sup_{\Psi_i \in \mathbb{L}(t)} \inf_{\Psi_j \in \mathbb{L}(s)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_i) \leq \mathscr{D}_{\mathbb{L}}(s,u) + \mathscr{D}_{\mathbb{L}}(u,t). \tag{29}$$

Finally, we have

$$\begin{array}{lll}
\mathscr{D}_{\mathbb{L}}(s,t) = & \max\{\sup_{\Psi_j \in \mathbb{L}(s)} \inf_{\Psi_i \in \mathbb{L}(t)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_i), \sup_{\Psi_i \in \mathbb{L}(t)} \inf_{\Psi_j \in \mathbb{L}(s)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_j, \Psi_i)\} & (\text{by definition}) \\
\leq & \mathscr{D}_{\mathbb{L}}(s,u) + \mathscr{D}_{\mathbb{L}}(u,t) &
\end{array}$$

where the last relation follows by Equations (26) and (29).

To conclude, we need to show that $\mathscr{D}_{\mathbb{L}}$ is 1-bounded. We have

$$\begin{array}{lll}
\mathscr{D}_{\mathbb{L}}(s,t) = & \mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\mathbb{L}(s), \mathbb{L}(t)) & \\
= & \max\left\{\sup_{\Psi_i \in \mathbb{L}(s)} \inf_{\Psi_j \in \mathbb{L}(t)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_i, \Psi_j),\ \sup_{\Psi_j \in \mathbb{L}(t)} \inf_{\Psi_i \in \mathbb{L}(s)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_i, \Psi_j)\right\} & \\
\leq & \max\{1, 1\} & (\mathscr{D}^{\mathrm{d}}_{\mathbb{L}} \text{ is 1-bounded}) \\
= & 1. &
\end{array}$$

□

Proposition 11. *Let $s \in \mathbf{S}$. The set $\mathbb{L}(s)$ is a closed subset of $\mathbb{L}$ wrt. the topology induced by $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}$.*

Proof. As we are working on a metric space, the proof obligation is equivalent to prove that each sequence in $\mathbb{L}(s)$ that admits a limit converges in $\mathbb{L}(s)$, namely

$$\text{for each } \{\Psi_n\}_{n\in\mathbb{N}} \subseteq \mathbb{L}(s) \text{ s.t. there is } \Psi \in \mathbb{L} \text{ with } \lim_{n\to\infty} \Psi_n = \Psi \text{ then } \Psi \in \mathbb{L}(s). \tag{30}$$

From Theorem 1 we have that $\mathbb{L}(s) = \{\top\} \cup \{\Psi_{\mathscr{Z}} \mid \mathscr{Z} \in \mathrm{Res}(s)\}$. Since a finite union of closed sets is closed, the proof obligation Equation (30) is equivalent to prove that

$$\{\top\} \text{ is closed} \tag{31}$$
$$\{\Psi_{\mathscr{Z}} \mid \mathscr{Z} \in \mathrm{Res}(s)\} \text{ is closed} \tag{32}$$

Equation (31) is immediate since the only sequence in $\{\top\}$ admitting a limit is the constant sequence $\Psi_n = \top$ for all $n \in \mathbb{N}$.

Let us deal now with Equation (32). First of all, we notice that sequences in $\{\Psi_{\mathscr{Z}} \mid \mathscr{Z} \in \mathrm{Res}(s)\}$ can be written in the general form

$$\Psi_n = \bigoplus_{i\in I_n} r_i^{(n)} \Phi_i^{(n)}$$

with $\{\bigoplus_{i\in I_n} r_i^{(n)} \Phi_i^{(n)}\}_{n\in\mathbb{N}} \subseteq \mathbb{L}(s) \setminus \{\top\}$.

Assume that there is a trace distribution formula $\Psi \in \mathbb{L}^{\mathrm{d}}$ s.t. $\lim_{n\to\infty} \Psi_n = \Psi$. We aim to show that $\Psi \in \mathbb{L}(s)$, namely that

$$\Psi = \Psi_{\mathscr{Z}} \text{ for some } \mathscr{Z} \in \mathrm{Res}(s). \tag{33}$$

In what follows, we assume wlog that limit trace distribution formula Ψ has the form $\Psi = \bigoplus_{j\in J} r_j \Phi_j$.

From $\{\bigoplus_{i\in I_n} r_i^{(n)} \Phi_i^{(n)}\}_{n\in\mathbb{N}} \subseteq \mathbb{L}(s) \setminus \{\top\}$ we gather that for each $n \in \mathbb{N}$ there is a resolution $\mathscr{Z}_n \in \mathrm{Res}(s)$ s.t. $\Psi_{\mathscr{Z}_n} = \bigoplus_{i\in I_n} r_i^{(n)} \Phi_i^{(n)}$. For each $n \in \mathbb{N}$, let $z_n = \mathrm{corr}^{-1}_{\mathscr{Z}_n}(s)$. Then $\Psi_{\mathscr{Z}_n} = \bigoplus_{i\in I_n} r_i^{(n)} \Phi_i^{(n)}$ implies that $I_n = \mathrm{Tr}(\mathscr{C}_{\max}(z_n))$, namely I_n is the set of traces to which the maximal computations of the process z_n are compatible. Hence, for each $i \in I_n$ we have that $\Phi_i^{(n)}$ is the tracing formula of trace i (or to be more formal of the trace indexed by i) and $r_i^{(n)} = \Pr(\mathscr{C}_{\max}(z_n, i))$.

We notice that

$$\begin{array}{ll} & \lim_{n\to\infty} \Psi_n = \Psi \\ \text{iff} & \lim_{n\to\infty} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_n, \Psi) = 0 \\ \text{iff} & \lim_{n\to\infty} \mathbf{K}(\mathscr{D}^{\mathrm{t}}_{\mathbb{L}})(\Psi_n, \Psi) = 0 \end{array}$$

that is iff the sequence $\{\Psi_n\}_{n\in\mathbb{N}}$ converges to Ψ with respect to the Kantorovich metric. Since we are considering distributions with finite support, the convergence with respect to the Kantorovich metric is equivalent to the weak convergence of probability distributions (also called convergence in distribution) which states that $\lim_{n\to\infty} \Psi_n(\Phi) = \Psi(\Phi)$ for each continuity point $\Phi \in \mathbb{L}^{\mathrm{t}}$ of Ψ. Since the probability distribution over trace formuale Ψ is discrete and with finite support, its continuity points are the trace formulae which are not in its support. Hence, we have that $\lim_{n\to\infty} \Psi_n(\Phi) = 0$ for each $\Phi \notin \{\Phi_j \mid j \in J\}$. More specifically, we obtain that $\lim_{n\to\infty} I_n = J$ which gives that if there is an index $\tilde{i}$ s.t. $\lim_{n\to\infty} \Phi_{\tilde{i}}^{(n)} \notin \{\Phi_j \mid j \in J\}$, or if $\{\Phi_{\tilde{i}}^{(n)}\}_{n\in N}$ has no limit, then $\lim_{n\to\infty} r_{\tilde{i}}^{(n)} = 0$. Furthermore, since $\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}$ is the discrete metric over $\mathbb{L}^{\mathrm{t}}$, we have that a sequence of trace formulae $\{\Phi^{(n)}\}_{n\in\mathbb{N}}$ converges to Φ iff the sequence is definitively constant, namely iff there is an $N \in \mathbb{N}$ s.t. $\Phi^{(n)} = \Phi$ for all $n \geq N$. Therefore, from $\lim_{n\to\infty} I_n = J$ we can infer that there is an $N \in \mathbb{N}$ s.t. $I_n = J$ for all $n \geq N$. Consequently, by construction of the sets

I_n, we obtain that $J = \mathrm{Tr}(\mathscr{C}_{\max}(z_N))$ thus giving that, for each $j \in J$, Φ_j is the tracing formula of the trace j (or more formally of the trace indexed by j) and $r_j = \Pr(\mathscr{C}_{\max}(z_N, j))$. Thus, from Definition 24, we infer that the resolution $\mathscr{Z}_N \in \mathrm{Res}(s)$, namely the resolution whose mimicking formula corresponds to the N-th trace distribution formula in the sequence $\{\Psi_n\}_{n\in\mathbb{N}}$, is s.t. $\Psi = \Psi_{\mathscr{Z}_N}$, thus proving Equation (33) and concluding the proof. □

From our $\mathbb{L}$-characterization of strong trace equivalence (Theorem 3) we obtain the following result.

Theorem 7. *The kernel of $\mathscr{D}_{\mathbb{L}}$ is trace equivalence.*

Proof. ($\Rightarrow$) Assume first that $s \approx_{\mathrm{st}} t$. We aim to show that this implies that $\mathscr{D}_{\mathbb{L}}(s,t) = 0$. By Theorem 3 we have that $s \approx_{\mathrm{st}} t$ implies that $\mathbb{L}(s) = \mathbb{L}(t)$ from which we gather

$$\mathscr{D}_{\mathbb{L}}(s,t) = \mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\mathbb{L}(s),\mathbb{L}(t)) = 0.$$

($\Leftarrow$) Assume now that $\mathscr{D}_{\mathbb{L}}(s,t) = 0$. We aim to show that this implies that $s \approx_{\mathrm{st}} t$. Since $\mathbb{L}(s)$ and $\mathbb{L}(t)$ are closed by Proposition 11 and since $\mathscr{D}_{\mathbb{L}}$ is a pseudometric by Proposition 10, from $\mathscr{D}_{\mathbb{L}}(s,t) = 0$ we can infer that $\mathbb{L}(s) = \mathbb{L}(t)$. By Theorem 3 we can conclude that $s \approx_{\mathrm{st}} t$. □

Finally, we obtain the characterization of the strong trace metric.

Theorem 8 (Characterization of strong trace metric)**.** *For all $s,t \in \mathbf{S}$ we have $\mathbf{d}_T(s,t) = \mathscr{D}_{\mathbb{L}}(s,t)$.*

Proof. By definition of trace metric (Definition 14) we have that

$$\mathbf{d}_T(s,t) = \max\left\{ \sup_{\mathscr{Z}_s\in\mathrm{Res}(s)} \inf_{\mathscr{Z}_t\in\mathrm{Res}(t)} D_T(\mathscr{Z}_s,\mathscr{Z}_t),\ \sup_{\mathscr{Z}_t\in\mathrm{Res}(t)} \inf_{\mathscr{Z}_s\in\mathrm{Res}(s)} D_T(\mathscr{Z}_s,\mathscr{Z}_t) \right\}.$$

By definition of $\mathbb{L}$-distance over processes (Definition 28) we have that

$$\begin{aligned}
\mathscr{D}_{\mathbb{L}}(s,t) = \;& \mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\mathbb{L}(s),\mathbb{L}(t)) \\
= \;& \mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\{\top\}\cup\{\Psi_{\mathscr{Z}_s} \mid \mathscr{Z}_s \in \mathrm{Res}(s)\}, \{\top\}\cup\{\Psi_{\mathscr{Z}_t} \mid \mathscr{Z}_t \in \mathrm{Res}(t)\}) \\
= \;& \mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\{\Psi_{\mathscr{Z}_s} \mid \mathscr{Z}_s \in \mathrm{Res}(s)\}, \{\Psi_{\mathscr{Z}_t} \mid \mathscr{Z}_t \in \mathrm{Res}(t)\}) \\
= \;& \max\left\{ \sup_{\mathscr{Z}_s\in\mathrm{Res}(s)} \inf_{\mathscr{Z}_t\in\mathrm{Res}(t)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_{\mathscr{Z}_s},\Psi_{\mathscr{Z}_t}),\ \sup_{\mathscr{Z}_t\in\mathrm{Res}(t)} \inf_{\mathscr{Z}_s\in\mathrm{Res}(s)} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_{\mathscr{Z}_s},\Psi_{\mathscr{Z}_t}) \right\}
\end{aligned}$$

where the third equality follows from the fact that by Definition 27 we have $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\top,\top) = 0$ and $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\top,\Psi) = 1$ for any $\Psi \neq \top$. Thus we have that $\top = \mathrm{argmin}_{\Psi\in\{\top\}\cup\{\Psi_{\mathscr{Z}_t}\mid\mathscr{Z}_t\in\mathrm{Res}(t)\}} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\top,\Psi)$ and symmetrically $\top = \mathrm{argmin}_{\Psi\in\{\top\}\cup\{\Psi_{\mathscr{Z}_s}\mid\mathscr{Z}_s\in\mathrm{Res}(s)\}} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\top)$. Moreover, for any $\Psi \neq \top$ we have that $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\Psi') \leq \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\top)$ for any $\Psi' \in \{\Psi_{\mathscr{Z}_t} \mid \mathscr{Z}_t \in \mathrm{Res}(t)\}$ and $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi'',\Psi) \leq \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\top,\Psi)$ for any $\Psi'' \in \{\Psi_{\mathscr{Z}_s} \mid \mathscr{Z}_s \in \mathrm{Res}(s)\}$.

Hence, to prove the thesis it is enough to show that

$$D_T(\mathscr{Z}_s,\mathscr{Z}_t) = \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_{\mathscr{Z}_s},\Psi_{\mathscr{Z}_t}) \text{ for all } \mathscr{Z}_s \in \mathrm{Res}(s), \mathscr{Z}_t \in \mathrm{Res}(t). \tag{34}$$

Let $\mathscr{Z}_s \in \mathrm{Res}(s)$, with $z_s = \mathrm{corr}^{-1}_{\mathscr{Z}_s}(s)$, and $\mathscr{Z}_t \in \mathrm{Res}(t)$, with $z_t = \mathrm{corr}^{-1}_{\mathscr{Z}_t}(t)$. Then by definition of mimicking formula (Definition 24) we have

$$\Psi_{\mathscr{Z}_s} = \bigoplus_{\alpha\in\mathrm{Tr}(\mathscr{C}_{\max}(z_s))} \Pr(\mathscr{C}_{\max}(z_s,\alpha))\Phi_\alpha$$

where for each $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z_s))$ we have that Φ_α is the tracing formula for the trace α. Similarly,

$$\Psi_{\mathscr{Z}_t} = \bigoplus_{\beta \in \mathrm{Tr}(\mathscr{C}_{\max}(z_t))} \Pr(\mathscr{C}_{\max}(z_t,\beta))\Phi_\beta$$

where for each $\beta \in \mathrm{Tr}(\mathscr{C}_{\max}(z_t))$ we have that Φ_β is the tracing formula for the trace β.

By definition of trace distance between resolutions (Definition 13) we have that

$$D_T(\mathscr{Z}_s,\mathscr{Z}_t) = \min_{\mathfrak{w}\in\mathfrak{W}(\mathscr{T}_{\mathscr{Z}_s},\mathscr{T}_{\mathscr{Z}_t})} \sum_{\alpha\in\mathrm{Tr}(\mathscr{C}_{\max}(z_s)),\beta\in\mathrm{Tr}(\mathscr{C}_{\max}(z_t))} \mathfrak{w}(\alpha,\beta)d_T(\alpha,\beta) \tag{35}$$

where, by definition of trace distance between traces (Definition 11), we have that $d_t(\alpha,\beta) = 0$ if $\alpha = \beta$ and $d_t(\alpha,\beta) = 1$ otherwise.

Hence, by definition of tracing formula (Definition 23), it is immediate that for all $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z_s)), \beta \in \mathrm{Tr}(\mathscr{C}_{\max}(z_t))$ we have $d_T(\alpha,\beta) = \mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_\alpha,\Phi_\beta)$, thus giving

$$(35) = \min_{\mathfrak{w}\in\mathfrak{W}(\mathscr{T}_{\mathscr{Z}_s},\mathscr{T}_{\mathscr{Z}_t})} \sum_{\alpha\in\mathrm{Tr}(\mathscr{C}_{\max}(z_s)),\beta\in\mathrm{Tr}(\mathscr{C}_{\max}(z_t))} \mathfrak{w}(\alpha,\beta)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_\alpha,\Phi_\beta). \tag{36}$$

Let $\bar{\mathfrak{w}}$ be an optimal matching for $D_T(\mathscr{Z}_s,\mathscr{Z}_t)$, namely

$$(36) = \sum_{\alpha\in\mathrm{Tr}(\mathscr{C}_{\max}(z_s)),\beta\in\mathrm{Tr}(\mathscr{C}_{\max}(z_t))} \bar{\mathfrak{w}}(\alpha,\beta)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_\alpha,\Phi_\beta). \tag{37}$$

Then, by definition of matching and of $\mathscr{T}_{\_}$ (Definition 12) we have that for any $\alpha \in \mathrm{Tr}(\mathscr{C}_{\max}(z_s)), \beta \in \mathrm{Tr}(\mathscr{C}_{\max}(z_t))$

$$\Pr(\mathscr{C}_{\max}(z_s,\alpha)) = \mathscr{T}_{\mathscr{Z}_s}(\alpha) = \textstyle\sum_{\beta\in\mathrm{Tr}(\mathscr{C}_{\max}(z_t))} \bar{\mathfrak{w}}(\alpha,\beta)$$
$$\Pr(\mathscr{C}_{\max}(z_t,\beta)) = \mathscr{T}_{\mathscr{Z}_t}(\beta) = \textstyle\sum_{\alpha\in\mathrm{Tr}(\mathscr{C}_{\max}(z_s))} \bar{\mathfrak{w}}(\alpha,\beta).$$

Therefore we have obtained that $\bar{\mathfrak{w}}$ is a matching for $\Psi_{\mathscr{Z}_s}$ and $\Psi_{\mathscr{Z}_t}$. In particular we notice that $\bar{\mathfrak{w}}$ is actually an optimal matching for $\Psi_{\mathscr{Z}_s},\Psi_{\mathscr{Z}_t}$. This follows from the optimality of $\bar{\mathfrak{w}}$ for $\mathscr{T}_{\mathscr{Z}_s},\mathscr{T}_{\mathscr{Z}_t}$. In fact each matching for $\Psi_{\mathscr{Z}_s},\Psi_{\mathscr{Z}_t}$ can be constructed from a matching for $\mathscr{T}_{\mathscr{Z}_s},\mathscr{T}_{\mathscr{Z}_t}$ using the same technique proposed above. Moreover, given $\mathfrak{w}_1 \in \mathfrak{W}(\mathscr{T}_{\mathscr{Z}_s},\mathscr{T}_{\mathscr{Z}_t})$ and $\mathfrak{w}_2$ being the matching for Ψ_1,Ψ_2 built from it, the reasoning above guarantees that

$$\sum_{\alpha\in\mathrm{Tr}(\mathscr{C}_{\max}(z_s)),\beta\in\mathrm{Tr}(\mathscr{C}_{\max}(z_t))} \mathfrak{w}_1(\alpha,\beta)d_T(\alpha,\beta) = \sum_{\substack{\alpha\in\mathrm{Tr}(\mathscr{C}_{\max}(z_s))\\ \beta\in\mathrm{Tr}(\mathscr{C}_{\max}(z_t))}} \mathfrak{w}_2(\alpha,\beta)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_\alpha,\Phi_\beta).$$

$\bar{\mathfrak{w}}$ being optimal for D_T implies $\tilde{\mathfrak{w}}$ being optimal for $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}$. Hence by Definition 27 we have

$$\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_{\mathscr{Z}_s},\Psi_{\mathscr{Z}_t}) = \sum_{\alpha\in\mathrm{Tr}(\mathscr{C}_{\max}(z_s)),\beta\in\mathrm{Tr}(\mathscr{C}_{\max}(z_t))} \bar{\mathfrak{w}}(\alpha,\beta)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}}(\Phi_\alpha,\Phi_\beta).$$

From Equation (37) we infer $D_T(\mathscr{Z}_s,\mathscr{Z}_t) = \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_{\mathscr{Z}_s},\Psi_{\mathscr{Z}_t})$ thus proving Equation (34) and concluding the proof. □

6.2 $\mathbb{L}_{\mathrm{w}}$-characterization of weak trace metric

The idea behind the definition of a metric on $\mathbb{L}_{\mathrm{w}}$ is pretty much the same to the strong case. The main difference is that the distance on $\mathbb{L}_{\mathrm{w}}$ is a pseudometric whose kernel is given by $\mathbb{L}_{\mathrm{w}}$-equivalence.

Definition 29 (Distance on $\mathbb{L}^{\mathrm{t}}_{\mathrm{w}}$)**.** The function $\mathscr{D}^{\mathrm{t}}_{\mathbb{L}_{\mathrm{w}}} : \mathbb{L}^{\mathrm{t}}_{\mathrm{w}} \times \mathbb{L}^{\mathrm{t}}_{\mathrm{w}} \to [0,1]$ is defined over $\mathbb{L}^{\mathrm{t}}_{\mathrm{w}}$ as follows:

$$\mathscr{D}^{\mathrm{t}}_{\mathbb{L}_{\mathrm{w}}}(\Phi_1,\Phi_2) = \begin{cases} 0 & \text{if } \Phi_1 \equiv_{\mathrm{w}} \Phi_2 \\ 1 & \text{otherwise.} \end{cases}$$

Clearly, $\mathscr{D}^{\mathrm{t}}_{\mathbb{L}_{\mathrm{w}}}$ is a pseudometric on $\mathbb{L}^{\mathrm{t}}_{\mathrm{w}}$ whose kernel is given by equivalence of trace formulae and we can lift it to a pseudometric over $\mathbb{L}^{\mathrm{d}}_{\mathrm{w}}$ via the Kantorovich lifting functional.

Definition 30 (Distance on $\mathbb{L}^{\mathrm{d}}_{\mathrm{w}}$)**.** The function $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}} : \mathbb{L}^{\mathrm{d}}_{\mathrm{w}} \times \mathbb{L}^{\mathrm{d}}_{\mathrm{w}} \to [0,1]$ is defined over $\mathbb{L}^{\mathrm{d}}_{\mathrm{w}}$ as follows:

$$\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}(\Psi_1,\Psi_2) = \mathbf{K}(\mathscr{D}^{\mathrm{t}}_{\mathbb{L}_{\mathrm{w}}})(\Psi_1,\Psi_2).$$

Proposition 12. *The function $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}$ is a 1-bounded pseudometric over $\mathbb{L}^{\mathrm{d}}_{\mathrm{w}}$.*

Proof. The same arguments used in the proof of Proposition 9 apply, where in place of item 1 we simply need to show that $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}(\Psi,\Psi) = 0$, which is immediate from the definition through the Kantorovich pseudometric. □

Theorem 9. *The kernel of $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}$ is $\mathbb{L}_{\mathrm{w}}$-equivalence of trace distribution formulae.*

Proof. ($\Rightarrow$) Assume first that $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}(\Psi_1,\Psi_2) = 0$ for $\Psi_1 = \bigoplus_{i\in I} r_i\Phi_i$ and $\Psi_2 = \bigoplus_{j\in J} r_j\Phi_j$. We aim to show that this implies $\Psi_1 \equiv^{\dagger}_{\mathrm{w}} \Psi_2$. From the assumption, we have

$$\begin{array}{rll} 0 = & \mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}(\bigoplus_{i\in I} r_i\Phi_i, \bigoplus_{j\in J} r_j\Phi_j) & \\ = & \min_{\mathfrak{w}\in\mathfrak{W}(\Psi_1,\Psi_2)} \sum_{i\in I,\, j\in J} \mathfrak{w}(\Phi_i,\Phi_j)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}_{\mathrm{w}}}(\Phi_i,\Phi_j) & \\ = & \sum_{i\in I,\, j\in J} \mathfrak{w}(\Phi_i,\Phi_j)\mathscr{D}^{\mathrm{t}}_{\mathbb{L}_{\mathrm{w}}}(\Phi_i,\Phi_j) & \text{(for } \mathfrak{w} \text{ optimal matching).} \end{array}$$

Thus, for each $i \in I$ and $j \in J$ we can distinguish two cases:

- either $\mathfrak{w}(\Phi_i,\Phi_j) = 0$,
- or $\mathfrak{w}(\Phi_i,\Phi_j) > 0$, implying $\mathscr{D}^{\mathrm{t}}_{\mathbb{L}_{\mathrm{w}}}(\Phi_i,\Phi_j) = 0$, which is equivalent to say that $\Phi_i \equiv_{\mathrm{w}} \Phi_j$ by Definition 29.

For each $i \in I$, let $J_i \subseteq J$ be the set of indexes j_i for which $\mathfrak{w}(\Phi_i,\Phi_{j_i}) > 0$ and, symmetrically, for each $j \in J$ let $I_j \subseteq I$ be the set of indexes i_j for which $\mathfrak{w}(\Phi_{i_j},\Phi_j) > 0$. So we have

$$\begin{array}{rll} \Psi_1 = & \bigoplus_{i\in I} r_i\Phi_i & \\ = & \bigoplus_{i\in I} \big(\sum_{j\in J} \mathfrak{w}(\Phi_i,\Phi_j)\big)\Phi_i & (\mathfrak{w} \in \mathfrak{W}(\Psi_1,\Psi_2)) \\ \equiv^{\dagger}_{\mathrm{w}} & \bigoplus_{i\in I} \big(\sum_{j_i\in J_i} \mathfrak{w}(\Phi_i,\Phi_{j_j})\big)\Phi_i & \text{(by construction of each } J_i) \\ \equiv^{\dagger}_{\mathrm{w}} & \bigoplus_{i\in I,\, j_i\in J_i} \mathfrak{w}(\Phi_i,\Phi_{j_i})\Phi_{j_i} & (\Phi_i \equiv_{\mathrm{w}} \Phi_{j_i} \text{ for each } j_i \in J_i) \\ \equiv^{\dagger}_{\mathrm{w}} & \bigoplus_{i\in I,\, j_i\in J_i,\, i'_{j_i}\in I_{j_i}} \mathfrak{w}(\Phi_{i'_{j_i}},\Phi_{j_i})\Phi_{i'_{j_i}} & (\Phi_{i'_{j_i}} \equiv_{\mathrm{w}} \Phi_{j_i} \text{ for each } i'_{j_i} \in I_{j_i}) \\ \equiv^{\dagger}_{\mathrm{w}} & \bigoplus_{i_j\in I_j,\, j\in J} \mathfrak{w}(\Phi_{i_j},\Phi_j)\Phi_{i_j} & \text{(all indexes } j \in J \text{ are involved)} \\ \equiv^{\dagger}_{\mathrm{w}} & \bigoplus_{j\in J} \big(\sum_{i_j\in I_j} \mathfrak{w}(\Phi_{i_j},\Phi_j)\big)\Phi_j & (\Phi_j \equiv_{\mathrm{w}} \Phi_{i_j} \text{ for each } i_j \in I_j) \\ \equiv^{\dagger}_{\mathrm{w}} & \bigoplus_{j\in J} \big(\sum_{i\in I} \mathfrak{w}(\Phi_i,\Phi_j)\big)\Phi_j & \text{(by construction of each } I_j) \\ = & \bigoplus_{j\in J} r_j\Phi_j & (\mathfrak{w} \in \mathfrak{W}(\Psi_1,\Psi_2)) \\ = & \Psi_2. & \end{array}$$

($\Leftarrow$). Assume that $\Psi_1 \equiv_{\mathrm{w}}^{\dagger} \Psi_2$. We aim to show that $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}(\Psi_1,\Psi_2)=0$. Assume wlog. that $\Psi_1 = \bigoplus_{i\in I} r_i\Phi_i$. By definition of $\equiv_{\mathrm{w}}$ (Definition 7) and definition of lifting of a relation (Definition 2), from $\Psi_2 \equiv_{\mathrm{w}}^{\dagger} \bigoplus_{i\in I} r_i\Phi_i$ we gather $\Psi_2 = \bigoplus_{\substack{i\in I\\ j_i\in J_i}} r_{j_i}\Phi_{j_i}$ with $\sum_{j_i\in J_i} r_{j_i} = r_i$ and $\Phi_{j_i} \equiv_{\mathrm{w}} \Phi_i$ for all $j_i \in J_i, i \in I$. Then

$$\begin{array}{rcll}
\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}(\Psi_1,\Psi_2) = & \mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}(\bigoplus_{i\in I} r_i\Phi_i, \bigoplus_{i\in I,\, j_i\in J_i} r_{j_i}\Phi_{j_i}) & \\
= & \min_{\mathfrak{w}\in\mathfrak{W}(\Psi_1,\Psi_2)} \sum_{\substack{i\in I,\, j_h\in J_h\\ h\in I}} \mathfrak{w}(\Phi_i,\Phi_{j_h})\mathscr{D}^{\mathrm{t}}_{\mathbb{L}_{\mathrm{w}}}(\Phi_i,\Phi_{j_h}) & \\
\leq & \sum_{\substack{i\in I,\, j_h\in J_h\\ h\in I}} \tilde{\mathfrak{w}}(\Phi_i,\Phi_{j_h})\mathscr{D}^{\mathrm{t}}_{\mathbb{L}_{\mathrm{w}}}(\Phi_i,\Phi_{j_h}) & \\
= & \sum_{i\in I,\, j_i\in J_i} r_{j_i}\mathscr{D}^{\mathrm{t}}_{\mathbb{L}_{\mathrm{w}}}(\Phi_i,\Phi_{j_i}) & \\
= & 0 & (\Phi_i \equiv_{\mathrm{w}} \Phi_{j_i} \text{ for each } j_i\in J_i \text{ and Def. 29})
\end{array}$$

where the inequality follows by observing that function $\tilde{\mathfrak{w}}$ defined by $\tilde{\mathfrak{w}}(\Phi_i,\Phi_{j_h}) = r_{j_i}$ if $h=i$ and $\tilde{\mathfrak{w}}(\Phi_i,\Phi_{j_h})=0$ otherwise, is a matching in $\mathfrak{W}(\Psi_1,\Psi_2)$. □

Corollary 1. *$\mathscr{Z}_1, \mathscr{Z}_2 \in \mathrm{Res}(\mathbf{S})$ are weak trace distribution equivalent iff $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}(\Psi_{\mathscr{Z}_1},\Psi_{\mathscr{Z}_2})=0$.*

Proof. ($\Rightarrow$) Assume first that $\mathscr{Z}_1$ and $\mathscr{Z}_2$ are weak trace distribution equivalent. Then from Theorem 4 we infer that $\Psi_{\mathscr{Z}_1} \equiv_{\mathrm{w}} \Psi_{\mathscr{Z}_2}$. By Theorem 9 this implies $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}(\Psi_{\mathscr{Z}_1},\Psi_{\mathscr{Z}_2})=0$.

($\Leftarrow$) Assume now that $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}(\Psi_{\mathscr{Z}_1},\Psi_{\mathscr{Z}_2})=0$. Then from Theorem 9 we infer that $\Psi_{\mathscr{Z}_1} \equiv_{\mathrm{w}} \Psi_{\mathscr{Z}_2}$. By Theorem 4 this implies that $\mathscr{Z}_1$ and $\mathscr{Z}_2$ are weak trace distribution equivalent. □

By the Hausdorff functional we lift the pseudometric $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}$ to a pseudometric over processes.

Definition 31. The $\mathbb{L}_{\mathrm{w}}$-*distance* over processes $\mathscr{D}_{\mathbb{L}_{\mathrm{w}}} : \mathbf{S}\times\mathbf{S} \to [0,1]$ is defined, for all $s,t\in\mathbf{S}$, by

$$\mathscr{D}_{\mathbb{L}_{\mathrm{w}}}(s,t) = \mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}})(\mathbb{L}_{\mathrm{w}}(s),\mathbb{L}_{\mathrm{w}}(t)).$$

Proposition 13. *The mapping $\mathscr{D}_{\mathbb{L}_{\mathrm{w}}}$ is a 1-bounded pseudometric over $\mathbf{S}$.*

Proof. The same arguments used in the proof of Proposition 10 apply. □

Proposition 14. *Let $s\in\mathbf{S}$. The set $\mathbb{L}_{\mathrm{w}}(s)$ is a closed subset of $\mathbb{L}_{\mathrm{w}}$ wrt. the topology induced by $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}$.*

Proof. Since $(\mathbb{L}^{\mathrm{d}}_{\mathrm{w}}, \mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}})$ is a pseudometric space (Proposition 12 and Theorem 9), to prove the thesis we need to show that the quotient space $\mathbb{L}_{\mathrm{w}}(s)_{/\equiv_{\mathrm{w}}}$ is a closed subset of $\mathbb{L}_{\mathrm{w}/\equiv_{\mathrm{w}}}$ with respect to the topology induced by $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}$ (in fact $(\mathbb{L}^{\mathrm{d}}_{\mathrm{w}/\equiv_{\mathrm{w}}}, \mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}})$ is a metric space). From Remark 2 we have that $\mathbb{L}^{\mathrm{d}}_{\mathrm{w}/\equiv_{\mathrm{w}}} = \mathbb{L}^{\mathrm{d}}$ and $\mathbb{L}_{\mathrm{w}}(s)_{/\equiv_{\mathrm{w}}} = \mathbb{L}(s)$. Moreover, we have that $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}{}_{|\mathbb{L}^{\mathrm{d}}_{\mathrm{w}/\equiv_{\mathrm{w}}}} = \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}$. Hence, the same arguments used in the proof of Proposition 11 allow us to prove that $\mathbb{L}_{\mathrm{w}}(s)_{/\equiv_{\mathrm{w}}}$ is a closed subset of $\mathbb{L}_{\mathrm{w}/\equiv_{\mathrm{w}}}$ wrt. the topology induced by $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}$. This gives the result also for $\mathbb{L}_{\mathrm{w}}(s)$ wrt to $\mathbb{L}_{\mathrm{w}}$ and $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}$. □

Theorem 10. *The kernel of $\mathscr{D}_{\mathbb{L}_{\mathrm{w}}}$ is weak trace equivalence.*

Proof. ($\Rightarrow$) Assume that $s \approx_{\mathrm{wt}} t$. We aim to show that $\mathscr{D}_{\mathbb{L}_{\mathrm{w}}}(s,t)=0$. By Theorem 5 we have that $s\approx_{\mathrm{wt}} t$ implies that $\mathbb{L}_{\mathrm{w}}(s) \equiv_{\mathrm{w}}^{\dagger} \mathbb{L}_{\mathrm{w}}(t)$. Since the kernel of $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}$ is given by $\equiv_{\mathrm{w}}^{\dagger}$ (Theorem 9), we can infer

$$\mathscr{D}_{\mathbb{L}_{\mathrm{w}}}(s,t) = \mathbf{H}(\mathscr{D}^{\mathrm{t}}_{\mathbb{L}_{\mathrm{w}}})(\mathbb{L}_{\mathrm{w}}(s),\mathbb{L}_{\mathrm{w}}(t)) = 0.$$

($\Leftarrow$) Assume now that $\mathscr{D}_{\mathbb{L}_{\mathrm{w}}}(s,t)=0$. We aim to show that this implies that $s\approx_{\mathrm{wt}} t$. Since (i) $\mathbb{L}_{\mathrm{w}}(s)$ and $\mathbb{L}_{\mathrm{w}}(t)$ are closed by Proposition 14, (ii) $\mathscr{D}_{\mathbb{L}_{\mathrm{w}}}$ is a pseudometric by Proposition 13 and (iii) the kernel of $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}_{\mathrm{w}}}$ is $\equiv_{\mathrm{w}}^{\dagger}$ by Theorem 9, from $\mathscr{D}_{\mathbb{L}_{\mathrm{w}}}(s,t)=0$ we can infer $\mathbb{L}_{\mathrm{w}}(s) \equiv_{\mathrm{w}}^{\dagger} \mathbb{L}_{\mathrm{w}}(t)$. Then, by Theorem 5 we can conclude $s\approx_{\mathrm{wt}} t$. □

Finally, we obtain the characterization of the weak trace metric.

Theorem 11 (Characterization of weak trace metric)**.** *For all $s,t \in \mathbf{S}$ we have $\mathbf{d}^{\mathrm{w}}_T(s,t) = \mathscr{D}_{\mathbb{L}_{\mathrm{w}}}(s,t)$.*

Proof. The same arguments used in the proof of Thm 8 apply. □

7 From boolean to real semantics

In this section we focus on $\mathbb{L}$ and we exploit the distance between formulae to define a real valued semantics for it, namely given a process s we assign to each formula a value in $[0,1]$ expressing the probability that s satisfies it. Then we show that our logical characterization of trace metric can be restated in terms of the general schema $\mathbf{d}_T(s,t) = \sup_{\Psi \in \mathbb{L}^{\mathrm{d}}} |\,[\Psi](s) - [\Psi](t)\,|$ where $[\Psi](s)$ denotes the value of the formula Ψ at process s, accordingly to the new real valued semantics. We remark that although, due to space restrictions, we present only the result for $\mathbb{L}$, the technique we propose would lead to the same results when applied to $\mathbb{L}_{\mathrm{w}}$.

First of all, we recall the notion of *distance function*, namely the distance between a point and a set.

Definition 32 (Distance function)**.** Let $\mathbb{L}' \subseteq \mathbb{L}^{\mathrm{d}}$. Given any $\Psi \in \mathbb{L}^{\mathrm{d}}$ we denote by $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}')$ the *distance between Ψ and the set $\mathbb{L}'$* defined by $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}') = \inf_{\Psi' \in \mathbb{L}'} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\Psi')$.

Then we obtain the following characterization of the Hausdorff distance.

Proposition 15. *Let $\mathbb{L}_1,\mathbb{L}_2 \subseteq \mathbb{L}^{\mathrm{d}}$. Then it holds that $\mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\mathbb{L}_1,\mathbb{L}_2) = \sup_{\Psi \in \mathbb{L}^{\mathrm{d}}} |\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_1) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_2)|$.*

Proof. It is clear that

$$\mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\mathbb{L}_1,\mathbb{L}_2) = \max\left\{ \sup_{\Psi_1 \in \mathbb{L}_1} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\mathbb{L}_2),\ \sup_{\Psi_2 \in \mathbb{L}_2} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_2,\mathbb{L}_1) \right\}. \tag{38}$$

Firstly we show that

$$\mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\mathbb{L}_1,\mathbb{L}_2) \leq \sup_{\Psi \in \mathbb{L}^{\mathrm{d}}} |\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_1) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_2)|. \tag{39}$$

Without loss of generality, we can assume that $\mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\mathbb{L}_1,\mathbb{L}_2) = \sup_{\Psi_1 \in \mathbb{L}_1} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\mathbb{L}_2)$. Then we have

$$\begin{array}{rl} \sup_{\Psi_1 \in \mathbb{L}_1} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\mathbb{L}_2) = & \sup_{\Psi_1 \in \mathbb{L}_1} |\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\mathbb{L}_2) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\mathbb{L}_1)| \\ \leq & \sup_{\Psi \in \mathbb{L}^{\mathrm{d}}} |\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_2) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_1)| \end{array}$$

from which Equation (39) holds.

Next, we aim to show the converse inequality, namely

$$\mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\mathbb{L}_1,\mathbb{L}_2) \geq \sup_{\Psi \in \mathbb{L}^{\mathrm{d}}} |\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_1) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_2)|. \tag{40}$$

To this aim, we show that

$$\text{for each } \Psi \in \mathbb{L}^{\mathrm{d}} \text{ it holds } |\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_1) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_2)| \leq \mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\mathbb{L}_1,\mathbb{L}_2). \tag{41}$$

- Assume $\Psi \in \mathbb{L}_1$. Then $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_1) = 0$ so that $|\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_1) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_2)| = \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_2)$. Moreover

  $$\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_2) \leq \sup_{\Psi_1 \in \mathbb{L}_1} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_1,\mathbb{L}_2) \leq \mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\mathbb{L}_1,\mathbb{L}_2)$$

  and Equation (41) follows in this case.

- The case of $\Psi \in \mathbb{L}_2$ is analogous and therefore Equation (41) follows also in this case.
- Finally, assume that $\Psi \notin \mathbb{L}_1 \cup \mathbb{L}_2$. Without loss of generality, we can assume that $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_1) \geq \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_2)$. By definition of *infimum* it holds that for each $\varepsilon > 0$ there is a formula $\Psi_\varepsilon \in \mathbb{L}_2$ s.t.

$$\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\Psi_\varepsilon) < \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_2) + \varepsilon. \tag{42}$$

Analogously, for each $\varepsilon' > 0$ and for each $\Psi_2 \in \mathbb{L}_2$ there is a $\Psi_{\varepsilon'} \in \mathbb{L}_1$ s.t.

$$\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_2,\Psi_{\varepsilon'}) < \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_2,\mathbb{L}_1) + \varepsilon'. \tag{43}$$

Let us fix $\varepsilon,\varepsilon' > 0$. Then let $\Psi_\varepsilon \in \mathbb{L}_2$ be the formula realizing Equation (42), with respect to Ψ, and let $\tilde{\Psi}_{\varepsilon'}$ be the formula in $\mathbb{L}_1$ realizing Equation (42), with respect to this Ψ_ε. Therefore, we have

$$\begin{array}{lll}
& |\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_1) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_2)| & \\
= & \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_1) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_2) & \\
< & \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_1) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\Psi_\varepsilon) + \varepsilon & \text{(by Equation (42))} \\
= & \inf_{\Psi_1 \in \mathbb{L}_1} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\Psi_1) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\Psi_\varepsilon) + \varepsilon & \text{(by Definition 32)} \\
< & \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\tilde{\Psi}_{\varepsilon'}) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\Psi_\varepsilon) + \varepsilon & \\
\leq & \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\Psi_\varepsilon) + \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_\varepsilon,\tilde{\Psi}_{\varepsilon'}) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\Psi_\varepsilon) + \varepsilon & \text{(by triangle inequality)} \\
= & \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_\varepsilon,\tilde{\Psi}_{\varepsilon'}) + \varepsilon & \\
< & \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_\varepsilon,\mathbb{L}_1) + \varepsilon' + \varepsilon & \text{(by Equation (43))} \\
\leq & \sup_{\Psi_2 \in \mathbb{L}_2} \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi_2,\mathbb{L}_1) + \varepsilon' + \varepsilon & \\
\leq & \mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\mathbb{L}_1,\mathbb{L}_2) + \varepsilon' + \varepsilon & \text{(by Equation (38)).}
\end{array}$$

Summarizing, we have obtained that

$$|\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_1) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}_2)| < \mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\mathbb{L}_1,\mathbb{L}_2) + \varepsilon' + \varepsilon$$

and since this inequality holds for each ε and ε', we can conclude that Equation (41) holds.

Equation (39) and Equation (40) taken together prove the thesis. □

To define the real-valued semantics of $\mathbb{L}^{\mathrm{d}}$ we exploit the distance $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}$. Informally, to quantify how much the formula Ψ is satisfied by process s we evaluate first how far Ψ is from being satisfied by s. This corresponds to the minimal distance between Ψ and a formula satisfied by s, namely to $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}(s))$. Then we simply notice that, as our distances are all 1-bounded, being $\mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}(s))$ far from s is equivalent to be $1 - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}(s))$ close to it. Thus we assign to Ψ the real value $1 - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}(s))$ in s.

Definition 33 (Real-valued semantics of $\mathbb{L}^{\mathrm{d}}$)**.** We define the *real-valued semantics* of $\mathbb{L}^{\mathrm{d}}$ as the function $[\_](\_)\colon \mathbb{L}^{\mathrm{d}} \times \mathbf{S} \to [0,1]$ defined for all $\Psi \in \mathbb{L}^{\mathrm{d}}$ and $s \in \mathbf{S}$ as $[\Psi](s) = 1 - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}(s))$.

We can restate our characterization theorem (Theorem 3) as a probabilistic $\mathbb{L}^{\mathrm{d}}$-model checking problem.

Theorem 12 (Characterization of strong trace metric II)**.** *For all $s,t \in \mathbf{S}$ we have*

$$\mathbf{d}_T(s,t) = \sup_{\Psi \in \mathbb{L}^{\mathrm{d}}} |\, [\Psi](s) - [\Psi](t) \,|\,.$$

Proof. From Theorem 3 we have $\mathbf{d}_T(s,t) = \mathscr{D}_{\mathbb{L}}(s,t)$. Hence the thesis is equivalent to prove

$$\mathscr{D}_{\mathbb{L}}(s,t) = \sup_{\Psi \in \mathbb{L}^{\mathrm{d}}} |\, [\Psi](s) - [\Psi](t) \,|\,.$$

We have

$$\begin{array}{rll} \mathscr{D}_{\mathbb{L}}(s,t) = & \mathbf{H}(\mathscr{D}^{\mathrm{d}}_{\mathbb{L}})(\mathbb{L}(s),\mathbb{L}(t)) & \text{(by Definition 28)} \\ = & \sup_{\Psi\in\mathbb{L}^{\mathrm{d}}} \mid \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}(s)) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}(t)) \mid & \text{(by Proposition 15)} \\ = & \sup_{\Psi\in\mathbb{L}^{\mathrm{d}}} \mid \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}(s)) - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}(t)) + 1 - 1 \mid & \\ = & \sup_{\Psi\in\mathbb{L}^{\mathrm{d}}} \mid 1 - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}(t)) - \big(1 - \mathscr{D}^{\mathrm{d}}_{\mathbb{L}}(\Psi,\mathbb{L}(s))\big) \mid & \\ = & \sup_{\Psi\in\mathbb{L}^{\mathrm{d}}} \mid [\Psi](t) - [\Psi](s) \mid & \text{(by Definition33).} \end{array}$$

□

8 Concluding remarks

We have provided a logical characterization of the strong and weak variants of trace metric on finite processes in the PTS model. Our results are based on the definition of a *distance* over the two-sorted boolean logics $\mathbb{L}$ and $\mathbb{L}_{\mathrm{w}}$, which we have proved to characterize resp. strong and weak probabilistic trace equivalence by exploiting the notion of *mimicking formula* of a resolution.

Our distance is a 1-bounded pseudometric that quantifies the syntactic disparities of the formulae and we have proved that the trace metric corresponds to the distance between the sets of formulae satisfied by the two processes. This approach, already successfully applied in [8] to the characterization of the bisimilarity metric, is not standard. Logical characterizations of the trace metrics have been obtained in terms of the probabilistic L-model checking problem, where L is the class of logical properties of interest, [1, 3, 9]. However we have proved that our approach can be exploited to regain classical one: by means of our distance between formulae we have defined a real-valued semantics for $\mathbb{L}$, namely a probabilistic model checking of a formula in a process, and then we have proved that the trace metric constitutes the least upper bound to the error that can be observed in the verification of an $\mathbb{L}$ formula.

Another interesting feature of our approach is its generality, since it can be easily applied to some variants of the trace equivalence and trace metric. In [4, 26] the authors distinguish between resolutions obtained via deterministic schedulers and the ones obtained via randomized schedulers. The only difference between the two classes is in the evaluation of the probability weights: in deterministic resolutions, which are the ones we have considered in this paper, each possible resolution of nondeterminism is considered singularly and thus the target probability distributions of their transitions are the same as in the considered process. In randomized resolutions, internal nondeterminism is solved by assigning a probability weight to each choice and thus the target distributions are obtained from the convex combination of the target distributions of the considered process. Since the definition of the mimicking formulae depends solely on the values of the probability weights in the resolutions and not on how these weights are evaluated, our characterization can be applied also to the case of trace equivalences and metrics defined in terms of randomized resolutions.

As a first step in the future development of our work, we aim to extend our results to the trace equivalence defined in [4] which, differently from the equivalence of [26] considered in this paper, is compositional wrt. the parallel composition operator. Roughly speaking, in [4] for each given trace it is checked whether the resolutions of two processes assign the same probability it, whereas in [26] for a chosen resolution of the first process we check whether there is a resolution for the second process that assigns the same probability to all traces. Furthermore, no trace metric has been defined yet for the equivalence in [4]. Our idea is then firstly to define such a trace metric and secondly to simplify the logic $\mathbb{L}$ by substituting the trace distribution formulae with a simple test on the execution probability of a trace, with an operator similar to the probabilistic operator in [25]. By applying our approach to the new logic we will obtain the characterization of the trace equivalence and metric.

Then, we will study metrics and logical characterizations for the testing equivalences defined in [4].

Finally, in [3] a sequence of Kantorovich bisimilarity-like metrics converging to the trace metric on MCs is provided. Hence we aim to combine our characterization results in [8] with the ones in this paper in order to see if a similar result of convergence can be obtained also with our technique on PTSs.

References

[1] Luca de Alfaro, Marco Faella & Mariëlle Stoelinga (2009): *Linear and Branching System Metrics*. *IEEE Trans. Software Eng.* 35(2), pp. 258–273, doi:10.1109/TSE.2008.106.

[2] Luca de Alfaro, Rupak Majumdar, Vishwanath Raman & Mariëlle Stoelinga (2008): *Game Refinement Relations and Metrics*. *Logical Methods in Computer Science* 4(3).

[3] Giorgio Bacci, Giovanni Bacci, Kim G. Larsen & Radu Mardare (2015): *Converging from Branching to Linear Metrics on Markov Chains*. In: *Proceedings of ICTAC 2015*, pp. 349–367, doi:10.1007/978-3-319-25150-9_21.

[4] Marco Bernardo, Rocco De Nicola & Michele Loreti (2014): *Revisiting Trace and Testing Equivalences for Nondeterministic and Probabilistic Processes*. *Logical Methods in Computer Science* 10(1), doi:10.2168/LMCS-10(1 : 16)2014.

[5] Franck van Breugel (2005): *A Behavioural Pseudometric for Metric Labelled Transition Systems*. In: *Proceedings of CONCUR 2005*, pp. 141–155, doi:10.1007/11539452_14.

[6] Franck van Breugel & James Worrell (2001): *Towards Quantitative Verification of Probabilistic Transition Systems*. In: *Proceedings of ICALP*, pp. 421–432, doi:10.1007/3-540-48224-5_35.

[7] Franck van Breugel & James Worrell (2005): *A behavioural pseudometric for probabilistic transition systems*. *Theor. Comput. Sci.* 331(1), pp. 115–142.

[8] Valentina Castiglioni, Daniel Gebler & Simone Tini (2016): *Logical Characterization of Bisimulation Metrics*. In: *Proceedings of QAPL 2016*, Electronic Proceedings in Theoretical Computer Science, doi:10.4204/EPTCS.227.4.

[9] Przemyslaw Daca, Thomas A. Henzinger, Jan Kretínský & Tatjana Petrov (2016): *Linear Distances between Markov Chains*. In: *Proceedings of CONCUR 2016*, pp. 20:1–20:15, doi:10.4230/LIPIcs.CONCUR.2016.20.

[10] Yuxin Deng, Tom Chothia, Catuscia Palamidessi & Jun Pang (2006): *Metrics for Action-labelled Quantitative Transition Systems*. *Electronic Notes in Theoretical Computer Science* 153(2), pp. 79–96, doi:10.1016/j.entcs.2005.10.033.

[11] Yuxin Deng & Wenjie Du (2011): *Logical, Metric, and Algorithmic Characterisations of Probabilistic Bisimulation*. *CoRR* abs/1103.4577.

[12] Josee Desharnais, Vineet Gupta, Radha Jagadeesan & Prakash Panangaden (2004): *Metrics for labelled Markov processes*. *Theoretical Computer Science* 318(3), pp. 323–354, doi:10.1016/j.tcs.2003.09.013.

[13] Josée Desharnais, Radha Jagadeesan, Vineet Gupta & Prakash Panangaden (2002): *The Metric Analogue of Weak Bisimulation for Probabilistic Processes*. In: *Proc. LICS 2002*, pp. 413–422.

[14] Wenjie Du, Yuxin Deng & Daniel Gebler (2016): *Behavioural Pseudometrics for Nondeterministic Probabilistic Systems*. In: *Proceedings of SETTA 2016*, pp. 67–84, doi:10.1007/978-3-319-47677-3_5.

[15] D. Gebler, K. G. Larsen & S. Tini (2016): *Compositional bisimulation metric reasoning with Probabilistic Process Calculi*. *Logical Methods in Computer Science* 12(4).

[16] Daniel Gebler & Simone Tini (2015): *SOS Specifications of Probabilistic Systems by Uniformly Continuous Operators*. In: *Proc. CONCUR 2015*, pp. 155–168.

[17] Alessandro Giacalone, Chi-Chang Jou & Scott A. Smolka (1990): *Algebraic Reasoning for Probabilistic Concurrent Systems*. In: *Proc. IFIP Work. Conf. on Programming, Concepts and Methods*, pp. 443–458.

[18] Hans Hansson & Bengt Jonsson (1994): *A logic for reasoning about time and reliability*. Formal Aspects of Computing 6(5), pp. 512–535, doi:10.1007/BF01211866.

[19] Matthew Hennessy & Robin Milner (1985): *Algebraic laws for nondeterminism and concurrency*. J. Assoc. Comput. Mach. 32, pp. 137–161.

[20] Holger Hermanns, Augusto Parma, Roberto Segala, Björn Wachter & Lijun Zhang (2011): *Probabilistic Logical Characterization*. Information and Computation 209(2), pp. 154–172, doi:10.1016/j.ic.2010.11.024.

[21] Leonid V. Kantorovich (1942): *On the Transfer of Masses*. Original article in Russian, translation in Management Science, $5 : 1-4(1959)$.

[22] Robert M. Keller (1976): *Formal Verification of Parallel Programs*. Commun. ACM 19(7), pp. 371–384, doi:10.1145/360248.360251.

[23] Marta Z. Kwiatkowska & Gethin Norman (1996): *Probabilistic Metric Semantics for a Simple Language with Recursion*. In: Proc. MFCS'96, pp. 419–430.

[24] Kim G. Larsen, Radu Mardare & Prakash Panangaden (2012): *Taking It to the Limit: Approximate Reasoning for Markov Processes*. In: Proc. MFCS 2012, pp. 681–692.

[25] Augusto Parma & Roberto Segala (2007): *Logical Characterizations of Bisimulations for Discrete Probabilistic Systems*. In: Proceedings of FoSSaCS 2007, pp. 287–301, doi:10.1007/978-3-540-71389-0_21.

[26] Roberto Segala (1995): *A Compositional Trace-Based Semantics for Probabilistic Automata*. In: Proceedings of CONCUR '95, pp. 234–248, doi:10.1007/3-540-60218-6_17.

[27] Roberto Segala (1995): *Modeling and Verification of Randomized Distributed Real-Time Systems*. Ph.D. thesis, MIT.

[28] Lin Song, Yuxin Deng & Xiaojuan Cai (2007): *Towards Automatic Measurement of Probabilistic Processes*. In: Proceedings of QSIC 2007, pp. 50–59, doi:10.1109/QSIC.2007.65.

[29] William J. Stewart (1994): *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press.

Language-based abstractions for dynamical systems

Andrea Vandin
IMT School for Advanced Studies Lucca, Lucca, Italy
andrea.vandin@imtlucca.it

Ordinary differential equations (ODEs) are the primary means to model dynamical systems in a wide range of natural and engineering sciences. When the complexity of the considered system is high, the number of equations required limits our capability of performing effective analysis. This has motivated a large body of research, across many disciplines, into abstraction techniques that provide smaller ODE systems preserving the original dynamics in some appropriate sense (e.g., [2, 14, 15, 5]).

Our own line of research [10, 18, 7, 9, 20, 8, 6, 13] consists of a computer science perspective to this problem, borrowing ideas from the concurrency theory community. We recast the ODE reduction problem to that of finding an appropriate equivalence relation over ODE variables, akin to classical models of computation based on labelled transition systems. We studied such "differential equivalences" for two basic intermediate languages, trading expressivity for efficiency:

i) *IDOL* (Intermediate Drift-Oriented Language) [9] covers a general class of non-linear ODEs with derivatives containing polynomials, rationals, minima/maxima, and absolute values. This is sufficient, e.g., to capture the existing ODE semantics of stochastic process algebras [13, 4]. The largest equivalences of IDOL terms are computed using a symbolic partition-refinement algorithm that exploits an encoding into a satisfiability modulo theories (SMT) problem;

ii) *Reaction networks* [6, 8], a slight generalization of chemical reaction networks, characterise ODEs with polynomial derivatives. In this case, the partition refinement is based on Paige and Tarjan's seminal proposal [16], giving an efficient algorithm that runs in polynomial time.

A tutorial-like presentation unifying the two approaches can be found in [20], while [7, 18] address the more general problem of computing all differential equivalences of a model. Our framework for ODE reduction has been implemented in the tool ERODE [10] (`http://sysma.imtlucca.it/tools/erode/`), allowing us to provide evidence of effective reductions in realistic models from the literature.

We remark that our techniques consider exact aggregations. In some cases, however, one might be interested in more permissive, approximate, notions that do not discriminate ODE variables with nearby trajectories in practice (e.g., [17, 1, 3, 11, 19, 12]). In an ongoing research we are developing approximate variants of our differential equivalences, aiming at maintaining computational tractability, and certified error bounds that do not grow fast with time.

References

[1] Alessandro Aldini, Mario Bravetti & Roberto Gorrieri (2004): *A process-algebraic approach for the analysis of probabilistic noninterference*. Journal of Computer Security 12(2), pp. 191–245.

[2] Masanao Aoki (1968): *Control of large-scale dynamic systems by aggregation*. IEEE Trans. Autom. Control 13(3), pp. 246–253, doi:10.1109/TAC.1968.1098900.

[3] Franck van Breugel & James Worrell (2006): *Approximating and computing behavioural distances in probabilistic transition systems*. Theoretical Computer Science 360(1–3), pp. 373–385.

[4] Luca Cardelli (2008): *On process rate semantics*. Theoretical Computer Science 391(3), pp. 190–215.

Submitted to:
QAPL 2017

[5] Luca Cardelli, Attila Csikász-Nagy, Neil Dalchau, Mirco Tribastone & Max Tschaikowski (2016): *Noise Reduction in Complex Biological Switches*. Scientific Reports 6, p. 20214.

[6] Luca Cardelli, Mirco Tribastone, Max Tschaikowski & Andrea Vandin (2015): *Forward and Backward Bisimulations for Chemical Reaction Networks*. In: *CONCUR 2015*, pp. 226–239, doi:10.4230/LIPIcs.CONCUR.2015.226.

[7] Luca Cardelli, Mirco Tribastone, Max Tschaikowski & Andrea Vandin (2016): *Comparing Chemical Reaction Networks: A Categorical and Algorithmic Perspective*. In: *LICS 2016*, pp. 485–494, doi:10.1145/2933575.2935318.

[8] Luca Cardelli, Mirco Tribastone, Max Tschaikowski & Andrea Vandin (2016): *Efficient Syntax-Driven Lumping of Differential Equations*. In: *TACAS 2016*, pp. 93–111, doi:10.1007/978-3-662-49674-9_6.

[9] Luca Cardelli, Mirco Tribastone, Max Tschaikowski & Andrea Vandin (2016): *Symbolic computation of differential equivalences*. In: *POPL 2016*, pp. 137–150, doi:10.1145/2837614.2837649.

[10] Luca Cardelli, Mirco Tribastone, Max Tschaikowski & Andrea Vandin (2017): *ERODE: A Tool for the Evaluation and Reduction of Ordinary Differential Equations*. In: *TACAS 2017*. To appear.

[11] Vineet Gupta, Radha Jagadeesan & Prakash Panangaden (2006): *Approximate reasoning for real-time probabilistic processes*. *LMCS* 2(1), doi:http://dx.doi.org/10.2168/LMCS-2(1:4)2006.

[12] Giulio Iacobelli & Mirco Tribastone (2013): *Lumpability of fluid models with heterogeneous agent types*. In: *DSN*, pp. 1–11.

[13] Giulio Iacobelli, Mirco Tribastone & Andrea Vandin (2015): *Differential Bisimulation for a Markovian Process Algebra*. In: *MFCS 2015*, pp. 293–306, doi:10.1007/978-3-662-48057-1_23.

[14] Yoh Iwasa, Viggo Andreasen & Simon Levin (1987): *Aggregation in model ecosystems. I. Perfect aggregation*. Ecological Modelling 37(3-4), pp. 287–302.

[15] Miles S. Okino & Michael L. Mavrovouniotis (1998): *Simplification of Mathematical Models of Chemical Reaction Systems*. Chemical Reviews 2(98), pp. 391–408.

[16] Robert Paige & Robert Tarjan (1987): *Three Partition Refinement Algorithms*. SIAM Journal on Computing 16(6), pp. 973–989.

[17] Alessandra Di Pierro, Chris Hankin & Herbert Wiklicky (2003): *Quantitative Relations and Approximate Process Equivalences*. In: *CONCUR*, pp. 498–512. Available at `http://dx.doi.org/10.1007/978-3-540-45187-7_33`.

[18] Stefano Tognazzi, Mirco Tribastone, Max Tschaikowski & Andrea Vandin (2017): *EGAC: A Genetic Algorithm to Compare Chemical Reaction Networks*. In: *GECCO 2017*. To appear.

[19] Max Tschaikowski & Mirco Tribastone (2016): *Approximate Reduction of Heterogenous Nonlinear Models With Differential Hulls*. *IEEE Trans. Automat. Contr.* 61(4), pp. 1099–1104.

[20] Andrea Vandin & Mirco Tribastone (2016): *Quantitative Abstractions for Collective Adaptive Systems*. In: *SFM 2016, Bertinoro Summer School*, pp. 202–232, doi:10.1007/978-3-319-34096-8_7.

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

Diego Latella Mieke Massink
Consiglio Nazionale delle Ricerche
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
{Diego.Latella, Mieke.Massink}@cnr.it

Collective Adaptive Systems (CAS) consist of a large number of interacting objects. The design of such systems requires scalable analysis tools and methods, which have necessarily to rely on some form of approximation of the system's actual behaviour. Promising techniques are those based on mean-field approximation. The FlyFast model-checker uses an on-the-fly algorithm for bounded PCTL model-checking of selected individual(s) in the context of very large populations whose global behaviour is approximated using deterministic limit mean-field techniques. Recently, a front-end for FlyFast has been proposed which provides a modelling language, PiFF in the sequel, for the *Predicate-based* Interaction for FlyFast. In this paper we present details of PiFF design and an approach to state-space reduction based on probabilistic bisimulation for inhomogeneous DTMCs.

1 Introduction

Collective Adaptive Systems (CAS) consist of a large number of entities with decentralised control and varying degrees of complex autonomous behaviour. They form the basis of many modern *smart city* critical infrastructures. Consequently, their design requires support from formal methods and scalable automatic tools based on solid mathematical foundations. In [28, 26], Latella et al. presented a scalable mean-field model-checking procedure for verifying bounded Probabilistic Computation Tree Logic (PCTL, [18]) properties of an individual[1] in the context of a system consisting of a large number of interacting objects. The model-checking procedure is implemented in the tool FlyFast[2]. The procedure performs on-the-fly, mean-field, approximated model-checking based on the idea of fast simulation, as introduced in [29]. More specifically, the behaviour of a generic agent with S states in a system with a *large* number N of instances of the agent at given step (i.e. time) t is approximated by $\mathbf{K}(\mu(t))$ where $\mathbf{K}(\mathbf{m})$ is the $S \times S$ probability transition matrix of an (inhomogeneous) DTMC and $\mu(t)$ is a vector of size S approximating the mean behaviour of (the rest of) the system at t; each element of $\mu(t)$ is associated with a distinct state of the agent, say C, and gives an approximation of the fraction of instances of the agent that are in state C in the global system, at step t. Note that such an approximation is a *deterministic* one, i.e. μ is a *function* of the step t (the exact behaviour of the rest of the system would instead be a *large* DTMC in turn); note furthermore, that the above transition matrix does not depend on N [28, 26].

Recently, modelling and programming languages have been proposed specifically for autonomic computing systems and CAS [11, 5]. Typically, in such frameworks, a system is composed of a set of independent *components* where a component is a process equipped also with a set of *attributes* describing features of the component. The attributes of a component can be *updated* during its execution so

[1] The technique can be applied also to a finite selection of individuals; in addition, systems with several distinct types of individuals can be dealt with; for the sake of simplicity, in the present paper we consider systems with many instances of a single individual only and we focus in the model-checking a single individual in such a context.

[2] http://j-sam.sourceforge.net/

Submitted to:
QAPL 2017

that the association between attribute *names* and attribute *values* is maintained in the dynamic *store* of the component. Attributes can be used in *predicates* appearing in language constructs for component interaction. The latter is thus typically modelled using *predicate-based* output/input *multicast*, originally proposed in [25], and playing a fundamental role in the interaction schemes of languages like SCEL [11] and CARMA [5]. In fact, predicate-based communication can be used by components to dynamically organise themselves into ensembles and as a means to dynamically select partners for interaction. Furthermore, it provides a way for representing component features, like for instance component location in space, which are fundamental for systems distributed in space, such as CAS [30].

In [8] we proposed a front-end modelling language for FlyFast that provides constructs for dealing with *components* and *predicate-based interaction*; in the sequel, the language—which has been inspired by CARMA— will be referred to as PiFF, which stands for for Predicate-based Interaction for FlyFast. Components interact via predicate-based communication. Each component consists of a behaviour, modelled as a DTMC-like agent, like in FlyFast, and a set of attributes. The attribute name-value correspondence is kept in the current store of the component. Actions are *predicate based multi-cast* output and input primitives; predicates are defined over attributes. Associated to each action there is also an (atomic) probabilistic store-update. For instance, assume components have an attribute named loc which takes values in the set of points of a space, thus recording the current location of the component. The following action models a multi-cast via channel α to all components in the same location as the sender, making it change location randomly: $\alpha^*[\mathsf{loc} = \mathbf{my}.\mathsf{loc}]\langle\rangle\,\mathtt{Jump}$. Here `Jump` is assumed to randomly update the store and, in particular attribute loc. The computational model is *clock-synchronous*, as in FlyFast, but at the component level. In addition, each component is equipped with a local *outbox*. The effect of an output action $\alpha^*[\pi_r]\langle\rangle\sigma$ is to deliver output label $\alpha\langle\rangle$ to the local outbox, together with the predicate π_r, which (the store of) the receiver components will be required to satisfy, as well as the current store of the component executing the action; the current store is updated according to update σ. Note that output actions are *non-blocking* and that successive output actions of the same component overwrite its outbox. An input action $\alpha^*[\pi_s]()\sigma$ by a component will be executed with a probability which is proportional to the *fraction* of all those components whose outboxes currently contain the label $\alpha\langle\rangle$, a predicate π_r which is satisfied by the component, and a store which satisfies in turn predicate π_s. If such a fraction is zero, then the input action will not take place (input is blocking), otherwise the action takes place, the store of the component is updated via σ, and its outbox cleared.

Related Work CAS are typically *large* systems, so that the formal analysis of models for such systems hits invariantly the state-space explosion problem. In order to mitigate this problem, the so called 'on-the-fly' paradigm is often adopted (see e.g. [9, 4, 20, 15]).

In the context of probabilistic model-checking several on-the-fly approaches have been proposed, among which [12], [27] and [17]. In [12], a probabilistic model-checker is shown for the *time bounded* fragment of PCTL. An on-the-fly approach for *full* PCTL model-checking is proposed in [27] where, actually, a specific *instantiation* is presented of an algorithm which is *parametric* with respect to the specific probabilistic processes modelling language and logic, and their specific semantics. Finally, in [17] an on-the-fly approach is used for detecting a maximal relevant search depth in an infinite state space and then a *global* model-checking approach is used for verifying bounded Continuous Stochastic Logic (CSL) [1, 2] formulas in a continuous time setting on the selected subset of states.

An on-the-fly approach by itself however, does not solve the challenging scalability problems that arise in truly large parallel systems, such as CAS. To address this type of scalability challenges in probabilistic model-checking, recently, several approaches have been proposed. In [19, 16] approximate

probabilistic model-checking is introduced. This is a form of statistical model-checking that consists in the generation of random executions of an *a priori* established maximal length [24]. On each execution the property of interest is checked and statistics are performed over the outcomes. The number of executions required for a reliable result depends on the maximal error-margin of interest. The approach relies on the analysis of individual execution traces rather than a full state space exploration and is therefore memory-efficient. However, the number of execution traces that may be required to reach a desired accuracy may be large and therefore time-consuming. The approach works for general models, i.e. models where stochastic behaviour can also be non Markovian and that do not necessarily model populations of similar objects. On the other hand, the approach is not independent from the number of objects involved. As recalled above, in [26] a scalable model-checking algorithm is presented that is based on mean-field approximation, for the verification of time bounded PCTL properties of an individual in the context of a system consisting of a large number of interacting objects. Correctness of the algorithm with respect to exact probabilistic model-checking has been proven in [26] as well. Also this algorithm is actually an instantiation of the above mentioned parametric algorithm for (exact) probabilistic model-checking [27], but the algorithm is instantiated on (time bounded PCTL and) the *approximate*, mean-field, semantics of a population process modelling language. It is worth pointing out that FlyFast allows users to perform simulations of their system models and to analyse the latter using their *exact* probabilistic semantics and *exact* PCTL model-checking. In addition, the tool provides *approximate* model-checking for bounded PCTL, using the model semantics based on mean-field.

The work of Latella et al. [26] is based on mean-field approximation in the *discrete time* setting; approximated mean-field model-checking in the *continuous time* setting has been presented in the literature as well, where the deterministic approximation of the global system behaviour is formalised as an initial value problem using a set of differential equations. Preliminary ideas on the exploitation of mean-field convergence in continuous time for model-checking were informally sketched in [22], but no model-checking algorithms were presented. Follow-up work on the above mentioned approach can be found in [23] which relies on earlier results on fluid model-checking by Bortolussi and Hillston [6], later published in [7], where a *global CSL* model-checking procedure is proposed for the verification of properties of a selection of individuals in a population, which relies on fast simulation results. This work is perhaps closest related to [26, 28]; however their procedure exploits mean-field convergence and fast simulation [10, 14] in a *continuous* time setting—using a set of differential equations—rather than in a discrete time setting—where an inductive definition is used. Moreover, that approach is based on an *interleaving* model of computation, rather than a clock-synchronous one; furthermore, a *global* model-checking approach, rather than an on-the-fly approach is adopted; it is also worth noting that the treatment of nested formulas, whose truth value may change over time, turns out to be much more difficult in the interleaving, continuous time, global model-checking approach than in the clock-synchronous, discrete time, on-the-fly one.

PiFF has been originally proposed in [8], where the complete formal, exact probabilistic, semantics of the language have been defined. The semantics definition consists of three transition rules—one for transitions associated with output actions, one for those associated with input actions, and one for transitions to be fired with residual probability. The rules induce a transition relation among component states and compute the relevant probabilities. From the component transition relation, a component one-step transition probability matrix is derived, the elements of which may depend on the fractions of the components in the system which are in a certain state. The system-wide one-step transition probability matrix is obtained by product—due to independence assumptions—using the above mentioned component probability matrix and the actual fractions in the current system global state. In [8] a translation of PiFF to the model specification language of FlyFast has also been presented which makes PiFF an additional front-

end for FlyFast extending its applicability to models of systems based on predicate-based interaction. In the above mentioned paper, correctness of the translation has been proved as well. In particular, it has been shown that the probabilistic semantics of any PiFF model are isomorphic to those of the translation of the model. In other words, the transition probability matrix of (the DTMCs of) the two models is the same. A companion translation of bounded PCTL formulas is also defined [8] and proven correct.

The notion of the outbox used in PiFF is reminiscent of the notion of the *ether* in PALOMA [13] in the sense that the collection of all outboxes together can be thought of as a kind of ether; but such a collection is intrinsically distributed among the components so that it cannot represent a bottleneck in the execution of the system neither a singularity point in the deterministic approximation.

We are not aware of other proposals, apart from [8], of probabilistic process languages, equipped both with standard, DTMC-based, semantics and with mean-field ones, that provide a predicate-based interaction framework, and that are fully supported by a tool for probabilistic simulation, exact and mean-field model-checking.

We conclude this section recalling that mean-field/fluid procedures are based on *approximations* of the global behaviour of a system. Consequently, the techniques should be considered as *complementary* to other, possibly more accurate but often not as scalable, analysis techniques for CAS, primarily those based on stochastic simulation, such as statistical model-checking.

In this paper we present some details of PiFF, a translation to FlyFast which simplifies that proposed in [8] and an approach to model reduction based on probabilistic bisimulation for Inhomogeneous DTMCs. In Section 2 we briefly present the main ingredients of the PiFF syntax and informal semantics, and we recall those features of FlyFast directly relevant for understanding the translation of PiFF to the FlyFast input language proposed in [8]. A revised and simplified version of the translation is described in Section 3. In Section 4 we introduce a simplified language for the definition of transition-probabilities in PiFF that allows us to define in Section 5 a model reduction procedure of the translation result, based on a notion of bisimulation for the kind of IDTMCs of interest, introduced in Section 5 as well. An example of application of the procedure is presented in Section 6. Some conclusions are drawn in Section 7. A formal proof of decidability of the cumulative probability test for state-space reduction based on bisimulation is provided in the Appendix.

2 Summary on PiFF and FlyFast

In the following we present the main ingredients of PiFF and the features of FlyFast relevant for the present paper.

2.1 PiFF

A PiFF system model specification $\Upsilon = (\Delta_\Upsilon, F_\Upsilon, \Sigma_0)^{(N)}$ is a triple where F_Υ is the set of relevant function definitions (e.g. store updates, auxiliary constants and functions), Δ_Υ is a set of state defining equations, and Σ_0 is the initial system state (an N-tuple of component states, each of which being a 3-tuple (C, γ, O) of agent state C, store γ and outbox O). We describe the relevant details below referring to [8] for the formal definition probabilistic semantics of the language.

The PiFF type system consists of floating point values and operations, as in FlyFast, plus simple enumeration types for attributes, declared according to the syntax **attype** $<$ `name` $>$ **enum** $<$ `id` $-$ `list` $>$. $<$ `id` $-$ `list` $>$ is a finite list of identifiers. Of course, attributes can also take floating point values.

In Figure 1 the attribute type `Space` is defined that consists of four values `A`,`B`,`C`,`D` modelling four

attype Space **enum** A, B, C, D;
⋮
const H = 0.6;
const L = 1 − H;
const Hdiv2 = H/2;
const Ldiv2 = L/2;
⋮
attribute loc : Space;
⋮
func Hr(x : Space) : Space; x **endfunc**;
func N(x : Space) : Space; **case** x **of** A : A; B : B; C : B; D : A **endfunc**;
func S(x : Space) : Space; **case** x **of** A : D; B : C; C : C; D : D **endfunc**;
func E(x : Space) : Space; **case** x **of** A : A; B : A; C : D; D : D **endfunc**;
func W(x : Space) : Space; **case** x **of** A : B; B : B; C : C; D : C **endfunc**;
⋮
func pHr(x : Space) : **float**; **case** x **of** A : H; B : L; C : H; D : L **endfunc**;
func pN(x : Space) : **float**; **case** x **of** A : 0; B : 0; C : Ldiv2; D : Hdiv2 **endfunc**;
func pS(x : Space) : **float**; **case** x **of** A : Ldiv2; B : Hdiv2; C : 0; D : 0 **endfunc**;
func pE(x : Space) : **float**; **case** x **of** A : 0; B : Hdiv2; C : Ldiv2; D : 0 **endfunc**;
func pW(x : Space) : **float**; **case** x **of** A : Ldiv2; B : 0; C : 0; D : Hdiv2 **endfunc**;
⋮
update Jump
my.loc := Hr(**my**.loc) **with** pHr(**my**.loc);
my.loc := N(**my**.loc) **with** pN(**my**.loc);
my.loc := S(**my**.loc) **with** pS(**my**.loc);
my.loc := E(**my**.loc) **with** pE(**my**.loc);
my.loc := W(**my**.loc) **with** pW(**my**.loc)
endupdate

Figure 1: A fragment of F_{SI}.

locations. Some auxiliary constants are defined, using the **const** construct inherited from FlyFast: **const** $<$ `name` $>$ $=$ $<$ `value` $>$.

A PiFF store update definition has the following syntax[3]:

update *upd*
my.$a_1 := e_{11}, \ldots,$ **my**.$a_k := e_{k1}$ **with** p_1;
$\vdots$
my.$a_1 := e_{1n}, \ldots,$ **my**.$a_k := e_{kn}$ **with** p_n
endupdate

where *upd* is the update name (unique within the system model specification), $a_1, \ldots, a_k$ are the attribute names of the component, $e_{11}, \ldots, e_{kn}$ and $p_1, \ldots, p_n$ are attribute/store-probability expressions respectively, with syntax defined according to the grammars $e ::= v_a \mid c_a \mid$ **my**.$a \mid fn_a(e_1, \ldots, e_m)$ and $p ::= v_p \mid c_p \mid fn_p(e_1, \ldots, e_m)$. In the above definition of attribute expressions v_a is an attribute value (drawn from finite set $\mathscr{V}$ of attribute values), c_a is an attribute constant in $\mathscr{V}$ defined using the **const**; $a \in \{a_1, \ldots, a_k\}$ is an attribute name and fn_a is an attribute function defined by the user in F_Υ, which, when applied to attribute expressions $e_1, \ldots, e_m$ returns an attribute value; the syntax for such function definitions *afd* is given below:

afd ::= **func** $fn_a(x_1 : T1, \ldots, x_m : Tm) : T$;*afb* **endfunc**
afb ::= e |**case** $(x_1, \ldots, x_m)$ **of**$(v_{a_{11}}, \ldots, v_{a_{m1}}) : e_1; (v_{a_{12}}, \ldots, v_{a_{m2}}) : e_2; \ldots (v_{a_{1k}}, \ldots, v_{a_{mk}}) : e_k$

where fn_a is the name of the attribute function, $x_1 : T1, \ldots, x_m : Tm$ are its parameters and their relative types, T is the type of the result of fn_a; e, e_i are attribute-expressions and $v_{a_{ij}}$ are attribute-values.

In Figure 1 attribute functions `N`, `S`, `E`, `W` are defined for North, South, East, and West, such that `Space` models the Cartesian space with four quadrants: `A` $=$ `N(D)` $=$ `E(B)`, `B` $=$ `N(C)` $=$ `W(A)`, and so on, as shown diagrammatically in Figure 2 *right*. Function `Hr` is the identity on `Space`.

In the definition of store-probability expressions $v_p \in (0,1]$, c_p is a store-probability constant in $(0,1]$ defined using the FlyFast **const** construct, and fn_p is a store-probability function defined by the user in F_Υ, which, when applied to attribute expressions $e_1, \ldots, e_m$ returns a probability value.The syntax for store-probability function definitions *pfd* is similar to that of attribute functions:

pfd ::= **func** $fn_p(x_1 : T1, \ldots, x_m : Tm)$: **float**;*pfb* **endfunc**
pfb ::= p |**case** $(x_1, \ldots, x_m)$ **of**$(v_{a_{11}}, \ldots, v_{a_{m1}}) : p_1; (v_{a_{12}}, \ldots, v_{a_{m2}}) : p_2; \ldots (v_{a_{1k}}, \ldots, v_{a_{mk}}) : p_k$

where fn_p is the name of the store-probability function, the result type is **float** (actually the range $[0,1]$) $x_1 : T1, \ldots, x_m : Tm$ are its parameters and their relative types, p, p_i are store-probability expressions and $v_{a_{ij}}$ are attribute-values. In any store update definition it must be guaranteed that the values of $p_1 \ldots p_n$ sum up[4] to 1. The informal meaning is clear. The store update will make attributes $a_1, \ldots, a_k$ take the values of $e_{1i}, \ldots, e_{ki}$ respectively with probability equal to the value of p_i.

In Figure 1 store-probability functions `pHr`, `pN`, `pS`, `pE`, `pW` are defined that give the probabilities of not moving (`pHr`), or of jumping to North (`pN`), South (`pS`), East (`pE`), West (`pW`), as functions of the

[3] In [8] a slightly different syntax for store updates has been used.

[4] In this version of the translation we allow only flat updates, i.e. the specific probability of each combination of values assigned to the attributes must be given explicitly. Other possibilities could be defined using *combinations* of (independent) probability distributions.

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

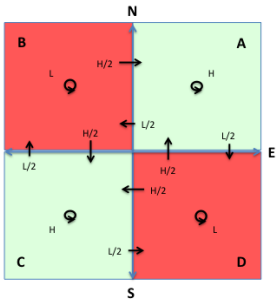


Figure 2: *SI*, a behavioural model.

current location.

Example 1 *A simplified version of the behaviour of the epidemic process discussed in [8] is shown in Figure 2* left[5]*. In Figure 1 we show a fragment of F_{SI} defining store update* `Jump` *together with the relevant type, constant and function definitions as introduced above. The component has just one attribute, named* loc*, with values in* `Space`*. The effect of* `Jump` *executed by a component in which* loc *is bound to quadrant ℓ is to leave the value of* loc *unchanged with probability* $\mathtt{pHr}(\ell)$*, change it to the quadrant North of ℓ with probability* $\mathtt{pN}(\ell)$*, and so on. Note that* $\mathtt{H} > \mathtt{L}$ *and this implies that higher probability is assigned to* `A` *and* `C` *and low probability to* `B` *and* `D`*. This is represented in Figure 2* right *where higher probability locations are shown in green and lower probability ones are shown in red; moreover, the relevant probabilities are represented as arrows (*$\mathtt{H}/2, \mathtt{L}/2$*) or self-loops (*$\mathtt{H}, \mathtt{L}$*). A susceptible (state* `S`*) component becomes infected (state* `I`*) via an* `inf` *action which takes place with probability equal to the fraction of components in the system which are currently infected (i.e.* $\mathsf{frc}(\mathtt{I})$*); it remains in state* `S` *via the self-loop labelled by action* `nsc`*, with probability* $\mathsf{frc}(\mathtt{S}) = 1 - \mathsf{frc}(\mathtt{I})$*. An infected node (state* `I`*) may recover, entering state* `S` *with action* `rec` *and probability* `ir`*; while infected, it keeps executing action* `inf`*, with probability* `ii`*. Note that, for the sake of simplicity, we use only* internal *actions, modelled by means of* output *actions with predicate false ($\bot$). We assume that in the initial global state all outboxes are non-empty; each contains the initial store of the specific component (i.e., its initial location), predicate $\bot$ and the empty tuple $\langle\rangle$).*

A PiFF state defining equation has the following (abstract) form: $C := \sum_{j\in J}[g_j]p_j :: act_j.C_j$ where either $[g_j]p_j$ is the keyword **rest** or:

- g_j is a boolean expression b which may depend on the current store, but not on the current occupancy measure vector: $b ::= \top \mid \bot \mid e \bowtie e \mid \neg b \mid b \wedge b$ and $e ::= v_a \mid c_a \mid \mathbf{my}.a$ where $\top$ ($\bot$) denotes the constant `true` (`false`), $\bowtie \in \{\geq, >, \leq, <\}$, v_a is an attribute value (drawn from finite set $\mathscr{V}$ of attribute values), c_a is an attribute constant in $\mathscr{V}$ defined using the FlyFast **const** construct, and a is the name of an attribute of the component.
- p_j is a transition probability expression $p ::= v_p \mid c_p \mid \mathsf{frc}(C) \mid \mathsf{frc}(\pi) \mid \prod_{i\in I} p_i \mid \sum_{i\in I} p_i \mid 1 - p$, for finite I, where $v_p \in (0,1]$, c_p a constant in $(0,1]$ defined via the **const** construct, and π is defined as b above, but where expressions e can also be attribute names a (i.e. $e ::= v_a \mid c_a \mid \mathbf{my}.a \mid a$); $\mathsf{frc}(C)$ is the fraction of components *currently* in state C over the total number N; similarly, $\mathsf{frc}(\pi)$ is the fraction of components the *current* store of which satisfies π, over the total number N. Note that it must be guaranteed that $\prod_{i\in I} p_i \leq 1$ and $\sum_{i\in I} p_i \leq 1$.

[5]We focus only on those features that are most relevant for the present paper. In [8] also other features are shown like, e.g. the use of (predicate-based) input actions, which are not the main subject of this paper.

- act_j can be an output action $\alpha^*[\pi]\langle\rangle\sigma$ or an input action $\alpha^*[\pi]()\sigma$, where π is as above and σ is the name of a store update. Note that in the case of an input action, π refers to the store of the partner component in the *previous* step of the computation.

If $[g_j]p_j = \textbf{rest}$, then act_j must be an output action $\alpha^*[\pi]\langle\rangle\sigma$, to be executed with the residual probability.

2.2 FlyFast

FlyFast accepts a specification $\langle\Delta, A, \mathbf{C}_0\rangle^{(N)}$ of a model of a system consisting of the clock-synchronous product of N instances of a probabilistic agent. The states of the DTMC-like agent model are specified by a set of state-defining equations Δ. The (abstract) form of a state defining equation is the following $C := \sum_{i=1}^{r} a_i.C_i$ where $a_i \in \mathscr{A}$—the set of FlyFast *actions*—$C, C_i \in \mathscr{S}$—the set of FlyFast *states*—and, for $i, j = 1, \ldots, r$ $a_i \neq a_j$ if $i \neq j$; note that $C_i = C_j$ with $i \neq j$ is allowed instead[6]. Each action has a probability assigned by means of an action probability function definition in A of the form $a :: exp$ where exp is an expression consisting of constants and $\mathsf{frc}(C)$ terms. Constants are floating point values or names associated to such values using the construct **const** $<$ `name` $>$ $=$ $<$ `value` $>$; $\mathsf{frc}(C)$ denotes the element associated to state C in the current occupancy measure vector[7]. So, strictly speaking, Δ and A characterise an inhomogeneous DTMC whose probability matrix $\mathbf{K}(\mathbf{m})$ is a function of the occupancy measure vector $\mathbf{m}$ such that for each pair of states C, C', the matrix element $\mathbf{K}(\mathbf{m})_{C,C'}$ is the probability of jumping from C to C' given the current occupancy measure vector $\mathbf{m}$. Letting $\mathscr{S}_\Delta$ be the set of states of the agent, with $|\mathscr{S}_\Delta| = S$, and $\mathscr{U}^S = \{(m_1, \ldots, m_S) | m_1 + \ldots + m_S = 1\}$ denote the unit simplex of dimension S, we have $\mathbf{K} : \mathscr{U}^S \times \mathscr{S}_\Delta \times \mathscr{S}_\Delta \to [0,1]$. Matrix $\mathbf{K}$ is generated directly from the input specification $\langle\Delta, A, \mathbf{C}_0\rangle^{(N)}$; the reader interested in the details of how to derive $\mathbf{K}$ is referred to [26, 28]. Auxiliary function definitions can be specified in A. The initial state $\mathbf{C}_0$ is a vector of size N consisting of the initial state of each individual object. Finally, note that in matrix $\mathbf{K}(\mathbf{m})$ the information on specific actions is lost, which is common in PCTL/DTMC based approaches; furthermore, we note that, by construction, $\mathbf{K}(\mathbf{m})$ does not depend on N (see [26, 28] for details).

3 A revised translation

As in [8], we define a translation such that, given a PiFF system specification $\Upsilon = (\Delta_\Upsilon, F_\Upsilon, \Sigma_0)^{(N)}$, the translation returns the FlyFast system specification $\langle\Delta, A, \mathbf{C}_0\rangle^{(N)}$ preserving probabilistic semantics. The predicate-based FlyFast front-end is then completed with a simple translation at the PCTL level, for which we refer to [8].

The system model specification translation consists of two phases. In the first phase, each action in the input system model specification Υ is annotated with an identifier which is unique within the specification. We let $\aleph(\Upsilon)$ denote the resulting specification. These annotations will make action names unique specification-wide thus eliminating complications which may arise from multiple occurrences of the same action, in particular when leading to the same state (see [8] for details). Of course, these annotations are disregarded in the probabilistic semantics, when considering the interaction model of components. In other words, an output action $\alpha\langle\rangle$ in outbox $(\gamma, \pi, \alpha\langle\rangle)$ must match with any input

[6] The concrete FlyFast syntax is: `state C{a_1.C_1 + a_2.C_2 ...a_r.C_r}`.

[7] The occupancy measure vector is a vector with as many elements as the number of states of an individual agent; the element associated to a specific state gives the fraction of the subpopulation currently in that state over the size of the overall population. The occupancy measure vector is a compact representation of the system global state.

action $\alpha()$ even if $\alpha\langle\rangle$ would actually correspond to $(\alpha,\iota)^*[\pi]\langle\rangle$ and $\alpha()$ would actually correspond to $(\alpha,\eta)^*[\pi']()$. Apart from this detail, the probabilistic semantics as defined in [8] remain unchanged.

The second phase is defined by the translation algorithm defined in Figure 5, which is a revised and simplified version of that presented in [8] and is applied to $\aleph(\Upsilon)$. We let $\mathscr{I}(\aleph(\Upsilon))$ denote the result of the translation, namely the pure FlyFast system specification $\langle\Delta,A,\mathbf{C}_0\rangle^{(N)}$.

We recall here some notation from [8]. We let $\mathscr{S}_{\Delta_\Upsilon}$ denote the set of states of Υ; Γ_{Δ_Υ} is the set of all stores defined over the attributes of Υ—a store is a finite mapping from the attributes of the component to a finite set of values $\mathscr{V}$, thus Γ_{Δ_Υ} is finite—and $\mathscr{O}_{\Delta_\Upsilon}$ the finite set of all outboxes of Υ. A Υ *component-state* is a triple $(C,\gamma,O)\in\mathscr{S}_{\Delta_\Upsilon}\times\Gamma_{\Delta_\Upsilon}\times\mathscr{O}_{\Delta_\Upsilon}=\Omega_{\Delta_\Upsilon}$. If the component-state is the target of a transition modelling the execution of an *output* action, then $O=(\gamma',\pi,\alpha\langle\rangle)$, where γ' is the store of the (component-state) source of the transition, π is the predicate used in the action—actualised with γ'—and $\alpha\langle\rangle$ the actual message sent by the action. If, instead, the component-state is the target of a transition for an *input* action, then $O=\langle\rangle$, i.e. the empty outbox. Note that the set of component states of $\aleph(\Upsilon)$ is identical to that of Υ. Also the set of all stores of $\aleph(\Upsilon)$ is the same as that of Υ. In the algorithm of Figure 5 by $t*t'$ we mean the *syntactical* term representing the product of terms t and t'; the notation is extended to $\mathsf{PROD}\{t|\mathsf{cond}(t)\}$, denoting the *syntactical* product $t_1*\ldots*t_n$ if $\{t|\mathsf{cond}(t)=\mathrm{tt}\}=\{t_1,\ldots,t_n\}\neq\emptyset$ and 1 otherwise. Similarly, $\mathsf{SUM}\{t|\mathsf{cond}(t)\}$ denotes the *syntactical* sum $t_1+\ldots+t_n$ if $\{t|\mathsf{cond}(t)=\mathrm{tt}\}=\{t_1,\ldots,t_n\}\neq\emptyset$ and 0 otherwise. The translation algorithm uses a few auxiliary functions which we briefly discuss below:

- $\mathscr{I}_{\mathscr{S}}:\Omega_{\Delta_\Upsilon}\to\mathscr{S}$ is a total injection which maps every component state of $\aleph(\Upsilon)$ to a distinct state of $\mathscr{I}(\aleph(\Upsilon))$; we recall that $\mathscr{S}$ denotes the set of state names of FlyFast models.
- $\mathscr{I}_{\mathscr{A}}:(\mathscr{S}_{\Delta_\Upsilon}\times\Gamma_{\Delta_\Upsilon})\times(\Lambda_{\Delta_\Upsilon}\times I_\aleph)\times\Omega_{\Delta_\Upsilon}\to\mathscr{A}$ is a total injection where, as in [8], $\Lambda_{\Delta_\Upsilon}$ is the set of action labels of Υ and $I_\aleph$ is the set of unique identifiers used in the first phase of the translation. We recall that $\mathscr{A}$ is the set of action names of FlyFast. The mapping of actions is a bit more delicate because we have to respect FlyFast static constraints and, in particular, we have to avoid multiple probability function definitions for the same action. A first source of potential violations (i.e. multiple syntactical occurrences of the same action) has been removed by action annotation in the first phase of the translation. A second source is the fact that the same action can take place in different contexts (for example with different stores) or leading to different target component states (maybe with different probabilities). To that purpose, we could distinguish different occurrences of the same action in different transitions, each characterised by its source component-state and its target component-state in Ω_{Δ_Υ}. In practice, since an action of a component cannot be influenced by the current outbox of the component, it is sufficient to restrict the first component of the domain from Ω_{Δ_Υ} to $(\mathscr{S}_{\Delta_\Upsilon}\times\Gamma_{\Delta_\Upsilon})$.
- The interpretation functions defined in Figure 3, namely those depending on stores only (and not on occupancy measure vectors); we assume $\mathbf{E_L}[\![\cdot]\!]_\gamma$ extended to $\mathbf{E_L}[\![fn]\!]_\gamma$ for defined function fn, in the standard way. In Figure 3 β_Υ denotes the constant to value bindings generated by the **const** construct in the input model specification Υ, whereas store update *upd* is defined as above.
- The translation function $\mathscr{I}_{\mathscr{P}}$ for transition probability expressions p_j, defined in Figure 4.

Output actions are dealt with in step 1 of the algorithm of Figure 5. Let us consider, for example, $(\mathtt{inf},1)^*[\bot]\langle\rangle\mathtt{Jump}$ in the definition of state `S` in Figure 2 (assuming annotations are integer values and the action has been annotated with 1). We know that the possible values for locations are `A,B,C,D`, so that the set of all stores is $\{\mathsf{loc}\}\to\{\mathtt{A},\mathtt{B},\mathtt{C},\mathtt{D}\}$. The algorithm generates 12 actions[8]. Let us focus

[8]Diagonal jumps are not contemplated in the model; technically this comes from the actual probability values used in the

$$
\begin{array}{lcl}
\mathbf{E_L}[\![\top]\!]_\gamma & = & \mathrm{tt} \\
\mathbf{E_L}[\![\bot]\!]_\gamma & = & \mathrm{ff} \\
\mathbf{E_L}[\![e_1 \bowtie e_2]\!]_\gamma & = & \mathbf{E_L}[\![e_1]\!]_\gamma \bowtie \mathbf{E_L}[\![e_2]\!]_\gamma \\
\mathbf{E_L}[\![\neg b]\!]_\gamma & = & \neg \mathbf{E_L}[\![b]\!]_\gamma \\
\mathbf{E_L}[\![b_1 \wedge b_2]\!]_\gamma & = & \mathbf{E_L}[\![b_1]\!]_\gamma \wedge \mathbf{E_L}[\![b_2]\!]_\gamma \\
\mathbf{E_L}[\![v_a]\!]_\gamma & = & v_a \\
\mathbf{E_L}[\![c_a]\!]_\gamma & = & \beta_\Upsilon(c_a) \\
\mathbf{E_L}[\![v_p]\!]_\gamma & = & v_p \\
\mathbf{E_L}[\![c_p]\!]_\gamma & = & \beta_\Upsilon(c_p) \\
\mathbf{E_L}[\![a]\!]_\gamma & = & a \\
\mathbf{E_L}[\![\mathbf{my}.a]\!]_\gamma & = & \gamma(a) \\
\mathbf{E_L}[\![fn_a(e_1,\ldots,e_m)]\!]_\gamma & = & \mathbf{E_L}[\![fn_a]\!]_\gamma(\mathbf{E_L}[\![e_1]\!]_\gamma,\ldots,\mathbf{E_L}[\![e_m]\!]_\gamma) \\
\mathbf{E_L}[\![fn_p(e_1,\ldots,e_m)]\!]_\gamma & = & \mathbf{E_L}[\![fn_p]\!]_\gamma(\mathbf{E_L}[\![e_1]\!]_\gamma,\ldots,\mathbf{E_L}[\![e_m]\!]_\gamma) \\
\mathbf{E_U}[\![upd]\!]_\gamma & = & \lambda\gamma'.\mathtt{dom}(\gamma') \neq \{a_1,\ldots,a_k\} \to 0; \\
 & & \quad \gamma'(a_1) = \mathbf{E_L}[\![e_{11}]\!]_\gamma \wedge \ \ldots\ \wedge \gamma'(a_k) = \mathbf{E_L}[\![e_{k1}]\!]_\gamma \to \mathbf{E_L}[\![p_1]\!]_\gamma; \\
 & & \quad \vdots \\
 & & \quad \gamma'(a_1) = \mathbf{E_L}[\![e_{1n}]\!]_\gamma \wedge \ \ldots\ \wedge \gamma'(a_k) = \mathbf{E_L}[\![e_{kn}]\!]_\gamma \to \mathbf{E_L}[\![p_n]\!]_\gamma; \\
 & & \quad \mathbf{otherwise} \ \to \ 0 \\
\mathbf{E_R}[\![\top]\!]_\gamma & = & \mathrm{tt} \\
\mathbf{E_R}[\![\bot]\!]_\gamma & = & \mathrm{ff} \\
\mathbf{E_R}[\![e_1 \bowtie e_2]\!]_\gamma & = & \mathbf{E_R}[\![e_1]\!]_\gamma \bowtie \mathbf{E_R}[\![e_2]\!]_\gamma \\
\mathbf{E_R}[\![\neg b]\!]_\gamma & = & \neg \mathbf{E_R}[\![b]\!]_\gamma \\
\mathbf{E_R}[\![b_1 \wedge b_2]\!]_\gamma & = & \mathbf{E_R}[\![b_1]\!]_\gamma \wedge \mathbf{E_R}[\![b_2]\!]_\gamma \\
\mathbf{E_R}[\![v_a]\!]_\gamma & = & v_a \\
\mathbf{E_R}[\![c_a]\!]_\gamma & = & \beta_\Upsilon(c_a) \\
\mathbf{E_R}[\![a]\!]_\gamma & = & \gamma(a)
\end{array}
$$

Figure 3: Interpretation functions relevant for the translation

on the action ξ associated to local position `A` (i.e. $\gamma = [\mathsf{loc} \mapsto \mathtt{A}]$) and possible next position B (i.e. $\gamma' = [\mathsf{loc} \mapsto B]$); the algorithm will generate the FlyFast probability function definition $\xi :: \mathtt{pW}(A) * (\mathsf{frc}(\mathtt{I1}) + \ldots + \mathsf{frc}(\mathtt{In}))$[9] as well as a transition leading to (a state which is the encoding, via $\mathscr{I}_{\mathscr{S}}$, of) the component state with `I` as (proper) state, store γ', and outbox $(\gamma, \bot, \mathtt{inf}\langle\rangle)$. Since the action is not depending on the current outbox, in practice a copy of such a transition is generated *for each* component state sharing the same proper state `S` and the same store γ. The translation scheme for input actions is defined in case 2 and is similar, except that one has also to consider the sum of the fractions of the possible partners. The translation of the **rest** case is straightforward. Note that for every $\zeta :: r * q \in A_\gamma$, r is a probability value associated to a store update; since any store update characterizes a probability distribution over stores, assuming the range of such a distribution is $\{r_1,\ldots,r_n\}$ if $\zeta_i :: r_i * q \in A_\gamma$, then also $\zeta_j :: r_j * q \in A_\gamma$ for all $j = 1,\ldots,n,\ j \neq i$ with $\sum_{i=1}^{n} r_j = 1$. Thus the remaining probability is $q_\gamma = (1 - \mathsf{SUM}\{q | \zeta :: r * q \in A_\gamma\})$, where q is either a term $\mathscr{I}_{\mathscr{P}}(p_j)_\gamma$, with p_j occurring in a summand of the state defining equation (see step 1), or a term $\mathscr{I}_{\mathscr{P}}(p_j)_\gamma * \mathsf{SUM}\{\mathsf{frc}(\mathscr{I}_{\mathscr{S}}(\Sigma)) | \ldots\}$ (see step 2). It worth

definition of `Jump`.

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

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

Figure 4: Transition probability expressions translation function definition

For each state equation $C := \sum_{j\in J}[g_j]p_j :: act_j.C_j$ in Δ_Υ:

1. For each *output* action $(\alpha,\iota)^*[\pi]\langle\rangle\sigma = act_k$ with $k \in J$ and $[g_k]p_k \neq \mathbf{rest}$,
   for each $\gamma \in \Gamma_{\Delta_\Upsilon}$ s.t. $\mathbf{E_L}[\![g_k]\!]_\gamma = \mathrm{tt}$ and $(C,\gamma,O) \in \Omega_{\Delta_\Upsilon}$ for some $O \in \mathscr{O}_{\Delta_\Upsilon}$, for each $\gamma' \in \Gamma_{\Delta_\Upsilon}$ s.t. $(C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)) \in \Omega_{\Delta_\Upsilon}$ and $\mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') > 0$, let $\xi = \mathscr{I}_{\mathscr{A}}((C,\gamma),(\alpha\langle\rangle,\iota),(C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)))$ be a fresh new action in the FlyFast model specification $\mathscr{I}(\aleph(\Upsilon)) = \langle\Delta,A,\mathbf{C}_0\rangle^{(N)}$ and add the following action probability function definition in A: $\xi :: \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') * \mathscr{I}_{\mathscr{P}}(p_k)_\gamma$.
   Moreover, for each outbox $O \in \mathscr{O}_{\Delta_\Upsilon}$ s.t. $(C,\gamma,O) \in \Omega_{\Delta_\Upsilon}$, the following summand is added to the equation in Δ for state $\mathscr{I}_{\mathscr{S}}((C,\gamma,O))$: $\xi.\mathscr{I}_{\mathscr{S}}((C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)))$;

2. For each *input* action $(\alpha,\iota)^*[\pi]()\sigma = act_k$, with $k \in J$ and $[g_k]p_k \neq \mathbf{rest}$,
   for each $\gamma \in \Gamma_{\Delta_\Upsilon}$ s.t. $\mathbf{E_L}[\![g_k]\!]_\gamma = \mathrm{tt}$ and $(C,\gamma,O) \in \Omega_{\Delta_\Upsilon}$ for some $O \in \mathscr{O}_{\Delta_\Upsilon}$, for each $\gamma' \in \Gamma_{\Delta_\Upsilon}$ s.t. $(C_k,\gamma',\langle\rangle) \in \Omega_{\Delta_\Upsilon}$ and $\mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') > 0$,
   let $\xi = \mathscr{I}_{\mathscr{A}}((C,\gamma),(\alpha(),\iota),(C_k,\gamma',\langle\rangle))$, be a fresh new action in the FlyFast model specification $\mathscr{I}(\aleph(\Upsilon)) = \langle\Delta,A,\mathbf{C}_0\rangle^{(N)}$ and add the following action probability function definition in A:
   $\xi :: \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') * \mathscr{I}_{\mathscr{P}}(p_k)_\gamma *$
   $\quad * \mathsf{SUM}\{\mathsf{frc}(\mathscr{I}_{\mathscr{S}}(\Sigma)) \mid \Sigma = (C'',\gamma'',(\overline{\gamma},\overline{\pi},\alpha\langle\rangle)) \in \Omega_{\Delta_\Upsilon} \ \wedge$
   $\quad \wedge\ \mathbf{E_R}[\![\overline{\pi}]\!]_\gamma = \mathbf{E_R}[\![\mathbf{E_L}[\![\pi]\!]_\gamma]\!]_{\overline{\gamma}} = \mathrm{tt}\}$.
   Moreover, for each outbox $O \in \mathscr{O}_{\Delta_\Upsilon}$ s.t. $(C,\gamma,O) \in \Omega_{\Delta_\Upsilon}$, the following summand is added to the equation in Δ for state $\mathscr{I}_{\mathscr{S}}((C,\gamma,O))$: $\xi.\mathscr{I}_{\mathscr{S}}((C_k,\gamma',\langle\rangle))$;

3. If there exists $k \in J$ s.t. $[g_k]p_k = \mathbf{rest}$, and $act_k = (\alpha,\iota)^*[\pi]\langle\rangle\sigma$, for each $\gamma \in \Gamma_{\Delta_\Upsilon}$ s.t. $(C,\gamma,O) \in \Omega_{\Delta_\Upsilon}$ for some $O \in \mathscr{O}_{\Delta_\Upsilon}$, let A_γ be the set of probability function definitions which has been constructed in steps (1) and (2) above. Let q_γ be defined by $q_\gamma = (1 - \mathsf{SUM}\{q \mid \zeta :: r * q \in A_\gamma\})$. For all $\gamma' \in \Gamma_{\Delta_\Upsilon}$ s.t. $(C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)) \in \Omega_{\Delta_\Upsilon}$, let $\xi = \mathscr{I}_{\mathscr{A}}((C,\gamma),(\alpha,\iota)\langle\rangle,(C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle))) \in \Omega_{\Delta_\Upsilon}$, be a fresh new action in the FlyFast model specification $\mathscr{I}(\aleph(\Upsilon)) = \langle\Delta,A,\mathbf{C}_0\rangle^{(N)}$ and add the following action probability function definition in A: $\xi :: \mathbf{E_U}[\![\sigma]\!]_\gamma(\gamma') * q_\gamma$.
   Moreover, for each outbox $O \in \mathscr{O}_{\Delta_\Upsilon}$ s.t. $(C,\gamma,O) \in \Omega_{\Delta_\Upsilon}$, the following summand is added to the equation in Δ for state $\mathscr{I}_{\mathscr{S}}((C,\gamma,O))$: $\xi.\mathscr{I}_{\mathscr{S}}((C_k,\gamma',(\gamma,\mathbf{E_L}[\![\pi]\!]_\gamma,\alpha\langle\rangle)))$;

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

Figure 5: The translation algorithm

pointing out here that the translation of Figure 5 is essentially the same as that presented in [8], when the latter is applied to the sublanguage of PiFF where one requires that each action occurs at most once. The annotations performed in the first phase of the translation ensure that this requirement is fulfilled; as we noted above, these annotations are purely syntactical and are disregarded at the semantics level. We also recall that in probabilistic, pure DTMC process language semantics, actions are in the end dropped and, for each pair of states, the cumulative probability of such actions is assigned to the single transition from one of the states to the other one. Consequently, correctness of the translation, proved in [8], is preserved by the simplified version presented in this paper.

We note that in the algorithm sets Ω_{Δ_Υ}, Γ_{Δ_Υ} and $\mathscr{O}_{\Delta_\Upsilon}$ are used. Of course, an alternative approach could be one which considers only the set $\overline{\Omega_{\Delta_\Upsilon}}$ of component states which are *reachable* from a given initial component state and, consequently, the sets $\overline{\Gamma_{\Delta_\Upsilon}}$ and $\overline{\mathscr{O}_{\Delta_\Upsilon}}$ of *used* stores and outboxes. In this way, the size of the resulting FlyFast model specification would be smaller (for example in terms of number of

states). On the other hand, this approach might require recompilation for each model-checking session starting from a different initial component state.

4 A simplified language for Bisimulation-based optimisation

In this section we consider a simplified language for transition probability expressions appearing in state defining equations that will allow us to perform bisimulation based optimisation of the result $\langle \Delta, A, \mathbf{C}_0 \rangle^{(N)}$. The restricted syntax for transition probability expressions p we use in this section is the following: $p ::= e_p \,|\, e_p \cdot \mathsf{frc}(C) \,|\, e_p \cdot \mathsf{frc}(\pi)$ and $e_p ::= v_p \,|\, c_p$ where v_p and c_p and π are defined as in Section 2.

By inspection of the FlyFast translation as defined in Section 3, and recalling that the set $\mathscr{S}_\Delta$ of the states of the resulting FlyFast model, ranged over by $z, z_i, \ldots$, has cardinality S, it is easy to see that the probability action definition in the result of a translation of a generic *output* action is either of the form $\xi :: k$, or it is of the form $\xi :: k * \mathsf{SUM}\{\mathsf{frc}(z_i) | i \in I\}$ where k is a FlyFast constant. Moreover, if p was of the form $e_p \cdot \mathsf{frc}(C)$, then index set $I \subseteq \{1, \ldots, S\}$ identifies those states in $\mathscr{S}_\Delta$ that represent (via $\mathscr{I}_\mathscr{S}$) component states with proper local state C; if instead, p was of the form $e_p \cdot \mathsf{frc}(\pi)$, then $I \subseteq \{1, \ldots, S\}$ identifies those states in $\mathscr{S}_\Delta$ that represent (via $\mathscr{I}_\mathscr{S}$) component states with a store satisfying π in the relevant store. At the FlyFast semantics level, recalling that $\mathsf{frc}(z_i)$ is exactly the i-th component m_i of the occupancy measure vector $\mathbf{m} = (m_1, \ldots, m_S)$ of the model, we can rewrite[10] the above as k or $k \cdot \sum_{i \in I} m_i$.

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

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

An immediate consequence of using the above mentioned restricted syntax for the probability function definitions is that, letting $\mathbf{K} : \mathscr{U}^S \times \mathscr{S} \times \mathscr{S} \to [0,1]$ be the transition probability matrix for the FlyFast translation of a model specification, we have that $\mathbf{K}(m_1, \ldots, m_S)_{z,z'}$ is a polynomial function of degree at most 2 in variables $m_1, \ldots, m_S$.

5 Bisimilarity and State-space Reduction

The following definition generalises standard probabilistic bisimilarity for state labelled DTMCs to the case in which transition probabilities are *functions* instead of constant values.

Definition 1 *For finite set of states $\mathscr{S}$, with $|\mathscr{S}| = S$, let $\mathbf{K} : \mathscr{U}^S \times \mathscr{S} \times \mathscr{S} \to [0,1]$ and, for $z \in \mathscr{S}$ and $Q \subseteq \mathscr{S}$, write $\mathbf{K}(\mathbf{m})_{z,Q}$ for $\sum_{z' \in Q} \mathbf{K}(\mathbf{m})_{z,z'}$. Let furthermore $\mathscr{L} : \mathscr{S} \to 2^{AP}$ be a state-labelling function, for a given set AP of atomic propositions. An equivalence relation $R \subseteq \mathscr{S} \times \mathscr{S}$ is called a* bisimulation relation *if and only if $z_1 \, R \, z_2$ implies: (i) $\mathscr{L}(z_1) = \mathscr{L}(z_2)$ and (ii) $\mathbf{K}(\mathbf{m})_{z_1,Q} = \mathbf{K}(\mathbf{m})_{z_2,Q}$, for all $\mathbf{m} \in \mathscr{U}^S$ and $Q \in \mathscr{S}/R$. The* bisimulation equivalence *on $\mathscr{S}$ is the largest bisimulation relation $R \subseteq \mathscr{S} \times \mathscr{S}$.*

We point out that the notion of bisimilarity does *not* introduce any approximation, and consequently error, in a model and related analyses. Bisimilarity is only a way for abstracting from details that are irrelevant

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

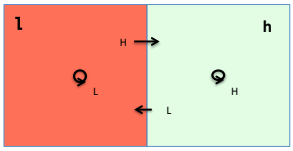


Figure 6: *SI* in two quadrants

for the specific analyses of interest. In particular, it is useful to remark that bisimilarity preserves also state labels, which are directly related to the atomic propositions of logic formulas for which model-checking is performed. Actually, it is well known that probabilistic bisimilarity coincides with PCTL equivalence, i.e. the equivalence induced on system states by their satisfaction of PCTL formulas, for finitely branching systems [3].

Note that $\mathbf{K}(m_1,\ldots,m_S)_{z_1,Q} = \mathbf{K}(m_1,\ldots,m_S)_{z_2,Q}$ for all $(m_1,\ldots,m_S) \in \mathscr{U}^S$ is in general not decidable. If instead we consider only transition probability matrices as in Section 4, we see that each side of the above equality is a polynomial function of degree at most 2 in variables $m_1,\ldots,m_S$ and one can define a normal form for the polynomial expressions in $m_1,\ldots,m_S$ supported by an ordering relation on the variable names (e.g. $m_1 \prec \ldots \prec m_S$) and get expressions of the general form $\left(\sum_{i=1}^{S}\sum_{j\geq i}^{S} h_{ij}\cdot m_i\cdot m_j\right) + \left(\sum_{i=1}^{S} h_i\cdot m_i\right) + h$ for suitable h_{ij},h_i,h. Actually, such expressions can always be rewritten in the form $\left(\sum_{i=1}^{S}\sum_{j\geq i}^{S} u_{ij}\cdot m_i\cdot m_j\right) + u$ for suitable u_{ij},u since, recalling that $\sum_{i=1}^{S} m_i = 1$, we get $\sum_{i=1}^{S} h_i\cdot m_i = \left(\sum_{i=1}^{S} m_i\right)\cdot\left(\sum_{i=1}^{S} h_i\cdot m_i\right)$ which, by simple algebraic manipulation, yields an expression of the following form: $\left(\sum_{i=1}^{S}\sum_{j\geq i}^{S} u'_{ij}\cdot m_i\cdot m_j\right)$; finally, we get $\left(\sum_{i=1}^{S}\sum_{j\geq i}^{S} u_{ij}\cdot m_i\cdot m_j\right) + u$ by letting $u_{ij} = h_{ij} + u'_{ij}$ and $u = h$. The following proposition thus establishes decidability of $\mathbf{K}(m_1,\ldots,m_S)_{z_1,Q} = \mathbf{K}(m_1,\ldots,m_S)_{z_2,Q}$ for all $(m_1,\ldots,m_S) \in \mathscr{U}^S$ for transition probability matrices as in Section 4:

Proposition 1
Let $A(m_1,\ldots,m_S) = \left(\sum_{i=1}^{S}\sum_{j\geq i}^{S} a_{ij}\cdot m_i\cdot m_j\right) + a$ *and* $B(m_1,\ldots,m_S) = \left(\sum_{i=1}^{S}\sum_{j\geq i}^{S} b_{ij}\cdot m_i\cdot m_j\right) + b$ *with* $a_{ij},b_{ij},a,b \in \mathbb{R}$, *where* $m_1,\ldots,m_S$ *are variables taking values over* $\mathbb{R}_{\geq 0}$ *with* $\sum_{i=1}^{S} m_i = 1$. *The following holds:* $(\forall m_1,\ldots,m_S.A(m_1,\ldots,m_S) = B(m_1,\ldots,m_S)) \Leftrightarrow ((\forall i,j = 1,\ldots,S \text{ with } i \leq j.a_{ij} = b_{ij}) \wedge a = b)$.

The above results can be used for reduction of the state-space of the individual agent, i.e. the resulting FlyFast model specification, after the application of the translation described in Section 3, by using for instance the standard probabilistic relational coarsest set partition problem algorithm (see e.g. [21], page 227) with slight obvious modifications due to the presence of state-labels and the need of symbolic computation capabilities required for checking (degree 2) polynomial expressions equality.[11] It is worth mentioning that state aggregation via bisimilarity is effective only if there is some sort of compatibility between (i) state labelling—and, consequently, the specific PCTL atomic propositions one uses—and (ii) the way probabilities are assigned to transitions—and, consequently, the cumulative probabilities to equivalence classes. We will come back on this issue in the following section.

	SA	*SB*	*SC*	*SD*	*IA*	*IB*	*IC*	*ID*
SA	$H\phi_S(\mathbf{m})$	$\frac{L}{2}\phi_S(\mathbf{m})$	0	$\frac{L}{2}\phi_S(\mathbf{m})$	$H\phi_I(\mathbf{m})$	$\frac{L}{2}\phi_I(\mathbf{m})$	0	$\frac{L}{2}\phi_I(\mathbf{m})$
SB	$\frac{H}{2}\phi_S(\mathbf{m})$	$L\phi_S(\mathbf{m})$	$\frac{H}{2}\phi_S(\mathbf{m})$	0	$\frac{H}{2}\phi_I(\mathbf{m})$	$L\phi_I(\mathbf{m})$	$\frac{H}{2}\phi_I(\mathbf{m})$	0
SC	0	$\frac{L}{2}\phi_S(\mathbf{m})$	$H\phi_S(\mathbf{m})$	$\frac{L}{2}\phi_S(\mathbf{m})$	0	$\frac{L}{2}\phi_I(\mathbf{m})$	$H\phi_I(\mathbf{m})$	$\frac{L}{2}\phi_I(\mathbf{m})$
SD	$\frac{H}{2}\phi_S(\mathbf{m})$	0	$\frac{H}{2}\phi_S(\mathbf{m})$	$L\phi_S(\mathbf{m})$	$\frac{H}{2}\phi_I(\mathbf{m})$	0	$\frac{H}{2}\phi_I(\mathbf{m})$	$L\phi_I(\mathbf{m})$
IA	Hir	$\frac{L}{2}ir$	0	$\frac{L}{2}ir$	Hii	$\frac{L}{2}ii$	0	$\frac{L}{2}ii$
IB	$\frac{H}{2}ir$	Lir	$\frac{H}{2}ir$	0	$\frac{H}{2}ii$	Lii	$\frac{H}{2}ii$	0
IC	0	$\frac{L}{2}ir$	Hir	$\frac{L}{2}ir$	0	$\frac{L}{2}ii$	Hii	$\frac{L}{2}ii$
ID	$\frac{H}{2}ir$	0	$\frac{H}{2}ir$	Lir	$\frac{H}{2}ii$	0	$\frac{H}{2}ii$	Lii

Figure 7: IDTMC transition probability matrix $\mathbf{K}(\mathbf{m})$, for $\mathbf{m}$ in $\mathscr{U}^8$.

6 Example

The application of the translation to the specification of Example 1 generates an agent model with 8 states, say $\mathscr{S}_\Delta = \{SA, SB, SC, SD, IA, IB, IC, ID\}$[12] with associated IDTMC probability transition matrix as shown in Figure 7 where m_{xy} represents the fraction of objects currently in state xy for $x \in \{S, I\}$ and $y \in \{A, B, C, D\}$—i.e. the components in state x and with $\mathsf{loc} = y$ in the original specification of Fig 2—so that $\mathbf{m} = (m_{SA}, m_{SB}, m_{SC}, m_{SD}, m_{IA}, m_{IB}, m_{IC}, m_{ID})$ is the occupancy measure vector. In Figure 7, functions ϕ_S and ϕ_I are used as abbreviations in the obvious way: $\phi_S(\mathbf{m}) = m_{SA} + m_{SB} + m_{SC} + m_{SD}$ and $\phi_I(\mathbf{m}) = m_{IA} + m_{IB} + m_{IC} + m_{ID}$. Let us assume now that we are interested in checking PCTL formulas on the model of Fig 2 which distinguish components located in A or C from those located in B or D, and those in state S from those in state I, that is we consider atomic propositions Sh, Ih, Sl and Il and a labelling $\mathscr{L}$ such that $\mathscr{L}(SA) = \mathscr{L}(SC) = \{Sh\}$, $\mathscr{L}(IA) = \mathscr{L}(IC) = \{Ih\}$, $\mathscr{L}(SB) = \mathscr{L}(SD) = \{Sl\}$, and $\mathscr{L}(IB) = \mathscr{L}(ID) = \{Il\}$.

Consider relation R on $\mathscr{S}_\Delta$ defined as $R = I_{\mathscr{S}_\Delta} \cup \{(SA, SC), (SB, SD), (IA, IC), (IB, ID)\} \cup \{(SC, SA), (SD, SB), (IC, IA), (ID, IB)\}$ where $I_{\mathscr{S}_\Delta}$ is the identity relation on $\mathscr{S}_\Delta$. It is very easy to show that R is a bisimulation according to Definition 1. Clearly R is an equivalence relation and its quotient $\mathscr{S}_\Delta/R$ is the set $\{Q_{Sh}, Q_{Sl}, Q_{Ih}, Q_{Il}\}$ with $Q_{Sh} = \{SA, SC\}, Q_{Sl} = \{SB, SD\}, Q_{Ih} = \{IA, IC\}, Q_{Il} = \{IB, ID\}$. In addition, for all $z_1, z_2 \in \mathscr{S}_\Delta$, whenever $z_1 R z_2$, we have $\mathscr{L}(z_1) = \mathscr{L}(z_2)$ and $\mathbf{K}(\mathbf{m})_{z_1,Q} = \mathbf{K}(\mathbf{m})_{z_2,Q}$ for all $Q \in \mathscr{S}_\Delta/R$ and for all $\mathbf{m}$, as one can easily check; clearly, R is also the largest bisimulation relation on $\mathscr{S}_\Delta$. The relationship between the two occupancy measure vectors is: $m_{Q_{Sh}} = m_{SA} + m_{SC}$, $m_{Q_{Sl}} = m_{SB} + m_{SD}$, $m_{Q_{Ih}} = m_{IA} + m_{IC}$, and $m_{Q_{Il}} = m_{IB} + m_{ID}$. We can thus use the reduced IDTMC defined by matrix $\widehat{\mathbf{K}}(m_{Q_{Sh}}, m_{Q_{Sl}}, m_{Q_{Ih}}, m_{Q_{Il}})$ shown in Figure 9. It corresponds to the FlyFast agent specification $\widehat{\Delta}$ given in Figure 8. In a sense, the high probability locations A and C, in the new model, have collapsed into a single one, namely h and the low probability ones (B and D) have collapsed into l, as shown in Figure 6. We point out again the correspondence between the symmetry in the space jump probability on one side and the definition of the state labelling function on the other side. Finally, note that a coarser labelling like, e.g. $\mathscr{L}'(SA) = \mathscr{L}'(SC) = \mathscr{L}'(IA) = \mathscr{L}'(IC) = \{h\}$, $\mathscr{L}'(SB) = \mathscr{L}'(SD) = \mathscr{L}'(IB) = \mathscr{L}'(ID) = \{l\}$ would make the model collapse into one with only two states, Q_h and Q_l, with probabilities $H : Q_h \to Q_h, H : Q_l \to Q_h$, $L : Q_h \to Q_l$ and $L : Q_h \to Q_l$ where only the location would be modelled whereas

[11] For instance, on page 227 of [21], line 12, $v(x,S) = v(y,S)$ should be replaced with $L(x) = L(y) \wedge v(x,S) = v(y,S)$ and in line 13, $v(x,S) \neq v(y,S)$ should be replaced with $L(x) \neq L(y) \vee v(x,S) \neq v(y,S)$, in order to take state labels into consideration as well. Of course $v(x,S)$ (L, respectively)is to be intended as $\mathbf{K}(\mathbf{m})_{z,Q}$ ($\mathscr{L}$, respectively), using the notation we introduced above for Bisimilarity.

[12] Actually the agent resulting from the translation of Figure 5 has a higher number of states due to the different possibilities for outbox values. Many of these states are unreachable from the initial state since the agent has no input action and we assume an initial unreachable state pruning has been performed.

```
action QSh_inf_QIh: H*(frc(QIh)+frc(QIl));        action QSh_nsc_QSh: H*(frc(QSh)+frc(QSl));
action QSh_inf_QIl: L*(frc(QIh)+frc(QIl));        action QSh_nsc_QSl: L*(frc(QSh)+frc(QSl));
action QSl_inf_QIh: H*(frc(QIh)+frc(QIl));        action QSl_nsc_QSh: H*(frc(QSh)+frc(QSl));
action QSl_inf_QIl: L*(frc(QIh)+frc(QIl));        action QSl_nsc_QSl: L*(frc(QSh)+frc(QSl));

action QIh_inf_QIh: H*ii;                         action QIh_rec_QSh: H*ir;
action QIh_inf_QIl: L*ii;                         action QIh_rec_QSl: L*ir;
action QIl_inf_QIh: H*ii;                         action QIl_rec_QSh: H*ir;
action QIl_inf_QIl: L*ii;                         action QIl_rec_QSl: L*ir;

state QSh{QSh_inf_QIh.QIh + QSh_inf_QIl.QIl + QSh_nsc_QSh.QSh + QSh_nsc_QSl.QSl}
state QSl{QSl_inf_QIh.QIh + QSl_inf_QIl.QIl + QSl_nsc_QSh.QSh + QSl_nsc_QSl.QSl}
state QIh{QIh_inf_QIh.QIh + QIh_inf_QIl.QIl + QIh_rec_QSh.QSh + QIh_rec_QSl.QSl}
state QIl{QIl_inf_QIh.QIh + QIl_inf_QIl.QIl + QIl_rec_QSh.QSh +QIl_rec_QSl.QSl}
```

Figure 8: Reduced agent specification $\widehat{\Delta}$

	Q_{Sh}	Q_{Sl}	Q_{Ih}	Q_{Il}
Q_{Sh}	$H\cdot(m_{Q_{Sh}}+m_{Q_{Sl}})$	$L\cdot(m_{Q_{Sh}}+m_{Q_{Sl}})$	$H\cdot(m_{Q_{Ih}}+m_{Q_{Il}})$	$L\cdot(m_{Q_{Ih}}+m_{Q_{Il}})$
Q_{Sl}	$H\cdot(m_{Q_{Sh}}+m_{Q_{Sl}})$	$L\cdot(m_{Q_{Sh}}+m_{Q_{Sl}})$	$H\cdot(m_{Q_{Ih}}+m_{Q_{Il}})$	$L\cdot(m_{Q_{Ih}}+m_{Q_{Il}})$
Q_{Ih}	$H\cdot ir$	$L\cdot ir$	$H\cdot ii$	$L\cdot ii$
Q_{Il}	$H\cdot ir$	$L\cdot ir$	$H\cdot ii$	$L\cdot ii$

Figure 9: IDTMC transition probability matrix function $\widehat{\mathbf{K}}(\mathbf{m})$, for $\mathbf{m}$ in $\mathscr{U}^4$.

information on the infection status would be lost.

7 Conclusions

PiFF [8] is a language for a predicate-based front-end of FlyFast, an on-the-fly mean-field model-checking tool. In this paper we presented a simplified version of the translation proposed in [8] together with an approach for model reduction that can be applied to the result of the translation. The approach is based on probabilistic bisimilarity for inhomogeneous DTMCs. An example of application of the procedure has been shown. The implementation of a compiler for PiFF mapping the language to FlyFast is under development as an add-on of FlyFast. We plan to apply the resulting extended tool to more as well as more complex models, in order to get concrete insights on the practical applicability of the framework and on the actual limitations imposed by the restrictions necessary for exploiting bisimilarity-based state-space reduction. Investigating possible ways of relaxing some of such restrictions will also be an interesting line of research.

Acknowledgments Research partially funded by EU Project n. 600708 *A Quantitative Approach to Management and Design of Collective and Adaptive Behaviours* (QUANTICOL).

References

[1] A. Aziz, K. Sanwal, V. Singhal & R. Brayton (2000): *Model checking Continuous Time Markov Chains*. *ACM Transactions on Computational Logic* 1(1), pp. 162–170.

[2] C. Baier, B. Haverkort, H. Hermanns & J.-P. Katoen (2003): *Model-Checking Algorithms for Continuous-Time Markov Chains*. *IEEE Transactions on Software Engineering. IEEE CS* 29(6), pp. 524–541.

[3] Christel Baier & Joost-Pieter Katoen (2008): *Principles of model checking*. MIT Press.

[4] Girish Bhat, Rance Cleaveland & Orna Grumberg (1995): *Efficient On-the-Fly Model Checking for CTL**. In: *LICS*, IEEE Computer Society, pp. 388–397. Available at `http://doi.ieeecomputersociety.org/10.1109/LICS.1995.523273`.

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

[6] Luca Bortolussi & Jane Hillston (2012): *Fluid Model Checking*. In M. Koutny & I. Ulidowski, editors: *CONCUR*, *LNCS* 7454, Springer-Verlag, pp. 333–347. Available at `http://dx.doi.org/10.1007/978-3-642-32940-1_24`.

[7] Luca Bortolussi & Jane Hillston (2015): *Model checking single agent behaviours by fluid approximation*. *Inf. Comput.* 242, pp. 183–226, doi:10.1016/j.ic.2015.03.002.

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

[9] C. Courcoubetis, M. Vardi, P. Wolper & M. Yannakakis (1992): *Memory-efficient algorithms for the verification of temporal properties*. *Form. Methods Syst. Des.* 1(2-3), pp. 275–288.

[10] R.W.R. Darling & J.R. Norris (2008): *Differential equation approximations for Markov chains*. *Probability Surveys* 5, pp. 37–79, doi:10.1214/07-PS121.

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

[12] G. Della Penna, B. Intrigila, I. Melatti, E. Tronci & M. Venturini Zilli (2004): *Bounded Probabilistic Model Checking with the Muralpha Verifier*. In A. J. Hu & A. K. Martin, editors: *FMCAD 2004*, *LNCS* 3312, Springer, pp. 214–229.

[13] C. Feng & J. Hillston (2014): *PALOMA: A Process Algebra for Located Markovian Agents*. In G. Norman & W. Sanders, editors: *QEST 2014*, *LNCS* 8657, Springer-Verlag, pp. 266–280.

[14] Nicolas Gast & Bruno Gaujal (2010): *A mean field model of work stealing in large-scale systems*. In Vishal Misra, Paul Barford & Mark S. Squillante, editors: *SIGMETRICS*, ACM, pp. 13–24. Available at `http://doi.acm.org/10.1145/1811039.1811042`.

[15] Stefania Gnesi & Franco Mazzanti (2011): *An Abstract, on the Fly Framework for the Verification of Service-Oriented Systems*. In Martin Wirsing & Matthias M. Hölzl, editors: *Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing*, *Lecture Notes in Computer Science* 6582, Springer, pp. 390–407, doi:10.1007/978-3-642-20401-2_18.

[16] G. Guirado, T. Hérault, R. Lassaigne & S. Peyronnet (2006): *Distribution, Approximation and Probabilistic Model Checking*. In: *PDMC 2005. LNCS, vol. 135*, Springer, pp. 19–30.

[17] E. M. Hahn, H. Hermanns, B. Wachter & L. Zhang (2009): *INFAMY: An Infinite-State Markov Model Checker*. In: *CAV09, LNCS, vol. 5643*, Springer, pp. 641–64.

[18] H. Hansson & B. Jonsson (1994): *A Logic for Reasoning about Time and Reliability*. *Formal Aspects of Computing. The International Journal of Formal Methods. Springer-Verlag* 6(5), pp. 512–535.

[19] T. Hérault, R. Lassaigne, F. Magniette & S. Peyronnet (2004): *Approximate Probabilistic Model Checking*. In: *VMCAI04. LNCS, vol. 2937*, Springer, pp. 73–84.

[20] Gerard J. Holzmann (2004): *The SPIN Model Checker - primer and reference manual*. Addison-Wesley.

[21] D. Huynh & L. Tian (1992): *On some equivalence relations for probabilistic processes*. Fundamenta Informaticae 17, pp. 211–234.

[22] A. Kolesnichenko, A. Remke & P.-T. de Boer (2012): *A logic for model-checking of mean-field models*. Technical Report TR-CTIT-12-11, http://doc.utwente.nl/80267/.

[23] A. Kolesnichenko, A. Remke & P.-T. de Boer (2013): *A logic for model-checking of mean-field models*. In: *DSN13*.

[24] Kim G. Larsen & Axel Legay (2016): *Statistical Model Checking: Past, Present, and Future*. In Tiziana Margaria & Bernhard Steffen, editors: *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I*, *Lecture Notes in Computer Science* 9952, pp. 3–15, doi:10.1007/978-3-319-47166-2_1.

[25] D. Latella (1983): *Comunicazione basata su proprietà nei sistemi decentralizzati*. [Property-based interprocess communication in decentralized systems] Graduation Thesis. Istituto di Scienze dell'Informazione. Univ. of Pisa, Italy (in italian).

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

[27] D. Latella, M. Loreti & M. Massink (2014): *On-the-fly Probabilistic Model Checking*. In I. Lanese & A. Sokolova, editors: *Proceedings of the 7th Interaction and Concurrency Experience (ICE 2014), June 6, 2014, Berlin, Germany*, *EPTCS, ISSN: 2075-2180,* `http://cgi.cse.unsw.edu.au/ rvg/eptcs/` 166, pp. 45–59. ISSN: 2075-2180, DOI:10.4204/EPTCS.166.6 .

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

[29] Jean-Yves Le Boudec, David McDonald & Jochen Mundinger (2007): *A Generic Mean Field Convergence Result for Systems of Interacting Objects*. In: *QEST07*, IEEE Computer Society Press, pp. 3–18. Available at `http://doi.ieeecomputersociety.org/10.1109/QEST.2007.3`. ISBN 978-0-7695-2883-0.

[30] M. Loreti & J. Hillston (2016): *Modelling and Analysis of Collective Adaptive Systems with CARMA and its Tools*. In M. Bernardo, R. De Nicola & J. Hillston, editors: *Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems*, chapter 4, *LNCS* 9700, Springer-Verlag, pp. 83–119. DOI: 10.1007/978-3-319-34096-8_4, ISBN 978-3-319-34095-1 (print), 978-3-319-34096-8 (online), ISSN 0302-9743.

A Appendix

Proof of Proposition 1
$\Leftarrow$: Trivial.
$\Rightarrow$:
We first prove that $a = b$:

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

$\Rightarrow$ $\{$Logic$\}$

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

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

$a = b$

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

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

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

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

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

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

$\Rightarrow$ $\{a = b$ (see above)$\}$

$a_{ii} = b_{ii}$

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

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

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

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

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

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

$\Rightarrow$ $\{a = b$ (see above)$\}$

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

$\Rightarrow$ $\{$Algebra$\}$

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

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

$a_{ij} = b_{ij}$ •

Bridging static and dynamic program analysis using fuzzy logic

Jacob Lidman & Josef Svenningsson
Chalmers University of Technology
{lidman, josefs}@chalmers.se

Static program analysis is used to summarize properties over all dynamic executions. In a unifying approach based on 3-valued logic properties are either assigned a definite value or unknown. But in summarizing a set of executions, a property is more accurately represented as being biased towards true, or towards false. Compilers use program analysis to determine benefit of an optimization. Since benefit (e.g., performance) is justified based on the common case understanding bias is essential in guiding the compiler. Furthermore, successful optimization also relies on understanding the quality of the information, i.e. the plausibility of the bias. If the quality of the static information is too low to form a decision we would like a mechanism that improves dynamically.

We consider the problem of building such a reasoning framework and present the fuzzy data-flow analysis. Our approach generalize previous work that use 3-valued logic. We derive fuzzy extensions of data-flow analyses used by the lazy code motion optimization and unveil opportunities previous work would not detect due to limited expressiveness. Furthermore we show how the results of our analysis can be used in an adaptive classifier that improve as the application executes.

1 Introduction

How can one reconcile static and dynamic program analysis? These two forms of analysis complement each other: static analysis summarizes all possible runs of a program and thus provide soundness guarantees, while dynamic analysis provides information about the particular runs of a program which actually happen in practice and can therefore provide more relevant information. Being able to combine these two paradigms has applications on many forms on analyses, such as alias analysis [16, 21] and dependence analysis [18].

Compilers use program analysis frameworks to prove *legality* as well as determining *benefit* of transformations. Specifications for legality are composed of *safety* and *liveness* assertions (i.e. universal and existentially quantified properties), while specifications for benefit use assertions that hold in the *common case*. This reason for adopting the common case is that few transformations improve performance in general (i.e., for every input, environment). Similarly most transformations could potentially improve performance in a least one case. As such, compiler optimizations are instead motivated based on (an approximation of) the majority case, i.e. the (weighted) mean. While determining legality has improved due to advances in the verification community the progress in establishing benefit has been slow.

In this paper we introduce fuzzy data-flow analysis, a framework for static program analysis based on fuzzy logic. The salient feature of our framework is that it can naturally incorporate dynamic information while still being a static analysis. This ability comes thanks to a shift from "crisp" sets where membership is binary, as employed in conventional static analysis, to fuzzy sets where membership is gradual.

Submitted to:
QAPL 2017

We make the following contributions:

- Section 3 introduces our main contribution, the fuzzy data-flow framework.
- Section 4 demonstrates the benefit of our framework by presenting a generalization of a well-known code motion algorithm and we show how this generalization provides new opportunities for optimizations previous approaches would not discover.
- Section 4 shows how fuzzy logic can benefit program analysis by (1) using second-order fuzzy sets to separate uncertainty in data-flow and control-flow and hence improve an inter-procedural analysis and (2) using fuzzy regulators to refine the results of our static analysis, hence improving the precision dynamically.

2 Preliminaries

We introduce and define fuzzy sets and the operators that form fuzzy logic. These concepts will be used in Section 3 to define the transfer functions of our data-flow analysis.

2.1 Fuzzy set

Elements of a crisp set[1] are either members or non-members w.r.t to a universe of discourse. A fuzzy set (FS) instead allow partial membership denoted by a number from the unit interval $[0,1]$. The membership degree typically denotes vagueness. The process to convert crisp membership to fuzzy grades is called *fuzzification* and the inverse is called *defuzzification*. Following Dubois et al. [9, 8] let S be a crisp set and $\mu : S \mapsto [0,1]$ a *membership function* (MF) then $\langle S, \mu \rangle$ is a fuzzy set. As a convention, if S is understood from context we sometimes refer to μ as a fuzzy set. The membership function formalizes the fuzzification. Fuzzy sets are ordered point-wise, i.e. $(S, \mu_A) \leq (S, \mu_B) \Leftrightarrow \forall s \in S\colon \mu_A(s) \leq \mu_B(s)$.

We can accommodate some notion about uncertainty of vagueness by considering a type-2 fuzzy set where the membership degree itself is a fuzzy set. Type-2 FS (T2FS) membership functions are composed of a primary (J_s) and secondary (μ) membership $\{\langle (s,u), \mu(s,u) \rangle \mid s \in S, u \in J_s \subseteq [0,1]\}$. Here uncertainty is represented by the secondary membership that define the possibility of the primary membership. When for each x and u, it holds $\mu(x,u) = 1$ the T2FS is called an *interval* T2FS. Gehrke et al. [10] showed that this can equivalently be described as an interval valued fuzzy sets (IVFS) where $\mu\colon S \to \{[l,u] \mid \bot \leq l \leq u \leq \top\}$. IVFS are a special case of lattice valued fuzzy sets (L-fuzzy sets) where the membership domain forms a lattice over $[0,1]$. Defuzzification of T2FS often proceeds in two phases. The first phase applies *type reduction* to transform the T2FS to a type-1 FS (T1FS). The second phase then applies a type-1 defuzzification.

2.2 Fuzzy logic

Fuzzy logic defines many-valued formal systems to reason about truth in the presence of vagueness. Contrary to classical logic the law of excluded middle ($p \vee \neg p = \top$) and the law of non-contradiction ($p \wedge \neg p = \bot$) does not, in general, hold for these systems. Fuzzy logic uses T-, S- and C- norms to generalize the logical operators $\wedge$, $\vee$ and $\neg$. We compactly represent a fuzzy logic by $\langle \tilde{\wedge}, \tilde{\vee}, \tilde{\neg} \rangle$[2] which is sometimes called a *De Morgan system* [9] because it satisfies a generalization of De Morgans laws: $\tilde{\neg}(P \tilde{\wedge} Q) \Leftrightarrow \tilde{\neg} P \tilde{\vee} \tilde{\neg} Q$ and $\tilde{\neg}(P \tilde{\vee} Q) \Leftrightarrow \tilde{\neg} P \tilde{\wedge} \tilde{\neg} Q$.

[1] In the context of fuzzy logic, crisp or Boolean set refer to a classical set to avoid confusion with fuzzy sets.

[2] Although one would expect the definition of a fuzzy logic to include a "fuzzy implication" operator in this work we do not consider it.

	Fuzzy logic	T-norm	S-norm	C-norm
1	*Min-Max*	$\min(x,y)$	$\max(x,y)$	$1-x$
2	*Algebraic Sum-product*	xy	$x+y-xy$	$1-x$
3	*Lukasiewicz*	$max(x+y-1,0)$	$min(x+y,1)$	$1-x$
4	*Nilpotent*	$\begin{cases} min(x,y) & x+y>1 \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} max(x,y) & x+y<1 \\ 1 & \text{otherwise} \end{cases}$	$1-x$

Table 1: Common instantiations of fuzzy logics

Definition 1. *Let U be a binary function $[0,1]^2 \to [0,1]$ that is commutative, associative and increasing and has an identity element $e \in [0,1]$. If $e=1$ then U is a **Triangular norm (T-norm)** and if $e=0$ then U is a **Triangular conorm (S-norm)**[3].*

Definition 2. *A **C-norm** is a unary function $n\colon [0,1] \to [0,1]$ that is decreasing, involutory (i.e., $n(n(x)) = x$) with boundary conditions (i.e, $n(0)=1, n(1)=0$).*

Standard examples of fuzzy logics are shown in Table 1 [9, 8]. Examples 1-3 are special cases (and limits) of the Frank family of fuzzy logics that are central to our work and formally defined in Definition 3.

Definition 3. *Let $s \in [0,1] \cup \{\infty\}$ then the **Frank family** of T-norms is defined by:*

$$T^s(x,y) = \begin{cases} \min(x,y) & s=0 \\ xy & s=1 \\ \max(x+y-1,0) & s=\infty \\ log_s\left(1+\frac{(s^x-1)(s^y-1)}{s-1}\right) & otherwise \end{cases}$$

The set of intervals in $[0,1]$ forms a bounded partial order $\langle \mathbb{I}, \sqsubseteq, \top, \bot \rangle$[4] where $[l_x,u_x] \leq [l_y,u_y] \Leftrightarrow (l_x \leq l_y) \wedge (u_x \leq u_y)$, $\top = [1,1]$ and $\bot = [0,0]$. As per Gehrke et al. [10] we can point-wise lift a T1FS fuzzy logic $\langle \tilde{\wedge}, \tilde{\vee}, \tilde{\neg} \rangle$ to a IVFS fuzzy logic, i.e., $[l_x,u_x] \odot [l_y,u_y] = [l_x \odot l_y, u_x \odot u_y]$, $\hat{\odot} \in \{\tilde{\wedge}, \tilde{\vee}\}$ and $\tilde{\neg}[l,u] = [\tilde{\neg} u, \tilde{\neg} l]$.

3 Fuzzy data-flow analysis

Static data-flow analyses deduce values of semantic properties that are satisfied the dynamics of the application. The dynamics is formalized as a system of monotone transfer functions and collector functions. Transfer functions describe how blocks alter the semantic properties. Collectors functions merge results from different, possibly mutual exclusive, branches of the application. The solution of the analysis is obtained through Kleene iteration and is a unique fixed-point of the system of equations. In a classical framework the domain of the values is binary, i.e. either true (1) or false (0). The interpretation of these values depends on the type of analysis. The value true means that the property can possibly hold in a *may*-analysis (i.e., it is impossible that the value is always false) while it means that the property always holds in a *must*-analysis. The value false could mean either the opposite of true or that the result is inconclusive.

Our fuzzy data-flow analysis instead computes the partial truth of the property, i.e. values are elements of $[0,1]$. A value closer to 0 means that the property is biased towards false and vice versa. Furthermore the transfer functions are logical formulas from a Frank family fuzzy logic and the collector

[3]The general concept, allowing any $e \in [0,1]$, is called a *uninorm* [9] and is either orlike (i.e., $U(0,1) = U(1,0) = 1$) or andlike (i.e., $U(0,1) = U(1,0) = 0$). Our work does not require the full generality.

[4]This should not be confused with the partial order used in the interval abstraction.

functions are weighted average functions where the constant α is determined prior to performing the analysis. In contrast to the classical framework Kleene iteration proceeds until the results differ by a constant ε which is the maximal error allowed by the solution. The error can be made arbitrarily small.

This section introduces the fuzzy data-flow framework and we prove termination using continuity properties and Banach's fixed-point theorem. Section 4 then presents an example analysis to demonstrate the benefit of the framework. The analysis is performed on a weighted flow-graph $G = \langle V, E, \alpha \rangle$ where V is a set of logical formulas (denoting the transfer function of each block), $E \subseteq V \times V$ is a set of edges (denoting control transfers) and $\alpha_e \in [0,1]$ denotes the normalized contribution for each edge e. As a running example we will use Figure 1 (left) which shows a flow-graph with four nodes and their corresponding logical formula. The flow graph has four control edges denoting contributions between nodes. For instance, Block 1 (B1) receives 0.1 of its contribution from B0 and 0.9 from B2, i.e. $\alpha_{\langle B0,B1\rangle} = 0.1$ and $\alpha_{\langle B2,B1\rangle} = 0.9$.

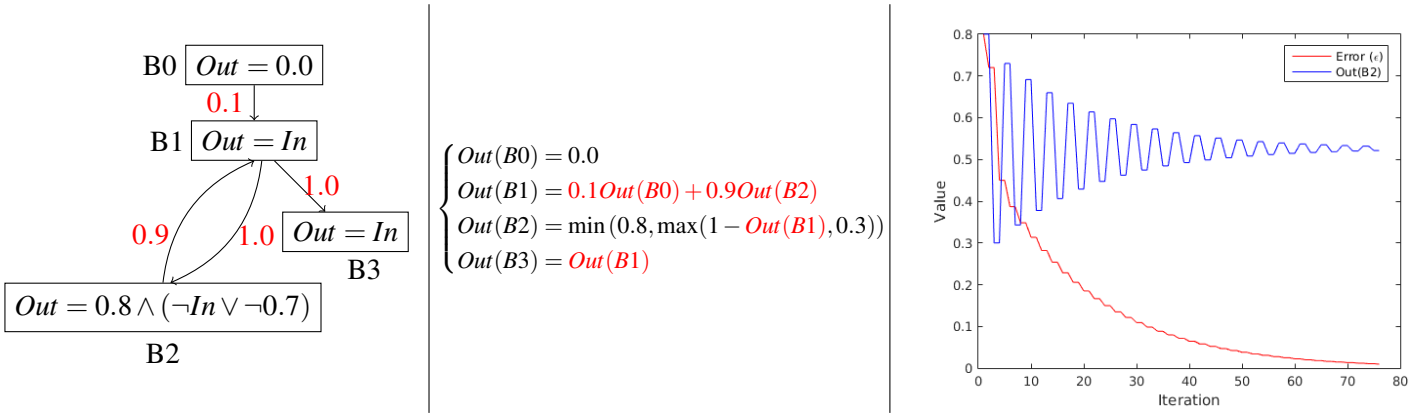


Figure 1: Example flow-graph (left) and its corresponding equation system (middle) and the analysis result and error as a function of iteration (right)

Definition 4. *Let $\mathscr{P}$ be a finite set of properties and $VS\colon \mathscr{P} \mapsto [0,1]$ a valuation for each property. We use $[\![\phi]\!](VS)$ to denote the interpretation of the fuzzy formula ϕ given a VS. Given a flow-graph $G = \langle V, E, \alpha \rangle$ with a unique start node v_{start} the map $GS\colon V \mapsto VS$ describes the value of each property at each node and a fuzzy data-flow analysis is a Kleene iteration of $F : GS \mapsto GS$:*

$$F(S) = \lambda v. \begin{cases} S(v_{start}) & v = v_{start} \\ \sum\limits_{\langle w,v\rangle \in E} \alpha_{\langle w,v\rangle} [\![v]\!](S(w)) & otherwise \end{cases}$$

Figure 1 (middle) shows the equation system, as implied by Definition 4, interpreted in a min-max fuzzy logic for the example flow-graph. The red colored text corresponds to the collector function, i.e. the weighted average, and the normal text is the interpretation of the logical formula. In order to prove termination of a fuzzy analysis we need to introduce a continuity property.

Definition 5. *A function $f\colon [0,1]^n \mapsto [0,1]$ is **K-Lipschitz continuous**[5] iff $\forall x, h\colon |f(\vec{x} - \vec{h}) - f(\vec{x})|_1 \leq K|\vec{h}|_1$. Where $|\vec{x}|_1$ is l_1-norm (i.e., the absolute value) of $\vec{x}$[6]. If $0 \leq K < 1$ then f is called a **contraction mapping** and if $0 \leq K \leq 1$ then f is called a **non-expansive mapping**.*

[5]Our definition restricts the domain and metric of both metric spaces (i.e., for the domain and co-domain of f) compared to the more general, and common, definition of a Lipschitz continuous function.

[6]Other l_p-norms can be used but only if we restrict the logic to the min-max fuzzy logic [15].

In a sequence of applications of a contraction mapping the difference between two consecutive applications will decrease and in the limit reach zero. By imposing a bounded error we guarantee that this sequence terminates in a bounded amount of time. The analysis error and result of B2 as a function of iteration for the example is shown in Figure 1 (right). Note that the error (red line) is decreasing and the value of B2 (blue line) tends towards a final value. We next proceed to prove that any fuzzy data-flow analysis iteratively computes more precise results and terminates in a bounded amount of time for a finite maximum error $\frac{1}{2^q}$ from some $q \in \mathbb{N} - \{0\}$. We let $[0,1]_q$ denote the maximal congruence set of elements from $[0,1]$ that are at least $\frac{1}{2^q}$ apart, i.e. $[0,1]_q = \{\frac{i}{2^q} \mid 0 \leq i \leq 2^q\}$. The set of intervals on $[0,1]$, i.e. $\mathbb{I}$ are defined analogously. For this we prove the *non-expansive* property of fuzzy formulas.

Theorem 1. *Let $x, y, C, w_i \in [0,1]_q$, for some $i \in \mathbb{N}$, $f_i(\vec{x})\colon [0,1]_q^n \mapsto [0,1]_q$ be 1-Lipschitz and $g_i(\vec{x})\colon [0,1]_q^n \mapsto [0,1]_q$ be K_i-Lipschitz.*

- *Functions $x+y$, $x-y$, xy, $\min(x,y)$ and $abs(x)$ are 1-Lipschitz. Constants are 0-Lipschitz.*
- *If $\sum_{i=0}^{N-1} w_i = 1$ then $\sum_{i=0}^{N-1} w_i f_i(\vec{x})$ is 1-Lipschitz.*
- *The composition $g_a \circ g_b$ is $K_a K_b$-Lipschitz.*

Finally,

- *Formulas defined in a Frank family Fuzzy logic are 1-Lipschitz.*
- *If $F\colon \mathbb{I}_q^n \to \mathbb{I}_q$ satisfies $\forall x \in \mathbb{I}_q^n\colon y \in x \Rightarrow f(y) \in F(x)$ then F is 1-Lipschitz.*

In summary, as per Theorem 1:

- Transfer functions in a Frank family fuzzy logic are non-expansive mappings.
- $S(v_{start})$ is constant and hence a contraction mapping.
- The composition of 1) Two non-expansive functions is a non-expansive function and 2) A non-expansive and a contraction function is a contraction function.

As the analysis is performed on the unit interval which together with the l_1-norm forms a complete metric space we can guarantee termination by Banach's fixed-point theorem.

Theorem 2 (Banach fixed-point theorem)**.** *Let (X,d) be a complete metric space and $f\colon X \mapsto X$ a contraction. Then f has a unique fixed-point x^* in X.*

This concludes our development of fuzzy data-flow analysis.

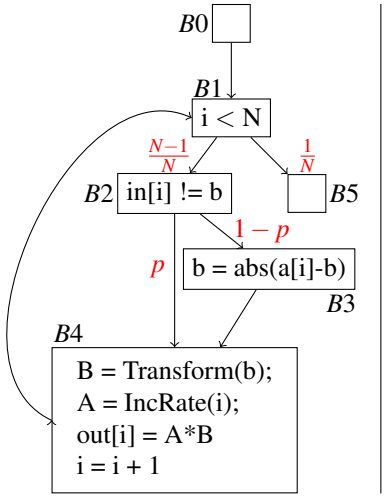
4 Lazy code motion

Improving performance often means removing redundant computations. Computations are said to be fully redundant, but not dead, if the operands at all points remain the same. For two such computations it is enough to keep one and store away the result for later. We can eliminate this redundancy using (global) common sub-expression elimination (GCSE). Furthermore a computation that does not change on some paths is said to be partially redundant. Loop invariant code motion (LICM) finds partially redundant computations inside loops and move these to the entry block of the loop. Lazy code motion is a compiler optimization that eliminate both fully and partially redundant computations, and hence subsumes both CSE and LICM. Knoop-Rüthing-Steffen (KRS) algorithm [13, 7] performs LCM in production compilers such as GCC when optimizing for speed. It consists of a series of data-flow analyses and can be summarized in these four steps:

1. Solve a very busy expression[7] and an available expression data-flow problem [17].
2. Introduce a set that describes the earliest block where an expression must be evaluated.

[7] Knoop et al. [13] refer to this as anticipatable expression data-flow problem.

```
void diffPCM() {
    b = 0, A = 0, B = 0;
    for(i = 0; i < N; i++)
        if(in[i] != b)
            b = abs(in[i]-b);
        B = Transform(b);
        A = IncRate(i);
        out[i] = A*B;
}
```




```
void diffPCM() {
    b = 0, A = 0, B = 0;
    for(i = 0; i < N; i++)
        Update← ANFIS decision of updating b
        Leave← ANFIS decision of leaving b
        if(in[i] != b)
            b = abs(in[i]-b);
            If Update < Leave: Decision error!
        else
            If Update > Leave: Decision error!
        B = Transform(b);
        A = IncRate(i);
        out[i] = A*B;
}
```

Figure 2: `diffPCM` function (left), the corresponding flow-chart (middle) and the version used in Section 4.3 which is annotated with ANFIS classifier invocations (right)

3. Determine the latest control flow edge where the expression must be computed.
4. Introduce *Insert* and *Delete* sets which describe where expressions should be evaluated.

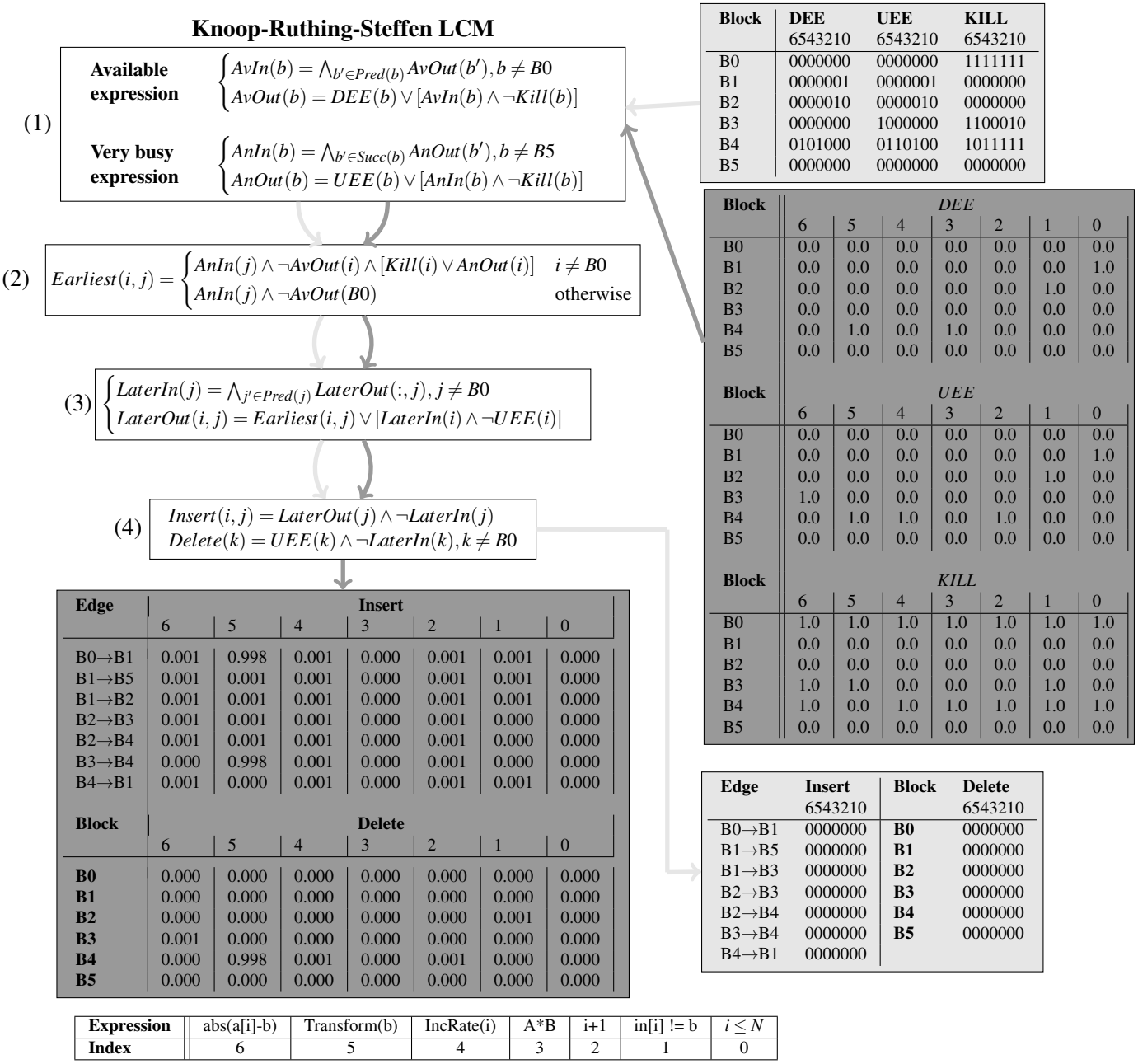
The target domain of the analysis is the set of static expressions in a program. Input to the analysis is three predicates determining properties about the expressions in different blocks:

- An expression "e" is *downward exposed* if it produces the same result if evaluated at the end of the block where it is defined. We use $DEE(b,e)$ to denote if "e" is downward exposed in block "b".
- An expression "e" is *upward exposed* if it produces the same result if evaluated at the start of the block where it is defined. We use $UEE(b,e)$ to denote this.
- An expression "e" is *killed* in block "b" if any variable appearing in "e" is updated in "b". We use $KILL(b,e)$ to denote this.

Very busy expression analysis is a backward-must data-flow analysis that depends on UEE and $KILL$ and computes the set of expressions that is guaranteed to be computed at some time in the future. Similarly Available expression analysis is a forward-must data-flow analysis that depends on DEE and $KILL$ and deduces the set of previously computed expressions that may be reused. The fixed-point system of these two analyses are shown in Figure 3. It is beyond the scope of this paper to further elaborate on the details of these analyses, the interested reader should consider Nielson et al. [17]. Here the LCM algorithm and the data-flow analyses it depends on, are applications we use to demonstrate the benefit of our framework. As such a rudimentary understanding is sufficient.

Consider the simplified differential pulse-code modulation routine `diffPCM` in Figure 2 (left). We assume that N and the relative number of times block $B3$ (denoted p) is statically known[8]. In each iteration `diffPCM` invokes the pure functions `Transform`, to encode the differential output, and `IncRate` to get a quantification rate. We use the KRS-LCM algorithm to determine if these invocations can be made prior to entering the loop and contrast this to a situation where the data-flow analyses are performed in the fuzzy framework. As we will show the "fuzzy KRS-LCM" allows us to uncover opportunites the classical KRS-LCM would miss.

[8] In this demonstration we let $p = 0.999$ and $N = 1000$, but our conclusions hold as N increases and p approaches 1.



Block	DEE	UEE	KILL
	6543210	6543210	6543210
B0	0000000	0000000	1111111
B1	0000001	0000001	0000000
B2	0000010	0000010	0000000
B3	0000000	1000000	1100010
B4	0101000	0110100	1011111
B5	0000000	0000000	0000000

Block	*DEE*						
	6	5	4	3	2	1	0
B0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
B1	0.0	0.0	0.0	0.0	0.0	0.0	1.0
B2	0.0	0.0	0.0	0.0	0.0	1.0	0.0
B3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
B4	0.0	1.0	0.0	1.0	0.0	0.0	0.0
B5	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Block	*UEE*						
	6	5	4	3	2	1	0
B0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
B1	0.0	0.0	0.0	0.0	0.0	0.0	1.0
B2	0.0	0.0	0.0	0.0	0.0	1.0	0.0
B3	1.0	0.0	0.0	0.0	0.0	0.0	0.0
B4	0.0	1.0	1.0	0.0	1.0	0.0	0.0
B5	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Block	*KILL*						
	6	5	4	3	2	1	0
B0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
B1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
B2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
B3	1.0	1.0	0.0	0.0	0.0	1.0	0.0
B4	1.0	0.0	1.0	1.0	1.0	1.0	1.0
B5	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Edge	Insert						
	6	5	4	3	2	1	0
B0→B1	0.001	0.998	0.001	0.000	0.001	0.001	0.000
B1→B5	0.001	0.001	0.001	0.000	0.001	0.001	0.000
B1→B2	0.001	0.001	0.001	0.000	0.001	0.001	0.000
B2→B3	0.001	0.001	0.001	0.000	0.001	0.000	0.000
B2→B4	0.001	0.001	0.001	0.000	0.001	0.000	0.000
B3→B4	0.000	0.998	0.001	0.000	0.001	0.000	0.000
B4→B1	0.001	0.000	0.001	0.000	0.001	0.001	0.000
Block	**Delete**						
	6	5	4	3	2	1	0
B0	0.000	0.000	0.000	0.000	0.000	0.000	0.000
B1	0.000	0.000	0.000	0.000	0.000	0.000	0.000
B2	0.000	0.000	0.000	0.000	0.000	0.001	0.000
B3	0.001	0.000	0.000	0.000	0.000	0.000	0.000
B4	0.000	0.998	0.001	0.000	0.001	0.000	0.000
B5	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Edge	Insert	Block	Delete
	6543210		6543210
B0→B1	0000000	**B0**	0000000
B1→B5	0000000	**B1**	0000000
B1→B3	0000000	**B2**	0000000
B2→B3	0000000	**B3**	0000000
B2→B4	0000000	**B4**	0000000
B3→B4	0000000	**B5**	0000000
B4→B1	0000000		

Expression	abs(a[i]-b)	Transform(b)	IncRate(i)	A*B	i+1	in[i] != b	$i \leq N$
Index	6	5	4	3	2	1	0

Figure 3: Knoop-Rüthing-Steffen LCM formulation (middle) using classical (left) and fuzzy (right/bottom) data-flow analysis

4.1 Type-1 static analysis

The data-flow problems of the KRS algorithm use expressions as domain. The mapping between expressions of `diffPCM` and indexes are listed in Figure 3 (bottom) together with the values of DEE, UEE and $KILL$ for each block (top right). The classical KRS algorithm conclude that both calls must be evaluated in B4 (bottom light gray box, "Delete" matrix, Column 4 and 5).

For the fuzzy data-flow analyses we use the Type-1 Min-Max fuzzy logic. The corresponding fuzzy sets of DEE, UEE and $KILL$ are given in Figure 3 (top dark gray box). Step (1) of the fuzzy KRS-LCM is hence the fixed-point to below system of equations:

Available expression analysis system

$$\begin{cases} AvOut(B0) = 0.0 \\ AvOut(B1) = DEE(B1) \vee \left(\left[\frac{1}{N} AvOut(B0) + \frac{N-1}{N} AvOut(B4) \right] \wedge \neg Kill(B1) \right) \\ AvOut(B2) = DEE(B2) \vee (AvOut(B1) \wedge \neg Kill(B2)) \\ AvOut(B3) = DEE(B3) \vee (AvOut(B2) \wedge \neg Kill(B3)) \\ AvOut(B4) = DEE(B4) \vee ([pAvOut(B2) + (1-p)AvOut(B3)] \wedge \neg Kill(B4)) \\ AvOut(B5) = DEE(B5) \vee (AvOut(B1) \wedge \neg Kill(B4)) \end{cases}$$

Very busy expression analysis system

$$\begin{cases} AnOut(B0) = UEE(B0) \vee (AnOut(B1) \wedge \neg Kill(B1)) \\ AnOut(B1) = UEE(B1) \vee \left(\left[\frac{N-1}{N} AnOut(B2) + \frac{1}{N} AnOut(B5) \right] \wedge \neg Kill(B1) \right) \\ AnOut(B2) = UEE(B2) \vee ([pAnOut(B4) + (1-p)AnOut(B3)] \wedge \neg Kill(B2)) \\ AnOut(B3) = UEE(B3) \vee (AnOut(B4) \wedge \neg Kill(B3)) \\ AnOut(B4) = UEE(B4) \vee (AnOut(B1) \wedge \neg Kill(B4)) \\ AnOut(B5) = 0.0 \end{cases}$$

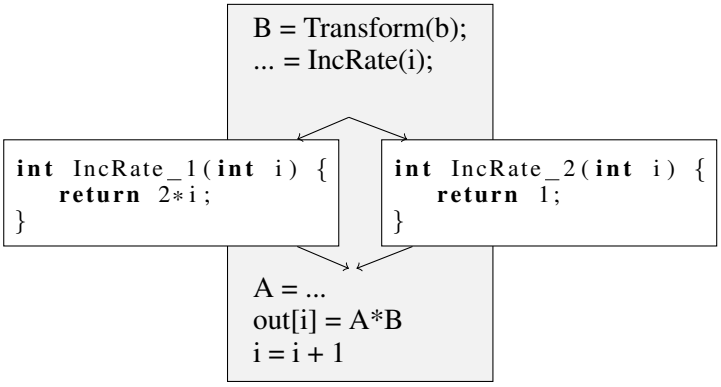
Steps (2) and (4) introduce (constant) predicates and are performed outside the analysis framework. Step (3) is done similarly to step (1). Figure 3 (bottom dark gray box) shows the result from step (4). In contrast to the classical LCM the result implies that it is *very plausible* (0.998) that we can delete the invocation of `Transform` ("Delete" matrix, Column 5) from block B4 and instead add it at the end of B0 and B3 (or start of B1 and B4). However, result for the invocation of `IncRate` remains. This is because the invocation depends on the value of i which is updated at the end of B4.

4.2 Type-2 static analysis

To increase data-flow analysis precision a function call is sometimes inlined at the call site. The improvement can however be reduced if the control-flow analysis is inaccurate and multiple targets are considered for a particular call site. We show how the uncertainty in control-flow and data-flow can be quantified in two different dimensions using type-2 interval fuzzy sets. As per Section 2 we can lift an arbitrary fuzzy predicate to intervals. Here we assume no knowledge about the relative number of calls to each target and treat the different calls non-deterministically.

We assume two different `IncRate` functions, as in Figure 4 (left), have been determined as targets. Their respective UEE and $Kill$ entries are the same but since i is updated at the end of block B4 their DEE entry will differ. The result of `IncRate_1` depends on the variable i and therefore $DEE(B4_1) = \mathbf{0101000}$, in contrast the entry for `IncRate_2` is $DEE(B4_2) = \mathbf{0111000}$, where $\mathbf{0} = [0,0]$ and $\mathbf{1} = [1,1]$. The new entry for block B4 is given by $DEE(B4) = DEE(B4_1) \,\tilde{\vee}\, DEE(B4_2) = \langle \mathbf{0}, \mathbf{1}, [0,1], \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0} \rangle$. The new $Kill$, DEE and UEE sets are given in Figure 4 (right).

Applying the fuzzy KRS-LCM, but with Type-1 min-max fuzzy logic lifted to Interval type-2 min-max fuzzy logic gives the values of *Delete* and *Insert* for expression `IncRate(i)` in Figure 4 (right).



Block	Kill	DEE	UEE
6	[1.0, 1.0]	[0.0, 0.0]	[0.0, 0.0]
5	[0.0, 0.0]	[1.0, 1.0]	[1.0, 1.0]
4	[1.0, 1.0]	[0.0, 1.0]	[1.0, 1.0]
3	[1.0, 1.0]	[1.0, 1.0]	[0.0, 0.0]
2	[1.0, 1.0]	[0.0, 0.0]	[1.0, 1.0]
1	[1.0, 1.0]	[0.0, 0.0]	[0.0, 0.0]
0	[1.0, 1.0]	[0.0, 0.0]	[0.0, 0.0]

Edge	Insert	Block	Delete
B0→B1	[0.001, 0.999]	B0	[0.000, 0.000]
B1→B5	[0.001, 0.999]	B1	[0.000, 0.000]
B1→B3	[0.001, 0.999]	B2	[0.000, 0.000]
B2→B3	[0.001, 0.999]	B3	[0.000, 0.000]
B2→B4	[0.001, 0.999]	B4	[0.002, 0.999]
B3→B4	[0.001, 0.999]	B5	[0.000, 0.000]
B4→B1	[0.000, 0.999]		

Figure 4: Implementations of `IncRate` inlined in block B4 (left); DEE, UEE and Kill vectors of block B4 and *Delete Insert* analysis result for expression `IncRate(i)` (right)

The result for invoking `IncRate` prior to the loop is $[0.001, 0.999]$ as opposed to 0.001 from the Type-1 analysis in Section 4.1. The added dimension in the result of the type-2 fuzzy analysis allows us to differentiate uncertain results from pessimistic results. In the given example we showed that the result of Section 4.1 is a pessimistic over-generalization and that the two paths need to be considered seperately to increase precision.

4.3 Hybrid analysis

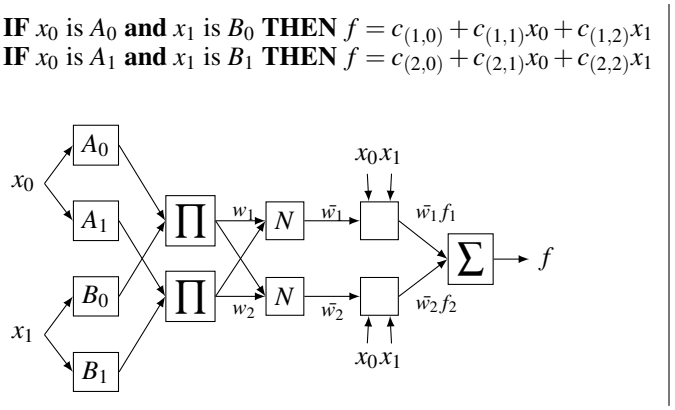
The result from a fuzzy data-flow analysis is a set of fuzzy membership degrees. This section shows how the result can automatically be improved following the static analysis using a fuzzy regulator/classifier, if more specific information is provided at a later point. The classifier, a Takagi-Sugeno Adaptive-Network-based fuzzy inference system (TS-ANFIS) [11, 12] shown in Figure 5, is composed of five layers:

1. Lookup fuzzy membership degree of the input value.
2. Compute the *firing strength of a rule*, i.e. conjunction of all membership degrees from each rule.
3. Normalize the firing strengths, i.e., $\bar{w}_i = w_i / \sum_j w_j$.
4. Weight the normalized firing strength to the consequent output of the rule $f_i(x)$.
5. Combine all rule classifiers, i.e. $f = \sum_i \bar{w}_i f_i$.

This classifier uses a polynomial (i.e., the consequent part of the adaptive IF-THEN rules) to decide the output membership. The order of the TS-ANFIS is the order of the polynomial. The classification accuracy of the TS-ANFIS can be improved online/offline by fitting the polynomial to the input data. For a first-order TS-ANFIS this can be implemented as follows:

- (*Offline*) (Affine) Least square (LS) optimization [11] is a convex optimization problem that finds an affine function (i.e., $y = a_0 + \sum_{i=1}^{n} a_i x_i$) which minimizes $||A[1;X] - Y||_2^2$ where X and Y are the input and output vectors of the training set.
- (*Online*) Least mean square (LMS) [11] is an adaptive filter that gradually (in steps of a given constant μ) minimizes $\mathbb{E}\left[|y - f(x)|^2\right]$, where $\langle x, y \rangle$ is an input/output sample.

To exemplify the functionality of the TS-ANFIS we consider the classification of $\vec{x} = \langle 0.6, 0.2 \rangle$ using the two rule TS-ANFIS from Figure 5 (left). Let $f_1(\vec{x}) = 0.2x_0 - 0.43x_1$, $f_2(\vec{x}) = 0.1x_1 + 0.5$ and membership functions be given as in Figure 5 (right). The membership degrees are marked in the figure as $\mu_{A0}(x0) =$



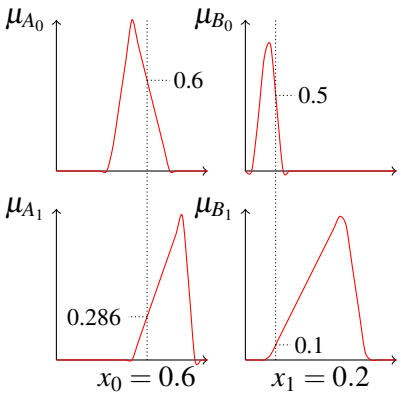


Figure 5: First-order Takagi-Sugeno ANFIS with two rules and two variables (left) and four example fuzzy sets (right)

0.6, $\mu_{B0}(x1) = 0.5$ for the first rule and $\mu_{A1}(x0) = 0.286$, $\mu_{B1}(x0) = 0.1$ for the second rule. Hence the weight of the first rule (i.e., w_1) is $0.6 \wedge 0.5 = 0.5$ and the second rule (i.e., w_2) is $0.286 \wedge 0.1 = 0.1$. The normalized weights are then $\bar{w_1} = 0.833$ and $\bar{w_1} = 0.167$. As the consequence functions output $f_1(\vec{x}) = 0.034$ and $f_2(\vec{x}) = 0.52$ we produce the prediction $0.833 f_1(\vec{x}) + 0.167 f_2(\vec{x}) = 0.115$.

We return to the `diffPCM` function and again consider if we can invoke `Transform(b)` prior to entering the loop. We saw in Section 4.1 that the fuzzy membership degree was 0.998. To improve classification accuracy we let the TS-ANFIS also use the i variable and the first input value (i.e., $in[0]$). These variables were not part of the analysis and so we conservatively assume the fuzzy membership degree to be the same for any value of these variables (in our experiments: 1.0). As shown in Figure 2 (right), we inserted calls to compute the ANFIS decision of updating and keeping the variable b constant in the `diffPCM` function. If the incorrect decision was made the error was noted and an error rate computed after handling all input samples.

We consider invoking the `diffPCM` function on four different input sets. Each input set defined as 10 periods with 25 input values in each period. The input sets (i.e., `in[...]`) is given in Figure 6 (top). We use the LMS algorithm[9] after each incorrect classification and the LS algorithm if the error rate of a period was larger than or equal to 80%. Note that the values of a period is not always perfectly representable by a linear classifier and sometimes varies between different periods, although periods are "*similar*". Hence we do not expect the classifier to be monotonically improving with increasing period. As shown in the result in Figure 6 (bottom) the classification error decreases fast with both period and input sample. In two cases a small residual error remains after the final period. This show that the TS-ANFIS can improve the analysis result dynamically and hence increase the accuracy of when `Transform` can be invoked prior to entering the loop.

5 Related work

Most systems include elements (e.g., input values, environment state) where information is limited but probabilistic and/or non-deterministic uncertainty can be formulated. For these systems a *most likely* or even *quantitative analysis* of properties is possible. Often this analysis relies on a probability theory for

[9]The constant μ for the four different runs was set to 0.001, 0.05, 0.15 and 0.1 respectively.

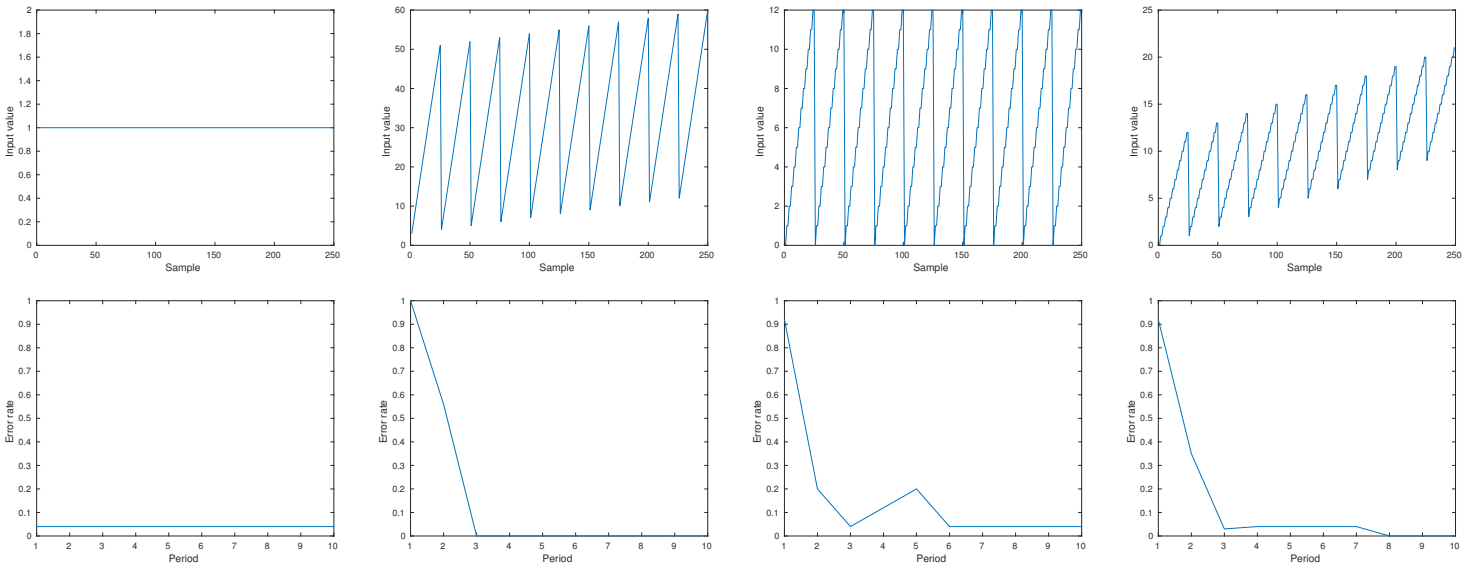


Figure 6: 10×25 input values (top) and the corresponding classification error rate (bottom)

logical soundness. Cousot and Monerau [3] introduced a unifying framework for probabilistic abstract interpretation. Much work have since, although perhaps implicitly, relied on their formulation. Often probabilistic descriptions are known with imprecision that manifests as non-deterministic uncertainty [2]. Adje et al. [1] introduced an abstraction based on the zonotope abstraction for Dempster-Shafer structures and P-boxes[10].

Di Pierro et al. [6] developed a probabilistic abstract interpretation framework and demonstrated an alias analysis algorithm that could guide the compiler in this decision. They later formulated data-flow problems (e.g., liveness analysis) in the same framework [5]. An important distinction between their (or similar probabilistic frameworks) and classical frameworks is the definition of the confluence operator. In contrast to a classical may- or must framework they use the weighted average. This is similar to the work by Ramalingam [20] that showed that the meet-over-paths (MOP) solution exists for such confluence operator with a transfer function defined in terms of min, max and negation (i.e., the Min-max fuzzy logic). Our work extends this to allow other transfer functions and integrates the static data-flow analysis with a dynamic refinement mechanism through fuzzy control theory.

6 Conclusion

A major problem for static program analysis is the limited input information and hence the conservative results. To alleviate the situation dynamic program analysis is sometimes used. Here accurate information is available, but in contrast to its static counter-part the results only cover a single or few runs. To bridge the gap, and find a promising middle-ground, probabilistic/speculative program analysis frameworks have been proposed. These frameworks can be considered to intersect both by being a static program analysis that uses dynamic information. We have introduced an abstraction based on fuzzy sets that supports such analyses. We applied our abstraction to data-flow problems of use for speculative compilation and showed how our analysis unveils opportunities that previous approaches could

[10] Lower and upper bounds on a cumulative probability distribution functions

not express and reason about. We furthermore showed that an abstraction based on fuzzy sets admit mechanisms from fuzzy control theory to enhance the analysis result dynamically allowing for a hybrid analysis framework.

References

[1] Assale Adje, Olivier Bouissou, Jean Goubault-Larrecq, Eric Goubault & Sylvie Putot (2014): *Static Analysis of Programs with Imprecise Probabilistic Inputs*. In: *Verified Software: Theories, Tools, Experiments*, *Lecture Notes in Computer Science* 8164, Springer Berlin Heidelberg, pp. 22–47.

[2] Patrick Cousot & Radhia Cousot (2014): *Abstract Interpretation: Past, Present and Future*. In: *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, ACM, pp. 2:1–2:10.

[3] Patrick Cousot & Michaël Monerau (2012): *Probabilistic Abstract Interpretation*. In: *22nd European Symposium on Programming (ESOP 2012)*, *Lecture Notes in Computer Science* 7211, Springer-Verlag, pp. 166–190.

[4] Jeff Da Silva & J. Gregory Steffan (2006): *A Probabilistic Pointer Analysis for Speculative Optimizations*. In: *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, ACM, pp. 416–425.

[5] Pierro A Di & H Wiklicky (2013): *Probabilistic data flow analysis: a linear equational approach*. In: *Proceedings of the Fourth International Symposium on Games, Automata, Logics and Formal Verification*, pp. 150–165.

[6] Alessandra Di Pierro, Chris Hankin & Herbert Wiklicky (2007): *A Systematic Approach to Probabilistic Pointer Analysis*. In: *Programming Languages and Systems*, *Lecture Notes in Computer Science* 4807, Springer Berlin Heidelberg, pp. 335–350.

[7] Karl-Heinz Drechsler & Manfred P. Stadel (1993): *A Variation of Knoop, RüThing, and Steffen's Lazy Code Motion*. *SIGPLAN Not.* 28(5), pp. 29–38.

[8] D. Dubois & H. Prade (1980): *Fuzzy sets and systems - Theory and applications*. Academic press, New York.

[9] D. Dubois, H.M. Prade & H. Prade (2000): *Fundamentals of Fuzzy Sets*. The Handbooks of Fuzzy Sets, Springer US.

[10] Mai Gehrke, Carol Walker & Elbert Walker (1996): *Some comments on interval valued fuzzy sets*. *International Journal of Intelligent Systems* 11(10), pp. 751–759.

[11] J.-S.R. Jang (1993): *ANFIS: adaptive-network-based fuzzy inference system*. *Systems, Man and Cybernetics, IEEE Transactions on* 23(3), pp. 665–685.

[12] Jyh-Shing Roger Jang & Chuen-Tsai Sun (1997): *Neuro-fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[13] Jens Knoop, Oliver Rüthing & Bernhard Steffen (1992): *Lazy Code Motion*. In: *Proceedings of the ACM SIGPLAN 1992 Conference on Programming Language Design and Implementation*, PLDI '92, ACM, pp. 224–234.

[14] S. Maleki, Yaoqing Gao, M.J. Garzaran, T. Wong & D.A. Padua (2011): *An Evaluation of Vectorizing Compilers*. In: *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pp. 372–382.

[15] A. Mesiarov (2007): *k-lp-Lipschitz t-norms*. *International Journal of Approximate Reasoning* 46(3), pp. 596 – 604. Special Section: Aggregation Operators.

[16] Markus Mock, Manuvir Das, Craig Chambers & Susan J. Eggers (2001): *Dynamic Points-to Sets: A Comparison with Static Analyses and Potential Applications in Program Understanding and Optimization*. In:

Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, PASTE '01, ACM, pp. 66–72.

[17] Flemming Nielson, Hanne R. Nielson & Chris Hankin (1999): *Principles of Program Analysis*. Springer-Verlag New York, Inc.

[18] P.M. Petersen & D.A. Padua (1996): *Static and dynamic evaluation of data dependence analysis techniques*. *Parallel and Distributed Systems, IEEE Transactions on* 7(11), pp. 1121–1132.

[19] Dimitrios Prountzos, Roman Manevich, Keshav Pingali & Kathryn S. McKinley (2011): *A Shape Analysis for Optimizing Parallel Graph Programs*. In: *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, ACM, pp. 159–172.

[20] G. Ramalingam (1996): *Data Flow Frequency Analysis*. In: *Proceedings of the ACM SIGPLAN 1996 Conference on Programming Language Design and Implementation*, PLDI '96, ACM, pp. 267–277.

[21] ConstantinoG. Ribeiro & Marcelo Cintra (2007): *Quantifying Uncertainty in Points-To Relations*. In: *Languages and Compilers for Parallel Computing*, *Lecture Notes in Computer Science* 4382, Springer Berlin Heidelberg, pp. 190–204.

Appendix A: Omitted proofs

of Theorem 1. Let $x,y,C,w_i \in [0,1]$, *for some* $i \in \mathbb{N}$, $f_i(\vec{x}) : [0,1]_q^n \mapsto [0,1]_q$ *be* 1*-Lipschitz and* $g_i(\vec{x}) : [0,1]_q^n \mapsto [0,1]_q$ *be* K_i*-Lipschitz.*

1. *Functions* $x+y$, $x-y$, xy, $\min(x,y)$ *and* $abs(x)$ *are* 1*-Lipschitz. Constants are* 0*-Lipschitz* Let $b \in [x,x+h]$ for some $0 \leq x \leq x+h \leq 1$:

   (a) $g(x) = abs\,(x)$:

   $$\begin{aligned} |g(x+h)-g(x)| &= |abs\,(x+h)-abs\,(x)| && \text{By definition} \\ &= |x+h-x| && x,h \in [0,1] \\ &\leq 1|h| \end{aligned}$$

   (b) $g(x,y) = x+y$:

   $$\begin{aligned} |g(x+h_1,y+h_2)-g(x,y)| &= |((x+h_1)+(y+h_2))-(x+y)| && \text{By definition} \\ &= |h_1+h_2| \\ &\leq |h_1|+|h_2| && \text{Triangle inequality} \\ &= 1|h| && \text{Distributivity} \end{aligned}$$

   (c) $g(x,y) = x-y$:

   $$\begin{aligned} |g(x+h_1,y+h_2)-g(x,y)| &= |((x+h_1)-(y+h_2))-(x-y)| && \text{By definition} \\ &= |h_1+(-1)h_2| \\ &\leq |h_1|+|-1||h_2| && \text{Triangle inequality} \\ &= 1|h| && \text{Distributivity} \end{aligned}$$

   (d) $g(x,y) = xy$:

   $$\begin{aligned} |g(x+h_1,y+h_2)-g(x,y)| &= |((x+h_1)(y+h_2))-(xy)| && \text{By definition} \\ &= |h_1y+xh_2| \\ &\leq |h_1+h_2| && 0 \leq x,y \leq 1 \\ &\leq |h_1|+|h_2| && \text{Triangle inequality} \\ &= 1|h| && \text{Distributivity} \end{aligned}$$

(e) $g(x,y) = \min(x,y)$:

$$
\begin{aligned}
|g(x+h_1,y+h_2) - g(x,y)| &= |\min(x+h_1,y+h_2) - \min(x,y)| && \text{By definition}\\
&= \left|\left(\frac{x+h_1+y+h_2}{2} - \frac{|x+h_1-y-h_2|}{2}\right) - \left(\frac{x+y}{2} - \frac{|x-y|}{2}\right)\right| && \min(x,y) = \frac{x+y}{2} - \frac{|x-y|}{2}\\
&\leq \left|\frac{h_1+h_2}{2} - \frac{|x|+|-1||y|+|h_1-h_2|}{2} + \frac{|x|+|-1||y|}{2}\right|\\
&= \left|\frac{h_1+h_2}{2} - \frac{|h_1-h_2|}{2}\right|\\
&= |\min(h_1,h_2)|\\
&\leq |h_1+h_2|\\
&= 1|h| && \text{Triangle inequality}
\end{aligned}
$$

(f) $g(x,y) = C$

$$|g(x+h) - g(x)| = |C-C| = 0 \leq 0|h|$$

2. *If* $\sum_{i=0}^{N-1} w_i = 1$ *then* $\sum_{i=0}^{N-1} w_i f_i(\vec{x})$ *is* 1*-Lipschitz*

$$
\begin{aligned}
\left|g(\vec{x}+\vec{h}) - g(\vec{x})\right| &= \left|\sum_{i=0}^{N-1} w_i f(x \vec{+} h) - \sum_{i=0}^{N-1} w_i f(\vec{x})\right| && \text{By definition}\\
&= \left|\sum_{i=0}^{N-1} w_i \left(f(x \vec{+} h) - f(\vec{x})\right)\right| && \text{Associativitiy and commutativity}\\
&\leq \left(\sum_{i=0}^{N-1} w_i K_i\right)|h| && \text{Triangle inequality, distributivity and } w_i \geq 0\\
&= 1|h| && K_i = 1 \text{ and } \sum_i w_i = 1
\end{aligned}
$$

3. *The composition* $g_a \circ g_b$ *is* $K_a K_b$*-Lipschitz* $g(\vec{x}) = f_a(\vec{x}) \circ f_a(\vec{x})$:

$$
\begin{aligned}
|g(x+h) - g(x)| &= \left|f_a(f_b(x \vec{+} h)) - f_a(f_b(\vec{x}))\right| && \text{By definition}\\
&\leq K_a \left|f_b(x \vec{+} h) - f_b(\vec{x})\right| && \text{Definition 5}\\
&\leq K_a K_b |h| && \text{Definition 5}
\end{aligned}
$$

4. *Formulas defined in a Frank family Fuzzy logic are* 1*-Lipschitz.* This follows from structural induction on the height of parse tree of the predicate $P(x)$. By De Morgan's laws it is enough to show the induction step for $\vee$ and $\neg$.

   - **Base case**:
     - v: $g(x) = x$: $|g(x+h) - g(x)| = |x+h-x| \leq 1|h|$.
     - $\top$ or $\bot$: $g(x) = C$: 1-Lipschitz by 3.

- **Induction step**:
  - $\neg\phi \equiv 1-\phi$: 1-Lipschitz from the base case for constants ($\top$ and $\bot$) and cases 1c and 3 and assumption that ϕ is 1-Lipschitz.
  - $\phi_1 \vee \phi_2$: $\begin{cases} \min(x,y) & \text{1-Lipschitz from Theorem 1 case 1e} \\ xy & \text{1-Lipschitz from Theorem 1 case 1d} \\ \max(x+y-1,0) & \text{Equal to } 1-\min(2-x-y,1) \text{ using De Morgans law.} \\ & \text{This expression is 1-Lipschitz from Theorem 1 case 1c, 1e and 3} \end{cases}$

5. *If* $F : \mathbb{I}_q^n \to \mathbb{I}_q$ *satisfy* $\forall x \in \mathbb{I}_q^n : y \in x \Rightarrow f(y) \in F(x)$ *then* F *is* 1*-Lipschitz*

   $F : \mathbb{I}_q^n \to \mathbb{I}_q$ can be decomposed into two functions $F_l : \mathbb{I}_q^n \to [0,1]_q$ and $F_u : \mathbb{I}_q^n \to [0,1]_q$ such that $\forall i : F(i) = [F_l(i), F_u(i)]$, i.e., $F_l(i)$ gives the infimum of $f(i)$ and $F_u(i)$ gives the supremum of i. We show that both F_l and F_u are 1-Lipschitz continuous:

   - $F_l(I) = \inf_{i \in I} f(i)$: Assume $I = [l,u]$, since I is finite we can rewrite the operation as pairwise applications of min, i.e., $min(f(l), min(f(l+\frac{1}{2^q}), min(f(l+\frac{2}{2^2}), ...)))$. As per above case 1e min is 1-Lipschitz. Similarly the composition of two 1-Lipschitz functions is also 1-Lipschitz, or in extension, a finite number of compositions.
   - $F_u(I) = \sup_{i \in I} f(i)$: $\max(x,y)$ is equivalent to $1-\min(1-x,1-y)$ which is 1-Lipschitz by above case 1c and 1e so proof follows in the same way as the $F_l(I)$ case.

□

Extended Abstract: An Experimental Study of a Bucketing Approach*

Yuri Gil Dantas
TU Darmstadt
Darmstadt, Germany

Tobias Hamann
TU Darmstadt
Darmstadt, Germany

Heiko Mantel
TU Darmstadt
Darmstadt, Germany

Johannes Schickel
TU Darmstadt
Darmstadt, Germany

<lastname>@mais.informatik.tu-darmstadt.de

1 Introduction

When a secret has influence on the timing of a program, an attacker can measure the execution time of the program in order to learn some information about the secret. More specifically, this can be done by sending ordinary inputs to the program and analyzing the time taken to execute the program. Traditionally, these attacks, namely Timing Side-Channel Attacks [2], are carried out against cryptographic implementations [2, 13] and web applications [1, 6]. Indeed, there have been several attacks developed against TLS protocol [2], AES [5] and RSA implementations [11], where researchers demonstrated the feasibility of fully recovering the secret key.

Although several approaches [3, 14, 12, 8, 7] have been proposed in order to eliminate timing side-channel attacks, the problem is still not solved, mainly due to practicality and effectiveness reasons. For instance, implementations based on the *static transformation* [8] approach are not fully practical due to the large performance penalty caused by the transformation. Moreover, *dynamic transformation* [7] is not always effective as demonstrated in [4].

Eliminating timing side-channel attacks is challenging, as countermeasures should not only eliminate these attacks by reducing the amount of information leakage from the program, but also should be practical to use. With this in mind, another approach, namely Bucketing [14, 9], has been proposed. Bucketing is a quantitative approach for reducing timing side-channel attacks by decreasing the number of possible timing observations, while minimizing the performance penalty. Although Bucketing has been shown to be sound, it has not been implemented to the best of our knowledge. In this paper, we provide an implementation of Bucketing at the application level. More concretely, we implement Bucketing using a runtime enforcement tool and experimentally evaluate the effectiveness of our implementation for reducing timing side-channel attacks. In summary, the contributions of this paper are two-fold:

- We implement Bucketing at the application level using a runtime enforcement tool. Our implementation is generic in the sense that it can be applied to any Java program with deterministic timing behavior, which is a foundational assumption of Bucketing [14].
- We evaluate the effectiveness of our implementation. For this, we carry out several experiments, with and without using Bucketing. In each experiment, we measure the running time of the program for different secret input values. For all experiments, we observed a quantitative reduction of information leakage from the program when using our implementation.

This paper is organized as follows. Section 2 introduces the concept of Bucketing. Section 3 explains briefly how we implemented Bucketing using a runtime enforcement tool, and Section 4 contains our experimental results. Finally, in Section 5, we conclude the paper by discussing future work.

*This work has been funded by the DFG as part of the project Secure Refinement of Cryptographic Algorithms (E3) within the CRC 1119 CROSSING.

Submitted to:
QAPL 2017

2 Bucketing Approach

Bucketing is a quantitative approach that allows one to discretize the execution time of a program in a way that the results of the computation are only returned at a small number of fixed points in time [14, 9]. That is, Bucketing aims to split all critical output values of a program into buckets such that each output has to wait until the enclosing bucket's upper bound time to be released.

Instead of describing the complete detail of Bucketing, which we refer to [14], we describe its behavior by means of an example (depicted in Figure 1). Assume a program that leaks sensitive information when releasing output events such that an attacker can make (four) different observations about the secret just by measuring the response time of the program. Next, assume that two buckets are defined, b_1, with an upper bound time of t_{b_1}, and b_2, with an upper bound time of t_{b_2}, where the execution time of secret input 1 is allocated into b_1 and secret inputs 2, 3 and 4 into b_2. As a result, whenever this program runs an operation wrt. secret input 1 (similar for inputs 2, 3, and 4), the output event will be held by b_1 until the execution time reaches t_{b_1}. Considering this scenario, the number of timing observations are reduced from four to two, and consequently the power of the attacker to gain information about this secret is also reduced. Moreover, in comparison to static transformation, Bucketing also keeps the performance overhead of such a program minimal. This can be clearly seen on the right side of Figure 1, since not all secret inputs were allocated to the worst-case execution time (i.e. t_{b_2}).

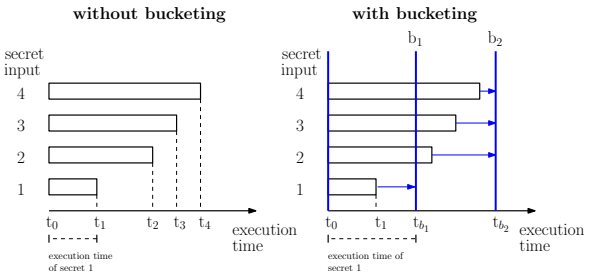


Figure 1: Illustration of a program's timing behavior when releasing sensitive outputs (without Bucketing on the left and with Bucketing on the right).

3 Bucketing Implementation

We implement Bucketing using a runtime enforcement tool, namely CliSeAu [10]. CliSeAu has a modular architecture consisting of four components: interceptor, coordinator, enforcer and local policy. For this particular implementation we just focus on the interceptor and enforcer. *Interceptor* is a component that performs the activity of intercepting attempts of the program to perform security-relevant events and *enforcer* is a component that enforces a countermeasure on the target program. For instantiating CliSeAu for Bucketing, one needs to define the sensitive methods (i.e. code that operates on secret data) of the target program such that CliSeAu can track each call of these methods. Besides, it is also required to instantiate the enforcer by defining the amount of buckets and their respective sizes.

For the sake of space, we only present the sequence diagram (depicted in Figure 2) that describes part of the flow events of our implementation. Firstly, the interceptor intercepts a security-relevant event whenever a sensitive method call is performed by the program. The interceptor sets the initial time of the event and forwards such an event to the enforcer. The enforcer can specify code to execute before and after the security-relevant event. In our implementation, we specify

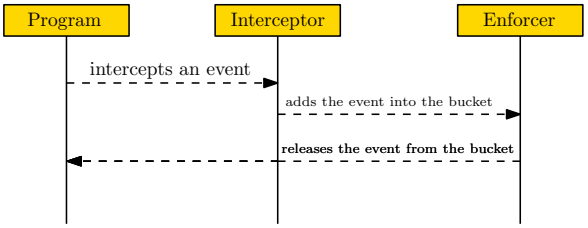


Figure 2: Simplified Diagram of Bucketing implementation

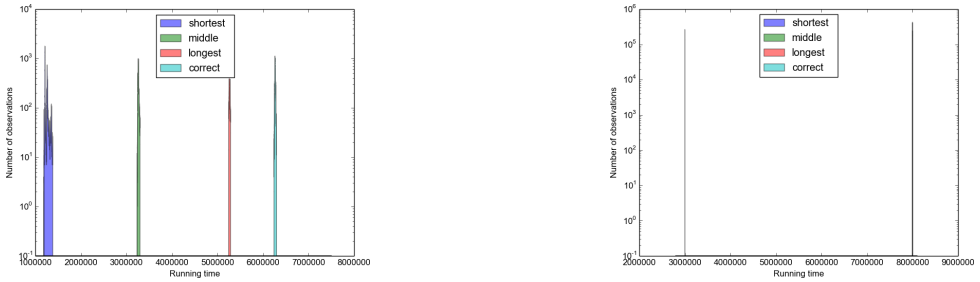
Bucketing for being executed after a security-relevant event such that the enforcer only releases the event when the upper bound time of the current bucket is reached.

Our Bucketing implementation is generic in the sense that it can be (easily) applied to any Java program with deterministic timing behavior, which is a foundational assumption of Bucketing. For more details about CliSeAu's genericity, we refer to [10].

4 Experiments

Our goal is to experimentally evaluate the effectiveness of our Bucketing implementation upon reducing timing side-channel attacks. For the sake of simplicity, we have implemented a simplified example of a client-server application where legitimate clients authenticate (for integrity reasons) their requests into the server using Message Authentication Code (MAC). For this, both legitimate client and server share a common secret key, which is required to build a valid MAC for arbitrary requests. In order to verify a MAC, our server builds its own MAC and compares with the MAC sent by the client. Finally, this comparison is performed by a string comparison method, where we intentionally add delay in four parts of the comparison in order to have a clear timing difference between the responses[1]. As a result, an attacker can explore this timing difference in order to construct a valid MAC. Our assumption is that if the time response of an input_1 takes longer than an input_2 the attacker is closer to guess the correct MAC.

We carry out our experiments as follows. Firstly, we explore the timing side-channel vulnerability of our server by sending four distinct secret inputs, namely *shortest*, *middle*, *longest*, and *correct*. The first three secret inputs (shortest, middle and longest) mean that the first, the middle, and the last character of the MAC are, respectively, incorrect. Moreover, the correct input means that all characters are correct. Figure 3a shows the results of this experiments when not using Bucketing. We can clearly observe differences in the running time values that correspond to four different secret input values. This gives us a hint that an attacker can adaptively infer the expected MAC value by sending arbitrary MACs.



(a) Running time when Bucketing is not applied

(b) Running time when Bucketing is applied

Figure 3: Running time values that correspond to four secret input values

Secondly, we investigate the effectiveness of our implementation upon reducing the timing-side channel vulnerability in our server. For this, we define two buckets of size 3 and 8 ms with the goal of decreasing the power of a potential attacker by reducing her number of timing observations, while taking performance into account, since only one bucket would suffice, in theory, to eliminate the vulnerability. Figure 3b depicts the running time values that correspond to four different secret inputs when Bucketing is applied. On one hand, we can observe that the running time values for the shortest input are always released at around 3 ms (i.e. the upper bound size of the first bucket). On the other hand, we can observe that the other three running time values are overlapping at around 8 ms. Therefore, in contrast to 3a,

[1]This intentional delay simulates programs where the timing differences between the observations are in the range of a few milliseconds rather than nanoseconds. Attacks on programs in this timing range have been shown in e.g. [6].

we cannot observe (much) difference in the running time values that correspond to these three secret input values. We consider these results promising, as they hint on the fact that our implementation of Bucketing is indeed effective in reducing the timing side-channel in our server.

5 Summary and Future Work

This paper provides an implementation of Bucketing at the application level, which is built on our tool for dynamic enforcement of security requirements in Java programs. We carry out a number of experiments for demonstrating its effectiveness. In summary, our experimental results give us a hint that our implementation is effective to reduce timing side-channel attacks. There are many directions for future work, e.g., we are currently investigating how precise and accurate our implementation is, i.e. how close to the actual bucket our implementation releases the information. We are investigating how effective our implementation is to reduce the capacity of timing side-channels [15]. Finally, we are also interested in applying our implementation to a more realistic scenario, where we do not make simplifying assumptions wrt. running time.

References

[1] Martin R. Albrecht & Kenneth G. Paterson: *Lucky Microseconds: A Timing Attack on Amazon's s2n Implementation of TLS*. In: *Advances in Cryptology - EUROCRYPT 2016*.

[2] Nadhem J. AlFardan & Kenneth G. Paterson: *Lucky Thirteen: Breaking the TLS and DTLS Record Protocols*. In: *2013 IEEE Symposium on Security and Privacy, SP 2013*.

[3] Aslan Askarov, Danfeng Zhang & Andrew C. Myers: *Predictive Black-Box Mitigation of Timing Channels*. In: *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010*.

[4] Michael Backes & Boris Köpf: *Formally Bounding the Side-Channel Leakage in Unknown-Message Attacks*. In: *Computer Security - ESORICS 2008*.

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

[6] Andrew Bortz & Dan Boneh: *Exposing private information by timing web applications*. In: *Proceedings of the 16th International Conference on World Wide Web, WWW 2007*.

[7] Benjamin A. Braun, Suman Jana & Dan Boneh (2015): *Robust and Efficient Elimination of Cache and Timing Side Channels*. *CoRR* abs/1506.00189.

[8] Bart Coppens, Ingrid Verbauwhede, Koen De Bosschere & Bjorn De Sutter: *Practical Mitigations for Timing-Based Side-Channel Attacks on Modern x86 Processors*. In: *30th IEEE Symposium on Security and Privacy*.

[9] Goran Doychev & Boris Köpf: *Rational Protection against Timing Attacks*. In: *IEEE 28th Computer Security Foundations Symposium, CSF 2015*.

[10] R. Gay, J. Hu & H. Mantel (2014): *CliSeAu: Securing Distributed Java Programs by Cooperative Dynamic Enforcement*. In: *Proceedings of the 10th International Conference on Information Systems Security (ICISS)*, LNCS 8880, Springer, pp. 378–398.

[11] Mehmet Sinan Inci, Berk Gülmezoglu, Gorka Irazoqui, Thomas Eisenbarth & Berk Sunar: *Cache Attacks Enable Bulk Key Recovery on the Cloud*. In: *Cryptographic Hardware and Embedded Systems - CHES 2016*.

[12] Emilia Käsper & Peter Schwabe: *Faster and Timing-Attack Resistant AES-GCM*. In: *Cryptographic Hardware and Embedded Systems - CHES 2009*.

[13] Paul C. Kocher: *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*. In: *Advances in Cryptology - CRYPTO '96*.

[14] Boris Köpf & Markus Dürmuth: *A Provably Secure and Efficient Countermeasure against Timing Attacks*. In: *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009*.

[15] Claude E. Shannon: *A mathematical theory of communication*. *Mobile Computing and Communications Review, 2001*.

Mean-payoff objectives for Markov Decision Processes

Pranav Ashok[1], Krishnendu Chatterjee[2], Przemysław Daca[2],
Jan Křetínský[1] and Tobias Meggendorfer[1]

[1]Technical University of Munich, Germany
[2]IST Austria

Markov decision processes (MDPs) are standard models for probabilistic systems with non-deterministic behaviors. The mean-payoff objective provides a mathematically elegant formalism to express performance related properties. The value-iteration (VI) approach provides one of the simplest and most efficient algorithmic approaches for MDPs with other properties (such as reachability objectives). Unfortunately, the straightforward VI approach does not work for the mean-payoff objective in MDPs. In particular, there is no stopping criterion which can give guarantees that the solution obtained through VI is ε-close to the optimal solution.

In our ongoing work, the contributions are threefold. (i) We refute a conjecture (presented in [4]) related to stopping criteria for mean-payoff objectives in MDPs; (ii) we present two practical algorithms for the mean-payoff objective in MDPs based on VI; and (iii) we present experimental results showing that our approach significantly outperforms the standard approaches on several benchmarks.

A core idea which we exploit is the fact that for infinite horizon objectives like mean-payoff, only rewards which can be obtained in maximal end-components (MECs) of the MDP matter. Keeping this in mind, we present the following two algorithms.

In the first, we show that a combination of local VI in MECs and VI for reachability objectives can provide approximation guarantees. The stopping criterion is known [4] for VI in communicating MDPs, the class of MDPs in which for every pair of two states s_i and s_j, there exists a deterministic strategy under which s_i is reachable from s_j. We first identify different MECs in the MDP and then use the fact that a MEC can be identified with a communicating MDP, to run VI on it until the desired precision is achieved. Next, we collapse these MECs into their representative states and reduce the problem of solving the mean-payoff objective to a problem of computing the reachability objective on a transformed MDP. The algorithm presented in [1], which is a version of asynchronous value iteration using sampling, allows us to compute the reachability objective with the desired guarantees.

In the second, we present an anytime algorithm based on the bounded read-time dynamic programming (BRTDP) approach [3]. In the BRTDP approach, paths are repeatedly sampled in the MDP directed by a heuristc, and VI is applied only for the states on these paths. Moreover, upper and lower bounds are maintained for each state and the VI operator is applied on them separately. The difference between the upper and lower bounds for every state is a natural measure of error. In addition to this standard approach, we equip our algorithm to detect when a path gets stuck in some end-component, in the lines of [1]. The end component is then collapsed into a representative state in a collapsed MDP, but unlike earlier, we do not wait for the stopping criterion to be satisfied while running VI on the end-component. The core idea of this approach is that if the probability of reaching a state (or a MEC representative state) is quite low, then we might not need to explore it (or obtain an ε-precise solution in the MEC). Our algorithm is able to detect when the solution in a MEC needs to be refined. It continues alternating between propagating bounds in the collapsed MDP and refining the values in the MECs through running VI locally. Furthermore, the biggest advantage of this algorithm, as in BRTDP, is that not all states need

Submitted to:
QAPL 2017

to be explored in order to get approximation guarantees.

Finally, we present the results of our benchmark against MultiGain [2], the only tool we are aware of, which can solve mean-payoff objectives with guarantees. Our results show that depending on the underlying structure of the MDP, our approaches behave comparable in performance. But on some large models, the second approach is able to obtain a solution within seconds whereas standard methods runs out of memory. In every non-trivial example, both our methods significantly outperform the Linear Programming based approach of MultiGain.

The manuscript of this work has been recently submitted.

References

[1] Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Kretínský, Marta Z. Kwiatkowska, David Parker & Mateusz Ujma (2014): *Verification of Markov Decision Processes Using Learning Algorithms*. In: *ATVA*, Springer, pp. 98–114, doi:10.1007/978-3-319-11936-6_8. Available at `http://dx.doi.org/10.1007/978-3-319-11936-6_8`.

[2] Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt & Antonín Kucera (2015): *MultiGain: A Controller Synthesis Tool for MDPs with Multiple Mean-Payoff Objectives*. In: *TACAS*, pp. 181–187, doi:10.1007/978-3-662-46681-0_12. Available at `http://dx.doi.org/10.1007/978-3-662-46681-0_12`.

[3] H. Brendan McMahan, Maxim Likhachev & Geoffrey J. Gordon (2005): *Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees*. In: *ICML*, pp. 569–576, doi:10.1145/1102351.1102423. Available at `http://doi.acm.org/10.1145/1102351.1102423`.

[4] Martin L. Puterman (2014): *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons.