Third International Workshop on Synthesis of Complex Parameters

SynCoP 2016

Pre-proceedings

Benoît Delahaye, Loïg Jezequel, and Jiří Srba

Contents

Preface

Benoît Delahaye
Université de Nantes, LINA CNRS UMR 6241, Nantes, France
benoit.delahaye@univ-nantes.fr

Jiří Srba
Aalborg University, Aalborg, Denmark
srba@cs.aau.dk

This volume contains the pre-proceedings of the 3rd International Workshop on Synthesis of Complex Parameters (SynCoP'16). The workshop was held in Eindhoven, The Netherlands on April 3rd, 2016, as a satellite event of the 19th European Joint Conferences on Theory and Practice of Software (ETAPS'16).

SynCoP aims at bringing together researchers working on verification and parameter synthesis for systems with discrete or continuous parameters, in which the parameters influence the behavior of the system in ways that are complex and difficult to predict. Such problems may arise for real-time, hybrid or probabilistic systems in a large variety of application domains. The parameters can be continuous (e.g., timing, probabilities, costs) or discrete (e.g., number of processes). The goal can be to identify suitable parameters to achieve desired behavior, or to verify the behavior for a given range of parameter values.

The scientific subject of the workshop covers (but is not limited to) the following areas:

- parameter synthesis,
- parametric model checking,
- regular model checking,
- robustness analysis,
- parametric logics, decidability and complexity issues,
- formalisms such as parametric timed and hybrid automata, parametric time(d) Petri nets, parametric probabilistic (timed) automata, parametric Markov decision processes, networks of identical processes,
- interactions between discrete and continuous parameters, and
- tools and applications to major areas of computer science, biology and control engineering.

Program

This volume contains five contributions: two invited talks and the three papers accepted for presentation at the workshop. The second invited talk was a joint talk with the Cassting'16 workshop. The two invited talks are:

- What Population Reveals about Individual Cell Identity: Single-Cell Parameter Estimation of Models of Gene Expression in Yeast (Gregory Batt)
- Parameterized Verification of Distributed Broadcast Protocols (Giorgio Delzanno)

The workshop received four submissions, three of which were accepted. Each regular paper was reviewed by at least four different reviewers. The three regular papers are:

- Rate Reduction for State-labelled Markov Chains with Upper Time-Bounded CSL Requirements (Bharath Siva Kumar Tati and Markus Siegle)
- Parametric, Probabilistic, Timed Resource Discovery System (Camille Coti)
- Weighted Branching Simulation Distance for Parametric Weighted Kripke Structures (Anders Mariegaard, Kim Guldstrand Larsen and Louise Foshammer)

Furthermore, six informal presentation were given at the workshop:

- Recent Advances in Precise Parameter Synthesis for Continuous-Time Markov Chains (Milan Češka, Nicola Paoletti, Luboš Brim and Marta Kwiatkowska)
- Parameter Synthesis for Parametric Interval Markov Chains (Benoit Delahaye, Didier Lime and Laure Petrucci)
- Synthesis of Shared Control Protocols (Nils Jansen and Ufuk Topcu)
- Language Emptiness of Continuous-Time Parametric Timed Automata (Nikola Beneš, Peter Bezděk, Kim G. Larsen and Jiří Srba)
- Integer-Complete Synthesis for Bounded Parametric Timed Automata (Étienne André)
- Quantitative Program Synthesis Based on Linear Operator Semantics (Herbert Wiklicky)

Support and Acknowledgement

SynCoP 2016 is partially supported by project PACS ANR-14-CE28-0002.

SynCoP has been organized as a satellite event of ETAPS'16. We thank the authors for their contributions, the program committee members for reviewing and selecting the papers, the invited speakers for accepting our invitations and providing inspiring presentations and the ETAPS organizing committee Jan Friso Groote, Erik de Vink and Anton Wijs for their support. We would also like to thank Étienne André and the SynCoP Steering Committee for their support.

Finally, we would like to thank the editorial board of the Electronic Proceedings in Theoretical Computer Science, and in particular Editor-in-Chief Rob van Glabbeek for his support.

In Aalborg and Nantes,

Benoît Delahaye and Jiří Srba

Program Committee

Organizers and Program Chairs

Benoît Delahaye	Nantes, France
Jiří Srba	Aalborg, Denmark

Program Committee

Nikola Beneš	Brno, Czech Republic
Nathalie Bertrand	Rennes, France
Alexandre Donzé	Berkeley, USA
Goran Frehse	Grenoble, France
Peter Habermehl	Paris, France
Holger Hermanns	Saarland, Germany
Joost-Pieter Katoen	Aachen, Germany
Marta Kwiatkowska	Oxford, UK
Radu Mardare	Aalborg, Denmark
Wojciech Penczek	Warszawa, Poland
Karin Quaas	Leipzig, Germany
Olivier H. Roux	Nantes, France
Ocan Sankur	Rennes, France
Tayssir Touili	Villetaneuse, France
Lijun Zhang	China

Proceedings chair

Loïg Jezequel	Nantes, France

Steering Committee

Parosh Abdulla	Uppsala, Sweden
Étienne André	Villetaneuse, France
Kim G. Larsen	Aalborg, Denmark
Didier Lime	Nantes, France
Wojciech Penczek	Warszawa, Poland
Laure Petrucci	Villetaneuse, France

Invited papers

What population reveals about individual cell identity: Single-cell parameter estimation of models of gene expression in yeast

Gregory Batt
INRIA Paris-Rocquencourt
Paris, France
gregory.batt@inria.fr

Significant cell-to-cell heterogeneity is ubiquitously observed in isogenic cell populations. Consequently, parameters of models of intracellular processes, usually fitted to population-averaged data, should rather be fitted to individual cells to obtain a population of models of similar but non-identical individuals. Here, we propose a quantitative modeling framework that attributes specific parameter values to single cells for a standard model of gene expression. We combine high quality single-cell measurements of the response of yeast cells to repeated hyperosmotic shocks and state-of-the-art statistical inference approaches for mixed-effects models to infer multidimensional parameter distributions describing the population, and then derive specific parameters for individual cells. The analysis of single-cell parameters shows that single-cell identity (e.g. gene expression dynamics, cell size, growth rate, mother-daughter relationships) is, at least partially, captured by the parameter values of gene expression models (e.g. rates of transcription, translation and degradation). Our approach shows how to use the rich information contained into longitudinal single-cell data to infer parameters that can faithfully represent single-cell identity.

Submitted to:
SynCoP 2016

Parameterized verification of distributed broadcast protocols

Giorgio Delzanno
DIBRIS
Genova, Italy
giorgio.delzanno@unige.it

We report on recent research lines related to a graph-based approach to parameterized verification of formal models of distributed algorithms and protocols. Verification algorithms for restricted classes of models exploit finite-state abstractions, symbolic representations based on graph orderings, the theory of well-structured transition systems, and reachability algorithms based on labeling procedures.

Submitted to:
SynCoP 2016

Regular papers

Rate Reduction for State-labelled Markov Chains with Upper Time-bounded CSL Requirements

Bharath Siva Kumar Tati Markus Siegle
Universität der Bundeswehr München
Germany
Bharath.Tati@unibw.de Markus.Siegle@unibw.de

This paper presents algorithms for identifying and reducing a dedicated set of transition rates of a state-labelled continuous-time Markov chain model. The purpose of the reduction is to make states to satisfy a given requirement, specified as a CSL upper time-bounded Until formula. We distinguish two different cases, depending on the type of probability bound. A natural partitioning of the state space allows us to develop possible solutions, leading to simple algorithms for both cases.

1 Introduction

In a production plant, there can be the requirement that "once started, the production process should be completed within 1 hour in 95% of all cases", or that "the probability of an alarm during the first 30 min of operation should be at most 5%". If the system is modelled as a state-labelled continuous-time Markov chain (SMC) and the requirements are formulated with the help of continuous stochastic logic (CSL), they can be verified automatically by stochastic model checking [1], supported by efficient tools such as the probabilistic model checker PRISM [10] or MRMC [9].

This paper addresses the question of how to improve a given system (also called plant $\mathscr{P}$) when it has been found that $\mathscr{P}$ violates such a given formal requirement. In an earlier paper [13] we presented solutions for the case of *untimed* probabilistic requirements, and building on that work we now present solutions for the case of *upper time-bounded Until*-type requirements (without nested or multiple *Until* operators). In general, one could think of many ways of how to modify a system in order to make it satisfy a given requirement. We decided to restrict ourselves to *rate reduction*, which means that some of the system's transition rates may be reduced, but the structure of the system remains untouched. We only allow rate *reductions*, as opposed to increasing any rates, motivated by the fact that it is usually easily possible to slow down some technical process (machine, processor, etc.), while it may not be possible to accelerate it. We work with a partitioning of the state space into classes, based on the requirement at hand. Our strategy then is to reduce all transition rates between some source class and target class by a *common reduction factor*. Depending on the case, different sets of transitions may be reduced by different reduction factors to achieve the goal. Before the start of the adaptation process, the value of all reduction factors is 1, and in the end all reduction factors will be still at most 1, but strictly greater than 0 (which means that no transitions are completely disabled). Throughout, our intention is to make as many as possible states of $\mathscr{P}$ satisfy the user requirement, but it is not always possible to make all states satisfying. This paper develops simple, intuitive algorithms, which are first motivated by examples. We show the correctness of the algorithms (while pointing out the limitations of Algorithm 2) and also analyze their complexity.

Related work: A related topic is model checking of parametric Markov chains, where reachability probabilities take the form of rational functions [5, 6, 7]. Here the goal is to find valid parameter values

Submitted to:
SynCoP 2016

in a multi-dimensional search space, as described in [8], where a discretization strategy was proposed together with refinement and/or sampling. Synthesizing optimal rate parameter values is also the goal of [3] (in the context of Markov models of biochemical systems), where time-bounded properties are considered. Closely related is the so-called *model repair* problem which occurs when a system violates a given requirement, which is to be fixed by modifying transition parameters while at the same time keeping cost at a minimum. Model repair has been addressed e.g. in [2, 4, 12], for parametric DTMC or MDP models, where solutions are obtained with the help of nonlinear optimization, sampling/refinement or greedy strategies. Our approach described here is different from all of the above, since we work with CTMC models which are a priori not parametric, but come with fixed rates. Once some requirement is violated, we seek to identify sets of transitions and reduce their rates by a common reduction factor in order to make as many states as possible satisfy the requirement. We do currently not consider the cost of rate reduction, and we restrict ourselves to simple algorithms which avoid expensive multi-dimensional parameter searches.

The rest of the paper is structured as follows: Sec. 2 provides the basic definitions and recalls an earlier result for untimed requirements. In Sec. 3, an example is elaborated on in order to explain the idea of rate reduction. It illustrates the benefits, but also the limitations of the proposed approach. The general algorithms are discussed in Sec. 4, which also includes their complexity analysis. Finally, Sec. 5 summarizes the main findings of the paper and touches on possible future work.

2 Preliminaries

A State labelled Markov chain (SMC) is defined as follows:

Definition 1 (SMC) *A SMC $\mathscr{P}$ is a tuple $(S_{\mathscr{P}}, R_{\mathscr{P}}, L_{\mathscr{P}})$ where*

- *$S_{\mathscr{P}}$ is a finite set of states*
- *$R_{\mathscr{P}} : S_{\mathscr{P}} \times S_{\mathscr{P}} \mapsto \mathbb{R}_{\geq 0}$, is the transition function (rate matrix)*
- *$L_{\mathscr{P}} : S_{\mathscr{P}} \to 2^{AP}$ is a state labelling function, where AP is a finite set of atomic propositions*

This definition does not impose any special structural conditions (such as irreducibility) on the state graph of the Markov chain. A finite timed *path* σ in a SMC $\mathscr{P}$ is a finite sequence $\sigma = [(s_0, t_0), (s_1, t_1), \cdots, (s_{n-1}, t_{n-1}), s_n] \in (S_{\mathscr{P}} \times \mathbb{R}_{>0})^* \times S_{\mathscr{P}}$, and with $Paths(s)$ we denote the set of all finite paths originating from state s. Probabilities are assigned to sets of finite timed paths by the usual cylinder set construction on sets of infinite timed paths. In order to specify user requirements and characterize execution paths of SMCs, we use a subset of CSL (continuous stochastic logic) [1].

Definition 2 (CSL with upper time bound) *The grammar for CSL state formulas Φ, Φ' and path formulas φ is given as:*

$$\Phi ::= q \,|\, \neg\Phi \,|\, \Phi \vee \Phi \,|\, P_{\sim b}(\varphi), \qquad \varphi ::= \Phi' \,\mathscr{U}^{\leq t}\, \Phi', \qquad \Phi' ::= q \,|\, \neg\Phi' \,|\, \Phi' \vee \Phi'$$

In the definition, $q \in AP$ is an atomic proposition, $\neg$ denotes negation, $\vee$ denotes disjunction, $b \in (0,1)$ is a probability value, and $\sim \in \{\leq, \geq\}$ a comparison operator. $P_{\sim b}(\varphi)$ asserts that the probability measure of the set of paths satisfying φ meets the bound given by $\sim b$. The path formula φ is constructed using the *Until* ($\mathscr{U}$) operator and an upper time bound $t > 0$. Note that we do not consider CSL requirements with nested *Until* operators (that's why we distinguish between Φ and Φ'), since parameter adaptations for *Until* operators at different levels are interdependent in a complex way. For similar reasons we do not consider requirements with multiple *Until* operators (although the grammar in Def. 2 does not exclude them). We also do not consider the CSL *next* operator since it would be rather trivial to handle.

Remark 1 This paper considers probability bounds $b \in (0,1)$ instead of $b \in [0,1]$, since the approach presented here does not aim to turn a non-zero probability into zero, or to turn a probability smaller than one into one. So, we do not treat requirements of the form $P_{\leq 0}(\varphi)$ or $P_{\geq 1}(\varphi)$, and for similar reasons we also do not treat requirements of the form $P_{>0}(\varphi)$ or $P_{<1}(\varphi)$. Furthermore, there is no need to distinguish between $< b$ and $\leq b$ (or $> b$ and $\geq b$), because in the continuous-time setting the probabilities are the same.

Definition 3 (Semantics of time-bounded Until path formula) *The satisfaction relation $\models$ for time-bounded* Until *path formulas is defined as in [1]:*

$$\sigma \models \Phi \,\mathcal{U}^{\leq t}\, \Psi \quad \textit{if} \quad \exists t' \in [0,t].(\sigma@t' \models \Psi \wedge \forall t'' \in [0,t').\sigma@t'' \models \Phi)$$

where $\sigma@t$ denotes the state occupied by the path σ at time t.

The untimed *Until* formula is obtained as: $\Phi \,\mathcal{U}\, \Psi = \Phi \,\mathcal{U}^{<\infty}\, \Psi$. Let $Sat(\Phi)$ denote the set of states satisfying state formula Φ, and let $Pr(s,\varphi) = Pr(\{\sigma \in Paths(s) \mid \sigma \models \varphi\})$ denote the probability of the set of φ-satisfying paths originating in state s. In order to accomplish the process of rate reduction, we will use a partitioning of the SMC state space:

Definition 4 (Partitioning of SMC) *Given an SMC $\mathscr{P}$ and a CSL requirement $\Phi_t = P_{\sim b}(\Phi \,\mathcal{U}^{\leq t}\, \Psi)$. Let $\varphi = \Phi \,\mathcal{U}\, \Psi$, the untimed version of the path formula. Then, states belonging to*

- $Sat(\neg\Phi \wedge \neg\Psi)$ *are placed in class* invalid
- $Sat(\Psi)$ *are placed in class* target
- $Sat(\Phi \wedge \neg\Psi)$ *are placed in class* transit

The transit class is further partitioned as:

- $Pr(s,\varphi) = 1$ *are placed in class* gototarget
- $Pr(s,\varphi) = 0$ *are placed in class* gotoinvalid
- $0 < Pr(s,\varphi) < 1$ *are placed in class* gobothways

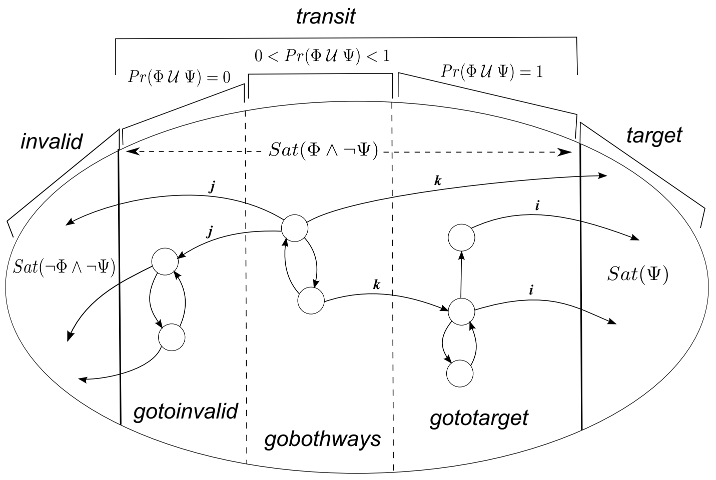
This partitioning is illustrated in Fig. 1[1]. Starting from states of class *gototarget*, the Markov chain will eventually reach the *target* class via Φ-states (almost surely within finite time). Conversely, states of class *gotoinvalid* do not possess any path satisfying the given (untimed, and therefore also time-bounded) *Until* requirement. From states of class *gobothways*, both of these behaviours are possible. During the parameter adaptation procedure, our attention will be on the states of the *transit* class. The Partitioning of the state space can be obtained efficiently, based on the state labelling and by applying standard graph algorithms [11]. Note that the time bound $\leq t$ and the probability bound $\sim b$ in the given CSL formula have no influence on the partitioning.

As a simple but important fact we emphasize that the probability of a state satisfying a time-bounded *Until* property will always be less than or equal the probability of that state satisfying the corresponding *untimed* property, i.e.

$$\forall s.\, \forall t.\, (Pr(s, \Phi \,\mathcal{U}^{\leq t}\, \Psi) \leq Pr(s, \Phi \,\mathcal{U}\, \Psi)) \tag{1}$$

Furthermore, for an SMC $\mathscr{P}$, a subset $X \subseteq S$ of its state set, and a timed or untimed CSL state formula Φ, we introduce the following notation: $(\mathscr{P} \models_X \Phi) \iff (\forall s \in X : s \models \Phi)$. As an example, $\mathscr{P} \models_{gobothways} \Phi$ means that all states from class *gobothways* satisfy the requirement Φ.

[1]For the purpose of this paper, there is no need to distinguish between classes *invalid* and *gotoinvalid*, but we prefer to separate them for reasons of symmetry.



transit
$0 < Pr(\Phi\,\mathcal{U}\,\Psi) < 1$
$Pr(\Phi\,\mathcal{U}\,\Psi) = 0$
$Pr(\Phi\,\mathcal{U}\,\Psi) = 1$
invalid
target
$Sat(\Phi \wedge \neg\Psi)$
$Sat(\neg\Phi \wedge \neg\Psi)$
$Sat(\Psi)$
j
j
k
k
i
i
gotoinvalid
gobothways
gototarget

Figure 1: Partitioning of the state space

For the purpose of rate reduction, we construct a reduced (parametric, see below) SMC $\mathscr{G}$ from SMC $\mathscr{P}$, where certain transition rates are reduced by factors i, j and k and all the states in classes *invalid* and *target* are made absorbing.

Definition 5 (Reduced parametric SMC $\mathscr{G}$) *Given SMC $\mathscr{P}$ as in Def. 1, a CSL requirement $\Phi_t = P_{\sim b}(\Phi\ \mathscr{U}^{\leq t}\ \Psi)$ and three reduction factors (parameters) $0 < i, j, k \leq 1$. The reduced SMC $\mathscr{G}(\Phi, \Psi)$ is defined as a tuple $(S_{\mathscr{G}}, R_{\mathscr{G}}, L_{\mathscr{G}})$ where:*

- *$S_{\mathscr{G}} = S_{\mathscr{P}}$, and the state space partitioning into classes is taken from $\mathscr{P}$*
- $$R_{\mathscr{G}}(s,s') = \begin{cases} 0 & s \in target \vee s \in invalid \\ i \cdot R_{\mathscr{P}}(s,s') & s \in gototarget \wedge s' \in target \\ j \cdot R_{\mathscr{P}}(s,s') & s \in gobothways \wedge s' \in (gotoinvalid \cup invalid) \\ k \cdot R_{\mathscr{P}}(s,s') & s \in gobothways \wedge s' \in (gototarget \cup target) \\ R_{\mathscr{P}}(s,s') & otherwise \end{cases}$$
- $L_{\mathscr{G}} = L_{\mathscr{P}}$

Let T_i denote the set of transitions whose rate is multiplied by reduction factor i (and analogously for T_j and T_k).

Considering the reduction factors i, j and k as variables, the reduced SMC $\mathscr{G}$ is a parametric SMC. Once they are fixed, $\mathscr{G}$ is a standard SMC. Fig. 1 shows how the transition rates between certain state classes will be multiplied by the reduction factors. The purpose of this multiplication will be explained in the course of the paper.

2.1 A result for untimed CSL

In [13], we presented algorithms for the rate reduction problem for untimed *Until* requirements, based on the following principle: In case of an upper probability bound ($\leq b$), all transition rates from *gobothways* to *gototarget* and to *target* are reduced by a global factor of $0 < k \leq 1$. Similarly, in case of a lower probability bound ($\geq b$), all rates from *gobothways* to *gotoinvalid* and to *invalid* are reduced by a factor of $0 < j \leq 1$. The idea is to thereby influence the branching probabilities of states from class *gobothways* in the desired direction. Theorem 3.2 of [13] states that following this strategy, the parameter synthesis problem for untimed *Until* requirements can always be solved for all states of class *gobothways*. We are going to use this result in the sequel, in combination with Eq. (1).

2.2 Binary Search Method (BSM)

In the time-bounded CSL scenario, we use the Binary Search Method (BSM) to approximate the maximum satisfying value of the reduction factor, where we search the range between 0 and 1 up to a precision of a given $\varepsilon > 0$. Since a closed-form solution for the transient state probabilities for a parametric system is not available, we use an approximation via BSM with multiple evaluations using uniformisation. At the start of the reduction process, the search interval is $(0,1]$. BSM continues to halve the search interval until its width is at most the predefined precision ε. In pseudo code, BSM reads as follows, the initial call being `BSM(0,1)`:

```
BSM(lower, upper)
  {
   while upper-lower > epsilon
   {
     middle = (lower + upper) / 2;
         if "middle satisfies the requirement" then
           lower = middle;
         else
           upper = middle;
   }
   return lower;
  }
```

As a result, BSM returns the lower bound of the final search interval, where the search is considered unsuccessful if that returned value is zero.

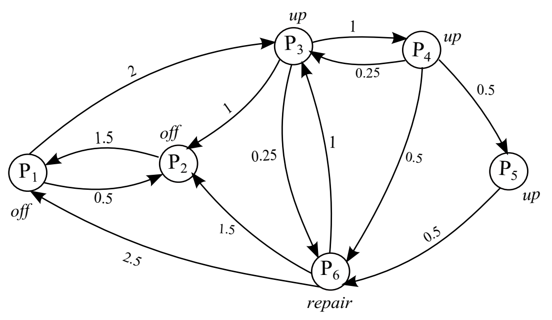
3 Example

Before giving the general rate reduction algorithms, we will explain our method with the help of an example for the two cases of upper time-bounded CSL *Until* requirements for the model SMC $\mathscr{P}$ given in Fig. 2. Plant $\mathscr{P}$ (SMC) models an abstract machine with 6 states, which can be *off*, *up* or under *repair*. When the machine is *up*, it can go into *off* or *repair* with some predefined rates. We present our new heuristics/algorithms, partly based on earlier heuristics given in our paper [13], to find reduction factors i, j and k (in case $\Phi = P_{\sim b}(up\ \mathscr{U}^{\leq t}\ repair)$ is violated). In order to create the reduced parametric SMC $\mathscr{G}$, we need to partition $\mathscr{P}$ by using the CSL path formula $\varphi = up\ \mathscr{U}\ repair$ according to Def. 4. SMC $\mathscr{G}$ for this example is given in Fig. 3, which also shows the partitioning. Note that for this example, class *gotoinvalid* is empty.

3.1 Case 1

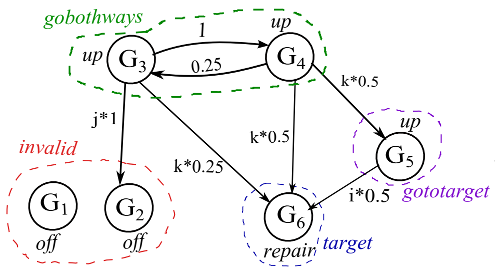
We have an abstract model as shown in $\mathscr{P}$ and the user property Φ_1 in this case shall be given as in Eq. (2).

$$\Phi_1 := P_{\leq 0.2}(\varphi), \text{where } \varphi = up\ \mathscr{U}^{\leq 5}\ repair \tag{2}$$



$$R_{\mathcal{P}} = \begin{pmatrix} 0 & 0.5 & 2 & 0 & 0 & 0 \\ 1.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0.25 \\ 0 & 0 & 0.25 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \\ 2.5 & 1.5 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Figure 2: Plant $\mathscr{P}$ along with rate matrix $R_{\mathscr{P}}$



$$R_{\mathcal{G}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & j*1 & 0 & 1 & 0 & k*0.25 \\ 0 & 0 & 0.25 & 0 & k*0.5 & k*0.5 \\ 0 & 0 & 0 & 0 & 0 & i*0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 3: Reduced SMC $\mathscr{G}$ and its rate matrix

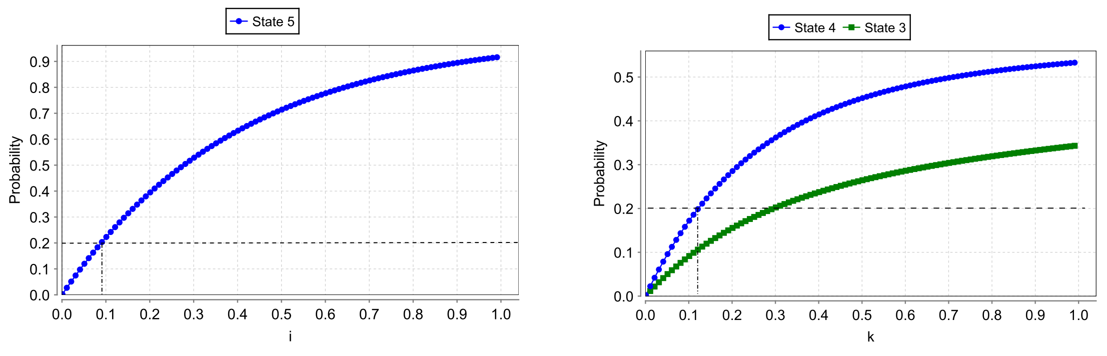
Model checking with PRISM [10], we get the probabilities for each state as follows:

$$Pr(P_1, \varphi) = 0 \le 0.2 \tag{3}$$
$$Pr(P_2, \varphi) = 0 \le 0.2 \tag{4}$$
$$Pr(P_3, \varphi) = 0.47323 > 0.2 \tag{5}$$
$$Pr(P_4, \varphi) = 0.83443 > 0.2 \tag{6}$$
$$Pr(P_5, \varphi) = 0.91791 > 0.2 \tag{7}$$
$$Pr(P_6, \varphi) = 1 > 0.2 \tag{8}$$

From the above probability values, we can see that $Sat(\Phi_1) = \{P_1, P_2\}$. Our focus is only on classes *gobothways* and *gototarget*, as the states from the other classes trivially satisfy/violate the requirement Φ_1. The user property Φ_1 is violated for all the states inside our classes of interest, i.e., P_3, P_4 and P_5. Hence, the process of rate reduction is required. Before the start of adaptation procedure, the values of i, k and j are equal to 1. Now, since the probability of reaching the *target* class within the specified time bound needs to be reduced, we have to slow down the rates going towards class *target*, i.e., adapt reduction factors i and k, such that the probabilities of the states in classes *gototarget* and *gobothways* to satisfy φ will fall below 0.2.

Initially, we adapt the i value, because the probability of the states in *gototarget* class will not get affected by adapting the k value, whereas the vice versa is not true. To graphically show the process of finding the satisfying range of the reduction factor i, we plotted[2] the probabilities at equidistant discrete points of i. The curve for state P_5 of *gototarget* class is shown in the Fig. 4a.

[2]All the graphs are created using PRISM tool [10].



(a) Adapting i for State 5; y-axis is $Pr(P_5, \varphi)$ (b) Adapting k for states 3 and 4, for fixed $i = 0.089$; y-axis is $Pr(\star, \varphi)$

Figure 4: Graphs for Case 1

Finding the reduction factors by solving a set of equations as in the case of untimed CSL is not feasible[3] for the time-bounded *Until* case. Hence, we used BSM to approximately find the satisfaction range of i, which is $0 < i \leq 0.089$.

Now, we will focus on the states from class *gobothways* (states P_3 and P_4). In order to make the states from this class satisfying, we need to adapt the k value. In case 1, BSM considers the state with highest probability (here it is state 4) within the class. While adapting k, we keep the i value fixed to its maximum from the range which we found earlier. By doing so, we obtain the range of k to satisfy Eq. (2) to be $(0 < k \leq 0.122)$. The graph in Fig. 4b, shows the probability plots for states 3 and 4 of $\mathscr{P}$ while reducing factor k. The k value can also be read from the graph in Fig. 4b, the curve for state 4 (blue curve) falls below the required probability bound 0.2 at $k = 0.122$. Hence, such a k range will be the solution for the whole $\mathscr{P}$. After applying the reduction factors i and k, the probabilities of the states in *gobothways* and *gototarget* class modify as follows, and therefore satisfy the user requirement.

$$Pr(P_3, \varphi) = 0.10675 \leq 0.2, \qquad Pr(P_4, \varphi) = 0.19988 \leq 0.2, \qquad Pr(P_5, \varphi) = 0.19948 \leq 0.2$$

From Eq. (1), we know that the probability for a time-bounded user requirement is always less than or equal to the probability of untimed user requirement. Therefore, since the untimed problem can be always solved (see Sec. 2.1) we can conclude that the time-bounded variant can also always be solved.

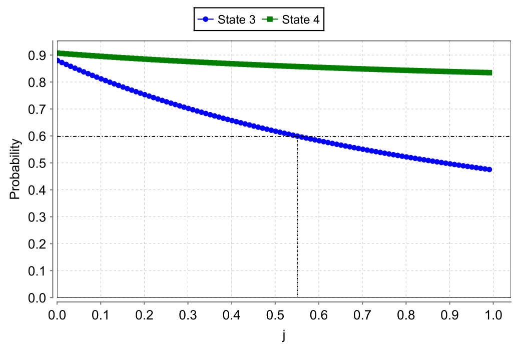
3.2 Case 2

The user requirement in case 2 shall be as in Eq. (9).

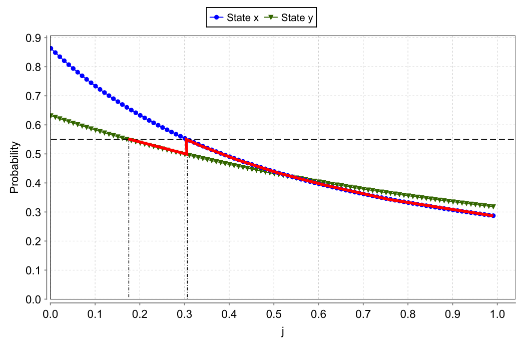
$$\Phi_2 := P_{\geq 0.95}(\varphi), \text{where again } \varphi = up\ \mathscr{U}^{\leq 5}\ repair \qquad (9)$$

The actual probabilities of each state for φ are the same as shown in equations from (3) to (8). We can see that probabilities of state P_3 and P_4 (and P_5) are lower than the required probability bound i.e., ≥ 0.95, and thus the user requirement is violated for those states. In case 2, our interest will be only on states from *gobothways* class for the following reasons: 1) All states from classes *invalid* and *gotoinvalid* trivially

[3]The transient probabilities for a fixed value of the reduction factor are computed numerically via uniformisation, a parametric closed-form solution is not available.



(a)



(b)

Figure 5: (a) Adapting j for states 3 and 4 for Case 2; y-axis is $Pr(\star, \varphi)$,
(b) Intersecting curves while adapting j (for a different example)

cannot satisfy the requirement. 2) States from *gototarget* may or may not satisfy the requirement, but if they don't, nothing can be improved by reducing any rate (one would have to increase the rate from *gototarget* to *target*, but this is not one of our options).

Here, since the probability should be increased beyond 0.95, the transition rates from *gobothways* to *gotoinvalid* or *invalid* should be reduced, in order to make the transitions between *gobothways* to *target* more likely to happen. For BSM, we have to reduce j, and so we choose the state with least probability (here it is state 3) within the *gobothways* class. The graph in Fig. 5a shows the probability plots of states 3 and 4 for φ. As it can be seen, while j approaches 0, the probability for φ reaches a value of 0.89 (but it does not reach 1.0 due to the time bound), but as per the requirement it should be greater than 0.95. In this case, our algorithm fails to find a satisfying solution.

However, let us take another user requirement for the same case 2 as given in Eq. (10),

$$\Psi_2 := P_{\geq 0.6}(\varphi), \text{where again } \varphi = up \ \mathcal{U}^{\leq 5} \ repair \tag{10}$$

Notice the change in the probability bound. Now, from the graph in Fig. 5a, we can observe that the satisfying range of j to be approximately $0 < j \leq 0.548$, where the curve is above required probability bound (i.e., ≥ 0.6). For the current j value, check all the states from *gobothways* class just to make sure they satisfy Φ_2. In general, reducing j makes the unbounded requirement $Pr(\Phi_1 \ \mathcal{U} \ \Phi_2)$ larger (even reaches prob. 1), but it slows down the process. Hence the existence of a solution in Case 2 depends on the combination of time and probability bounds given in the user requirement.

4 Algorithms

This section explains the general rate reduction algorithms for cases 1 and 2 of upper time-bounded CSL *Until* properties. But before we present the algorithms, we need to discuss the important phenomenon of intersecting curves. In the example from Sec. 3, the probability curves for different states never showed any intersection points. However, in general the probability curves for different states may intersect, as we can observe in other examples. Fig. 5b shows such an example of intersecting curves for varying reduction factor j, but intersection of curves may also occur when varying factors i or k. For that reason, once the value of the reduction factor has been determined for the state with the highest (in case 1) or lowest (in case 2) original probability, one needs to recheck all the other states from the class. As long as at least one of them violates the requirement, the value of the reduction factor has to be reduced further. This is the reason why our algorithms in this section employ while-loops for determining the reduction factors.

4.1 Case 1

The CSL requirement in this case is given as $\Phi_1 := P_{\leq b}(\Phi\ \mathcal{U}^{\leq t}\ \Psi)$, and the state space of $\mathcal{P}$ is partitioned according to Def. 4. For any state s from *invalid* $\cup$ *gotoinvalid*, it holds that $Pr(s, \Phi\ \mathcal{U}^{\leq t}\ \Psi) = 0$, and since $0 < b$ those states trivially satisfy the requirement. Similarly, for states from class *target* the probability to follow a satisfying path is 1, thus *target*-states trivially violate the requirement (and nothing can be done about that).

Therefore, the algorithm will only address states from classes *gobothways* and *gototarget*. Based on the fact stated by Eq. (1) and the fact that the untimed problem can always be solved (see Sec. 2.1), the problem for upper time-bounded *Until* as in Φ_1 can always be solved for these two classes. The algorithm proceeds by first adapting factor i on transitions from *gototarget* to *target*. It starts to reduce the probability for s_h, the state from class *gototarget* with the highest probability. Since the probability curves of different states within class *gototarget* may intersect with each other, once a satisfying i value is found for s_h, we need to check whether all the other states of class *gototarget* also satisfy Φ_1. As long as any of the states violates Φ_1, we continue the process of reducing i (while loop) by selecting the state with highest probability at that instance. Once all states from class *gototarget* satisfy Φ_1 factor i will remain fixed. As a second step, a similar procedure is followed for class *gobothways*, where the algorithm reduces factor k on transitions from *gobothways* to *gototarget* $\cup$ *target*. Again, because of possible intersection of probability curves, the process needs to be continued for class *gobothways* as motivated above. Algorithm 1 formalizes this rate reduction process for Case 1.

4.2 Case 2

In this case, the general form of CSL property to be checked is given as $\Phi_2 := P_{\geq b}(\Phi\ \mathcal{U}^{\leq t}\ \Psi)$. As already stated in Case 1, for any state s from *invalid* $\cup$ *gotoinvalid*, it holds that $Pr(s, \Phi\ \mathcal{U}^{\leq t}\ \Psi) = 0$, and since $0 \ngeq b$ those states trivially violate the requirement (and nothing can be done about it). Similarly, for states from class *target*, the probability to follow a satisfying path is 1, thus *target*-states trivially satisfy the requirement. For any state from class *gototarget*, the probability of satisfying paths may be above or below the bound b. For states of the former type, the requirement is satisfied, but for states of the latter type, it is not and nothing can be done about it, since one would have to accelerate the paths towards *target*, which is not possible (since we only allow rate reductions).

Hence, we only focus on the remaining class *gobothways* for rate reduction. If the given Φ_2 is violated, then the branching probability (to eventually reach class *target* instead of class *invalid*) is too low, or the speed of moving to class *target* is too slow. From the result for untimed *Until* (cf. Sec. 2.1) we know that the branching probability can be increased to any desired value (arbitrarily close to 1) by the reduction factor j. Following the similar strategy in case of time-bounded *Until* may lead to a solution, but not always (because of the speed being too slow), as demonstrated by example in Sec. 3.2. So the strategy is to try reducing factor j, and return j if a satisfying value is found, else return *fail*. Again, since probability curves may intersect, the search for a satisfying value of j needs to be performed in a while loop. The general algorithm for this case is given in Algorithm 2.

Algorithm 1 Rate reduction for $\Phi_1 := P_{\leq b}(\Phi\, \mathscr{U}^{\leq t}\, \Psi)$

```
Input: SMC $\mathscr{P}$ and time-bounded CSL Until property $\Phi_1$.
Output: Satisfying $i$, $k$ reduction factors and corresponding sets of transitions $T_i, T_k$
if $gobothways \cup gototarget = \emptyset$ then
    quit                                          ▷ No states whose probabilities can be modified
else
    if $\mathscr{P} \models_{gobothways \cup gototarget} \Phi_1$ then      ▷ No need for rate reduction
        quit
    else
        $\mathscr{G} = (S_\mathscr{G}, R_\mathscr{G}, L_\mathscr{G})$; $i = 1$; $k = 1$    ▷ Construct reduced SMC (Def. 5), initialize red. factors
        while TRUE do                                          ▷ Find i for class gototarget
            for current $i$, find $s_h = \arg\max_s \{Pr(s, \Phi \mathscr{U}^{\leq t} \Psi) \mid s \in gototarget\}$
            Apply BSM to $s_h$ to find satisfying $i$
            Check all $s \in gototarget$ for current $i$
            if any of the states still violates $\Phi_1$ then
                continue
            else
                fix factor $i$ and corresponding set $T_i$ ; break
            end if
        end while
        while TRUE do                                          ▷ Find k for class gobothways
            for current $k$ value, find $s'_h = \arg\max_s \{Pr(s, \Phi \mathscr{U}^{\leq t} \Psi) \mid s \in gobothways\}$
            Apply BSM to $s'_h$ to find satisfying $k$
            Check all $s \in gobothways$ for current $k$
            if any of the states still violates $\Phi_1$ then
                continue
            else
                break
            end if
        end while
        return factors $i$, $k$ and corresponding sets $T_i$, $T_k$
    end if
end if
```

4.3 A comment on optimality of Algorithm 2

As argued in Sec. 4.1, Algorithm 1 always succeeds in making all states of *gobothways* ∪ *gototarget* satisfying, whereas Algorithm 2 does not always find as solution. If Algorithm 2 fails, then there does not exist a common reduction factor j that will make all the states of class *gobothways* to satisfy Φ_2. In this case, some states from *gobothways* may indeed satisfy Φ_2, but not all of them. Algorithm 2 is not always optimal in the following sense: There exist cases where the algorithm fails but where it would be possible to use a more general form of rate reduction in order to make more (or maybe even all) states of *gobothways* to satisfy Φ_2. In those cases, it is possible to further increase the probability of certain states from *gobothways* to satisfy the path formula $\Phi\, \mathscr{U}^{\leq t}\, \Psi$ by reducing not only the rates from class *gobothways* to class *gotoinvalid* ∪ *invalid* (as Algorithm 2 does), but also reducing the rates of

Algorithm 2 Rate reduction for $\Phi_2 := P_{\geq b}(\Phi\, \mathscr{U}^{\leq t}\, \Psi)$

```
**Input:** SMC $\mathscr{P}$ and time-bounded CSL *Until* property $\Phi_2$.
**Output:** Satisfying $j$ reduction factor and corresponding set of transitions $T_j$, or *fail*
if $gobothways = \emptyset$ then
    quit                                          ▷ No states whose probabilities can be modified
else
    if $\mathscr{P} \models_{gobothways} \Phi_2$ then                          ▷ No need for rate reduction
        quit
    else
        $\mathscr{G} = (S_{\mathscr{G}}, R_{\mathscr{G}}, L_{\mathscr{G}})$; $j = 1$          ▷ Construct reduced SMC (Def. 5), initialize red. factor
        while TRUE do                                              ▷ Find j for class gobothways
            for current $j$ value find, $s_l = \arg\min_s \{Pr(s, \Phi\, \mathscr{U}^{\leq t}\, \Psi) \mid s \in gobothways\}$
            For state $s_l$ apply BSM to reduce $j$
            if solution found then                                                  ▷ i.e., if $j > 0$
                Check all $s \in gobothways$ with current $j$ value
                if any of the states still violates $\Phi_2$ then
                    continue
                else
                    return factor $j$ and corresponding set $T_j$
                end if
            else
                return *fail*
            end if
        end while
    end if
end if
```

certain transitions *among* the states of class *gobothways* (i.e. transitions within *gobothways*). However, selecting individual transitions among states of class *gobothways* for rate reduction is a very difficult task for which there are currently no known efficient algorithms or even heuristics, so this is beyond the scope of this paper.

4.4 Complexity

We assume a SMC with N states and $M = O(N^2)$ transitions. The time complexity for model checking time-bounded *Until* is the same as for CTMC transient solution by uniformisation, which is known to be $O(M \cdot q \cdot t)$, where q is the uniformisation rate and t is the time bound [1] (note that the uniformisation rate, which is at least the maximum of the states exit rates, actually decreases as we reduce transitions rates). For constructing the reduced SMC $\mathscr{G}$ the state classes need to be determined and the reduction factors need to be inserted, all of which can be done in time $O(M)$. Given the desired precision ε, each run of BSM requires $O(\log_2 \frac{1}{\varepsilon})$ steps. Therefore, the overall time complexity of Algorithms 1 and 2 is $O(M \cdot q \cdot t \cdot N \cdot \log_2 \frac{1}{\varepsilon})$, where the factor N reflects the iterations of the while loop(s).

5 Conclusion

In this paper, we have considered systems (also called "plants") specified as state-labelled Markov chains, and user requirements given as upper time-bounded CSL formulas (without multiple or nested *Until* operators). Whenever a user requirement is violated by some or all states of the plant, we try to *repair* the plant by reducing some dedicated sets of its transition rates, such that eventually the user requirement will be satisfied. We only allow for a slowdown of transition rates (but without completely disabling any transitions), since in most practical situations increasing the transition rates is not feasible. Upon model checking, some states in the plant's state space may already satisfy the requirement, whereas some others may not. Some states will have constant probability which cannot be modified by reducing the rates. Depending on the type of probability bound for the *Until* formula, we identified two possible cases for which we have devised simple and intuitive algorithms along with necessary and sufficient conditions for the solutions to exist. The algorithms partition the state space into different classes and find the appropriate sets of transitions between those classes whose rates need to be reduced, as well as the respective reduction factors.

Our ideal goal is to identify the maximum number of states which can be made to satisfy the user requirement. We have shown that Algorithm 1 (for the upper probability bound) always achieves this goal of maximality, but as discussed in Sec. 4.3, Algorithm 2 (for the lower probability bound) is not always optimal.

In this paper, we have not addressed lower time-bounded CSL user requirements, i.e. the case of $\Phi = P_{\sim b}(\Phi\ \mathscr{U}^{\geq t}\ \Psi)$. Based upon the probability bound, lower time-bounded CSL requirements can be further divided into two categories to which we refer as cases 3 and 4, and we are planning to elaborate on these cases in a forthcoming paper. Another interesting topic for future work would be to consider time-bounded CSL formulas with multiple or nested *Until* operators, a difficult problem, as already hinted at in Sec. 2. Furthermore, there is the interesting open question of how to repair a given plant at the level of the high-level modelling formalism, instead of at the level of the Markov chain.

References

[1] C. Baier, B. R. Haverkort, H. Hermanns & J.-P. Katoen (2003): *Model-Checking Algorithms for Continuous-Time Markov Chains*. *IEEE Trans. Software Eng.* 29(6), pp. 524–541, doi:10.1109/TSE.2003.1205180.

[2] E. Bartocci, R. Grosu, P. Katsaros, C.R. Ramakrishnan & S.A. Smolka (2011): *Model Repair for Probabilistic Systems*. In: *TACAS 2011*, *Lecture Notes in Computer Science* 6605, Springer, pp. 326–340, doi:10.1007/978-3-642-19835-9_30.

[3] M. Ceska, F. Dannenberg, M. Z. Kwiatkowska & N. Paoletti (2014): *Precise Parameter Synthesis for Stochastic Biochemical Systems*. In: *Computational Methods in Systems Biology - 12th International Conference, CMSB*, pp. 86–98, doi:10.1007/978-3-319-12982-2_7.

[4] T. Chen, E.M. Hahn, T. Han, M. Kwiatkowska, H. Qu & L. Zhang (2013): *Model Repair for Markov Decision Processes*. In: *Theoretical Aspects of Software Engineering (TASE), 2013*, pp. 85–92, doi:10.1109/TASE.2013.20.

[5] C. Daws (2004): *Symbolic and Parametric Model Checking of Discrete-Time Markov Chains*. In: *Theoretical Aspects of Computing - ICTAC 2004*, *Lecture Notes in Computer Science* 3407, Springer Berlin Heidelberg, pp. 280–294, doi:10.1007/978-3-540-31862-0_21.

[6] E. M. Hahn, H. Hermanns, B. Wachter & L. Zhang (2010): *PARAM: A Model Checker for Parametric Markov Models*. In: *Computer Aided Verification, 22nd International Conference, CAV*, pp. 660–664, doi:10.1007/978-3-642-14295-6_56.

[7] E. M. Hahn, H. Hermanns & L. Zhang (2011): *Probabilistic reachability for parametric Markov models*. *STTT* 13(1), pp. 3–19, doi:10.1007/s10009-010-0146-x.

[8] T. Han, J.-P. Katoen & A. Mereacre (2008): *Approximate Parameter Synthesis for Probabilistic Time-Bounded Reachability*. In: *Proceedings of the 29th IEEE Real-Time Systems Symposium, RTSS*, pp. 173–182, doi:10.1109/RTSS.2008.19.

[9] J.-P. Katoen, I.S. Zapreev, E.M. Hahn, H. Hermanns & D.N. Jansen (2011): *The Ins and Outs of the Probabilistic Model Checker MRMC*. *Perform. Eval.* 68(2), pp. 90–104, doi:10.1016/j.peva.2010.04.001.

[10] M. Z. Kwiatkowska, G. Norman & D. Parker (2011): *PRISM 4.0: Verification of Probabilistic Real-Time Systems*. In: *Computer Aided Verification - 23rd International Conference, CAV*, pp. 585–591, doi:10.1007/978-3-642-22110-1_47.

[11] K. Mehlhorn (1984): *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*. *EATCS Monographs on Theoretical Computer Science* 2, Springer. Available at `http://www.mpi-sb.mpg.de/~mehlhorn/DatAlgbooks.html`.

[12] S. Pathak, E. Ábrahám, N. Jansen, A. Tacchella & J.-P. Katoen (2015): *A Greedy Approach for the Efficient Repair of Stochastic Models*. In: *NASA Formal Methods - 7th Int. Symposium*, pp. 295–309, doi:10.1007/978-3-319-17524-9_21.

[13] B.-S.-K. Tati & M. Siegle (2015): *Parameter and Controller Synthesis for Markov Chains with Actions and State Labels*. In: *2nd International Workshop on Synthesis of Complex Parameters (SynCoP'15)*, *OASIcs* 44, pp. 63–76, doi:10.4230/OASIcs.SynCoP.2015.63.

Parametric, Probabilistic, Timed Resource Discovery System

Camille Coti
LIPN, CNRS UMR 7030, Université Paris 13, Sorbonne Paris Cité
camille.cot@lipn.univ-paris13.fr

This paper presents a fully distributed resource discovery and reservation system. Verification of such a system is important to ensure the execution of distributed applications on a set of resources in appropriate conditions. A semi-formal model for his system is presented using probabilistic timed automata. This model is timed, parametric and probabilistic, making it a challenge to the parameter synthesis community.

1 Introduction

The behavior of distributed systems in general can be challenging to prove. On the other hand, model checking techniques are particularly well adapted to verify that they follow a certain specification, because such techniques explore every possible execution of the system. Moreover, the execution of a distributed algorithm can depend on a large number of parameters, because of the complex combination of elements involved. It is therefore difficult to have a quantitative evaluation of what happens in the system for each and every value of each parameter.

This paper focuses on a resource management system. For some applications, resources can be shared between clients. These clients may need to access the resources in exclusive mode. As a consequence, a reservation system is necessary to arbitrate between clients and orchestrate the utilization of the resources.

For instance, a laboratory can buy a set of computation nodes and put them together in a cluster. A specific node, called the front-end, is used to access the computation nodes and a batch scheduler installed on this front-end node issues reservations on the nodes. The nodes cannot be accessed by a user that does not have an ongoing reservation on the said node issued by the batch scheduler. In practice, this reservation system is implemented using a single job queue that maintains a list of jobs that need to be executed and a list of available resources, and schedules the former on the latter [1, 9, 3, 2].

However, in many situations, this architecture cannot be implemented. For instance, a small lab who bought a handful of GPU nodes might not want to dedicate hardware and human time to setup and administrate a front-end node with a batch scheduling system. On some critical systems, this component can be seen as a single point of failure and then reduce the reliability of the whole system. However, in these situations, it is still necessary to make sure that applications have exclusive access to the machines they are using.

In this paper, we present a completely distributed reservation system, based on the state maintained by each machine itself, and the local network protocol Zeroconf [4]. This system was modeled using (timed) Petri Nets in [11, 10]. However, we have seen that this system is highly parametrizable. the purpose of this paper is to present it as a parametric system for verification and parameter synthesis. The resulting model has a high level of complexity, and therefore forms a case-study which cannot be reasonably solved with current methods. With this paper we aim at presenting it in order to open a discussion for methods that would tackle these problems, for instance by combining or hybriding methods.

The rest of this paper is organized as follow. The global architecture of the system and the algorithms are described in Section 2. Section 3 describes how it can be modeled. Section 4 specifies the expected behavior of the system and how parameter synthesis is useful to verify this behavior. Finally, section 5 concludes this paper and opens questions for the community.

2 Presentation of the system

This section presents the reservation system itself. The global architecture is described in section 2.1, the algorithms are presented in section 2.2 for the reservation protocol itself and in section 2.3 for the machines.

2.1 Architecture

The global architecture of the system, named QURD, is depicted in Figure 1. Computing resources *declare themselves* on the Zeroconf bus, and users/clients look on the Zeroconf bus to see which resources are available. An *application* (or a *job*) is made of several *processes* that are meant to run on a set of *resources*, also called *machines*. The user submits an application through a *client*.

The Zeroconf bus [4] is a network protocol used for self-configuration of network services. It was originally designed to allow automatic configuration of computers without any intervention from the user nor any centralized server, and later extended to various services. For instances, it is widely used for services such as DNS or network printers. In the latter example, printers declare themselves on the Zeroconf bus, and workstations look on the Zeroconf bus to find out which printers are available.

Zeroconf uses multicast UDP datagrams. It features three operations, two of them can be used for automatic service detection: *discover* and *advertise*. The third operation is called *resolution*: for instance, it is used by the multicast DNS protocol (mDNS) works as follows: 1) the client sends a multicast datagram to ask “what is the IP address that corresponds to this symbolic name”; 2) the hosts that has this symbolic name name answers with a unicast message to the client. A host that provides a service would typically *advertise* it. For this purpose, it sends a multicast datagram to tell all the machines of the networks “I provide this service and on that port”. The other mode that can be used for service detection, *discover*, works the other way around: 1) a client sends a multicast datagrapm to as “who provides this service?”; 2) servers that host the requested service reply to the client using a unicast datagram.

In our system, machines declare themselves on the Zeroconf bus when they are available and withdraw themselves when they are taken by a client, *i.e.* when a job is running on them: they use the *advertise* mode. Clients listen and receive the multicast declarations that are sent on the network.

However, this is not sufficient to ensure exclusive access to the machines, because of the asynchronous nature of the system and the absence of a consistent view between concurrent clients. Besides, the Zeroconf protocol does not have real-time accuracy: a machine which is not available anymore might be still visible on it. For instance, if a resource is available at a given moment, two clients will see it on the Zeroconf bus. Both will issue a request, but only one of them should get it.

2.2 Reservation algorithm

When a client wants to reserve a set of machines, the submission protocol works as follows. The client listens to the Zeroconf bus for available machines. It gets a list of available machines. It contacts them one by one and waits for a reply. If the machine acknowledges the reservation, the client receives an “OK” message and keeps the machine. Otherwise, the client receives a “KO” message.

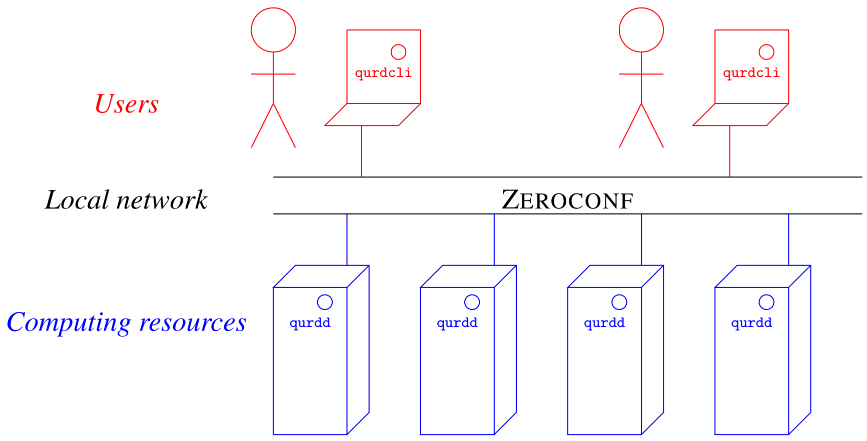


Figure 1: Architecture of a QURD system.

Once the client has enough machines, it starts its job on the said machines. Otherwise, two behaviors are possible. Either the client waits for other machines to become available, with a limit of time set by a timeout (called the *wait semantics*), or it cancels its reservation and frees the machine it has reserved, to try again later (called the *fail semantics*). The algorithm corresponding to the latter approach is given by algorithm 1.

Algorithm 1: Resource reservation algorithm (fail semantics)

```
reserveNodes( nbNodes) begin
    Data: machines = {}
    listenZeroconf();
    foreach machine m newly discovered do
        if card( machines ) < nbNodes then
            contactMachine( m ) ;
            ack = receiveAck( m );
            if ack == OK then
                machines.append( m ) ;
    if card( machines ) == nbNodes then
        return machines ;
    else
        freeMachines( machines ) ;
        return {} ;
```

One important issue with these two reservation semantics is to avoid deadlocks between concurrent reservation requests. For instance, if we consider a system with three machines and two concurrent reservation requests, one for two machines and one for three machines. If the first request gets one machine and the second gets two machines, neither of them has obtained enough machines and there is no spare machine left. This situation would lead to a deadlock without the possibility to release and retry (*fail* semantics) or release everything after a timeout (*wait* semantics). The *wait* semantics is still useful if there is no available resource because some of them are used by running applications. In this case, as soon as an application is done, the request can get its resources.

2.3 Exclusive access protocol

In order to make sure that jobs have exclusive access to the machines that have been assigned to them, each machine implements a protocol based on its state. When a machine is available for jobs, its state is set to *available*. It can be reserved only when it is in the stat *available*. Otherwise, in any other state, it answers all the requests with "KO".

2.4 End of a job

Once the process running on a machine is done, the machine's state is set to *finished*. A job ends when all the processes are done: the client keeps track of which machine is done. Once the job is finished, the machines switch back into state *available* and republish themselves on the Zeroconf bus.

2.5 Resource volatility

Resources can be subject to failures. If they fail while they are idle, they switch to state *unavailable* or, if they are still visible on the Zeroconf bus, they never answer the clients' requests and the clients try to find another resource. If they fail while they are in state *reserved*, they never confirm the start-up of the application to the client it has been reserved by and the client handles this case too. If it fails when it is in state *finished*, the client has already taken into account the fact that this resource is done executing its part of the job, so it has no consequence on the execution of the process. The resource goes to state *unavailable* instead of *available*.

The main issue is when the failure happens when the resource is running the job. The failure is detected by a failure detector [6] and the machine is considered to be *dead*. The client is notified of this failure and tries to find a new machine to replace the failed one.

3 Modeling the system

This section presents models for the two parts of the system: the machines (section 3.1) and the reservation system of the clients (section 3.2). These two components interact with each other. These interactions are detailed in section 3.3.

The models are presented using a finite-state formalism. Transitions between states are represented by edges. Guards can be present on edges, given between brackets. When the system can chose between two transitions depending on a probability, the probability to chose a given edge is given between parenthesis. Edges related with each other by a probability are linked by an arc. The actions taken by a transition is given on the corresponding edge. In particular, when a transition can be taken only after T units of time (*i.e.* the system must remain in a state during at least T units of time), the time is initialized before entering the state by an action on the incoming edge ($time := 0$), and a guard on the outgoing edge sets the condition on the time to take this transition ($time > T$).

Interactions between automata are modeled by synchronizations on actions, in a similar way as what is presented in [8]. For instance, when an automaton sends a request, the corresponding edge contains the `request!` action and the automaton receives it with `request?` on the guard of an edge.

3.1 Model of each machine

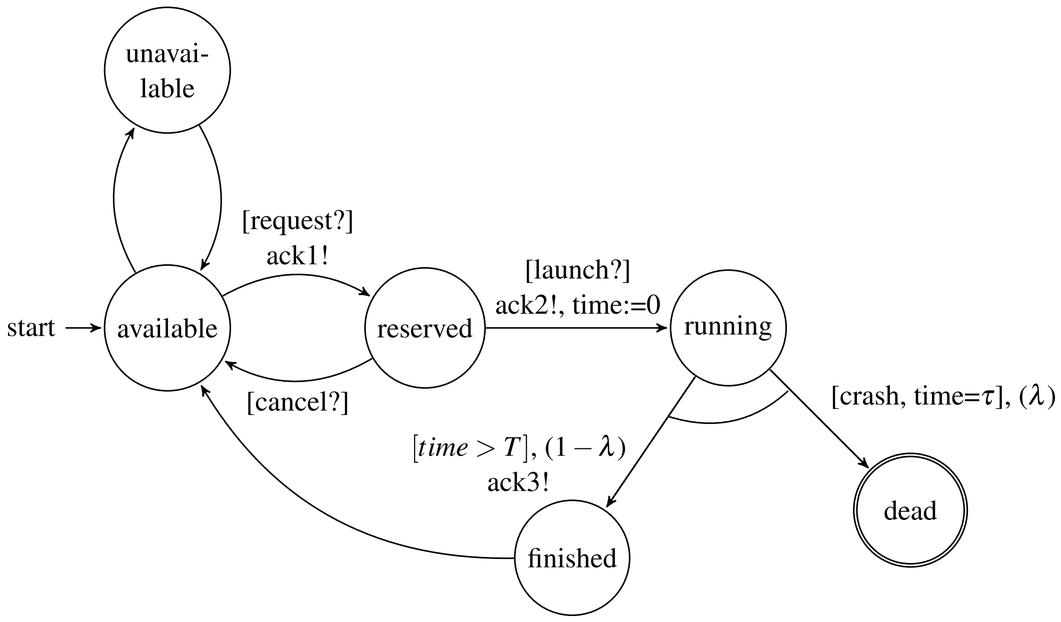


Figure 2: Automaton modeling the algorithm running on each machine.

The states of a machine are represented in the automaton depicted on figure 2. When no job is running on it, a machine is in the *available* state. For some (local) reason, such as an action from an administrator or a local user (when the machines are unused workstations, for instance with desktop grids), it can become *unavailable*. When the machine is in the state *available*, it can be reserved by a client: in this case, it enters the state *reserved*. We have seen in section 2.2 that a client can cancel a reservation; in this case, the machine returns to the state *available*.

Once the client has all the machines it needs to start a job, it sends a command to all of the machines it has reserved. These machines enter the state *running*, because it is running a process from a job. However, the machine can crash or fail during the execution. It happens with probability λ: then, the machine enters the state *dead*. The probability is given between parenthesis on the edges between states.

A more detailed model is given in figure 3. The resource has a probability λ of dying. If it dies, it reaches the state *fragile*. If it does not, if reaches the state *sustain*. It stays in the sustain state for (at least) T time units, representing the execution time of the process. Figure 2 is a more compact representation of this subpart of the model, since the transition after the state *running* is conditioned by a probability and a time.

Once the execution is done (*i.e.* after a given period of time and with probability $1-\lambda$), it enters the state *finished* and then can be *available* again.

When the machine is reserved, the machine notifies the client it acknowledges the request. Similarly, when the machine starts the application, it acknowledges the client. At the end of the execution, it notifies the client it is done with its local process.

We can seen that a machine can evolve between states, except if a failure happens and it dies. In this case, it stays in the state *dead* until an administrator performs an action to fix the problem and restart the machine, which can take a long time compared to the typical execution time of each transition of the automata presented here. As a consequence, we are considering here that the machine stays in the state *dead*.

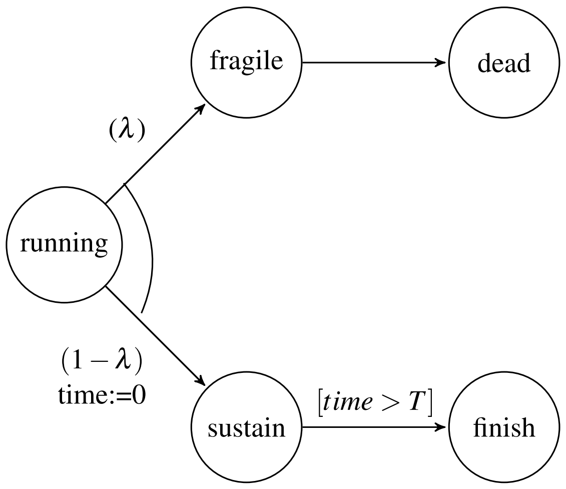


Figure 3: Modeling the volatility of a resource.

3.2 Model of the reservation system

The reservation system is modeled by the automaton given on figure 4. It uses counters. Initially, the system is in state *begin*. A counter is used to count the number of machines that have acknowledged the reservation; it is initialized to zero when the system transitions from the state *begin* to the state *reserve*. Each time a machine acknowledges the reservation, the counter is incremented. When the required number of machines have acknowledged (*NB OK* answers), the automaton reaches the state *launch*. In a similar way, it counts the number of machines that have acknowledged application start-up. Once they all have answered, the automaton reaches the state *wait*, until all the machines are done.

The automaton presented here implements the *wait* semantics. If not enough machines can be reserved, the system releases the machines and returns to the initial state.

A detailed model for the state *reserve* is given in Figure 5. This model is made of two parts. The top part of the model is the *discovery* system of Zeroconf: the client listens to the Zeroconf bus, and sends a request when it discovers a new machine. In the same time, the client waits for acknowledgements from the machines: this corresponds the lower part of the model. Every time an acknowledgement is received, a counted is incremented. When enough machines have answered, both automata reach the final state by synchronizing on the `done` action.

We have seen in section 2.5 that resources can fail during the execution of a job. In this case, the reservation system is informed by the failure detector and reaches the state *failure*. Then it requests an additional resource and starts the application on it.

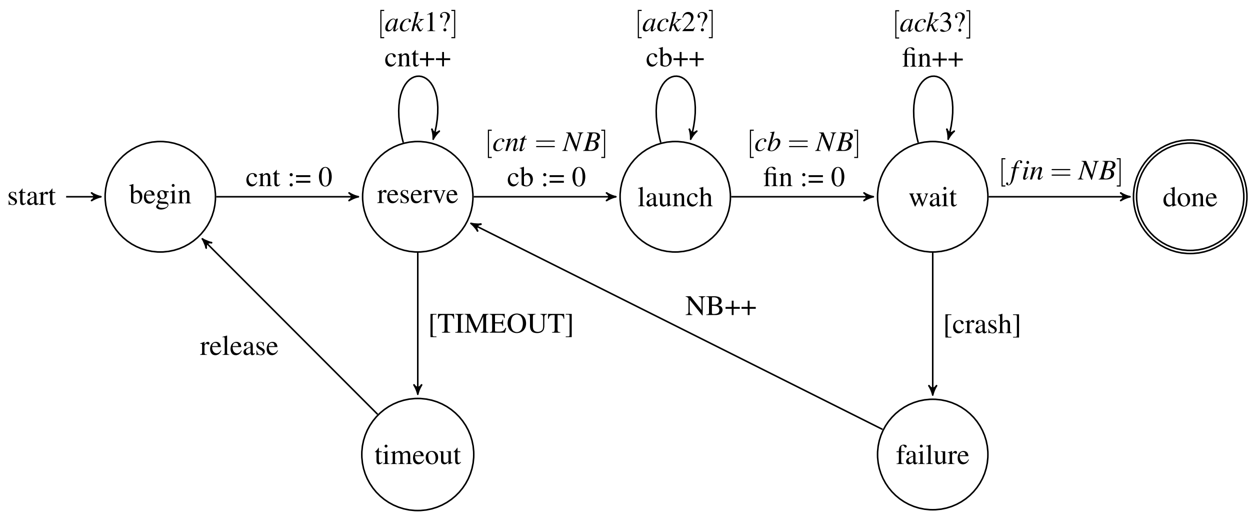


Figure 4: Automaton modeling the reservation system (*wait* semantics)

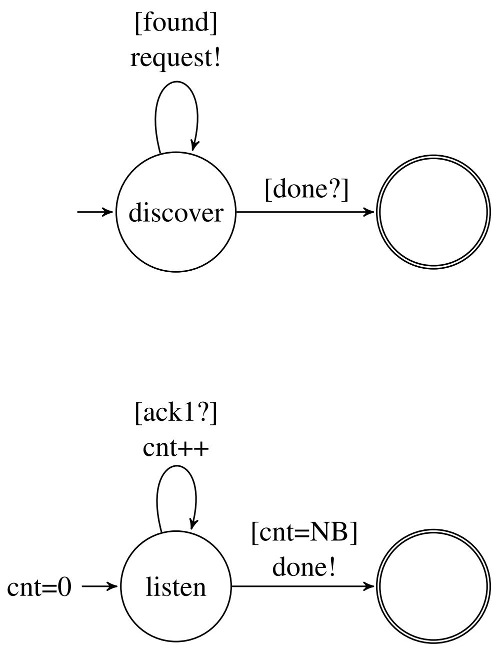


Figure 5: Detailed model for the state *reserve* (wait semantics)

3.3 Interactions between the automata

We have seen in Figure 2 that the machines need some actions from the reservation systems and that some actions are made to this reservation system. In a similar way, we have seen in figure 4 that the reservation system interacts with the machines. This set of actions forms a mini-protocol between the two automata.

1. The reservation system *sends a request* to the machines it has found on the Zeroconf bus (`request` action). When a machine receives a request, it answers *OK* or *KO* depending on the state it is currently in.
2. The available machines *acknowledge* the request (`ack1` action). For each acknowledgement it receives, the client increments a counter.
3. The reservation system *sends a command* to the machines that were assigned to it (`launch` action). When each machine receives it, it starts the command.
4. The available machines *acknowledge* the command (`ack2` action). The client counts the number of acknowledgement it receives.
5. The available machines *notify* the reservation system that the execution of its local process is done (`ack3` action). The client counts the number of acknowledgement it receives and, when it has received all of them, the execution is done.

4 Parameter synthesis and verification of the system

This system contains many parameters, and its behavior depends on these parameters. Parameter synthesis can be useful to verify is behavior under different parameters. An exploration of the behavior of such as system, for instance using behavioral cartography [7], can give the ranges of parameters for the system to behave as expected.

4.1 Expected behavior

The system must have several properties. The *soundness* of the system [5] (option to complete, proper completion and no dead transitions) was verified in [10, 11]: in the absence of failures, all the jobs are executed and complete. If resources can fail, there can be too many failures, in which case the jobs that need more resources than there are surviving resources cannot be executed. In this case, it was verified that *there exists* an execution in which all the jobs complete.

More specifically, it is also necessary to ensure *exclusive access* of the processes of all the jobs on the resources. It is one of the expected properties of this system: when a process of a job is executed on a resource, this resource must not be attributed to any other job. This property is important for instance for computation resources (computation nodes, GPUs...): if a node runs more processes than the number of cores it has, the processes need to share the execution time. This situation is called *oversubscription*. This property was also verified using Petri nets in [10, 11].

Timed models such as timed Petri nets [11] give a more precise idea of the behavior of the system in terms of, for example, deadlines. Verification techniques on such models can provide properties such as "in T time units, all the jobs have completed".

Parameter synthesis is particularly important to have a more precise idea of these properties. For instance, it can give precise completion time for a range of execution times. It is highly important for critical systems for example, to verify what is doable and under which conditions. In particular, one can want to make sure that for every possible execution, all the jobs complete before a certain number of time units. Parameter synthesis would help dimensioning the system (number of resources, number of jobs to put in the system) to be sure that the system behaves as required.

4.2 Parameters of the model

We have seen in sections 3.1 and 3.2 that the model depends on several parameters:

- The number of machines in the system;
- The number of clients issuing requests in the system;
- The number of resources asked by each client;
- The execution time of the processes of each job;
- The failure probability of each resource;
- In the *wait* semantics, the value of the timeout;
- In the *fail* semantics, the time after which the request is issued again.

The number of resources asked by each client is specific for each client. For instance, a client may ask for three machines and another one may ask for six machines. The execution time is assumed to be roughly the same for all the processes of a given job but not necessarily the same. As a consequence, machines spend some time in the state *finished*, but this time is generally small compared to the time spent in the state *running*.

However, if a resource has crashed during the execution of a process, the failed process must be re-executed, possibly from the beginning (some applications include a failure-recovery protocol that may reduce the re-execution time). Therefore, at the end of the execution of this job, the other resources used by the job are waiting for it in the state *finished*.

Therefore, the failure rate is very important to compute a likelihood of execution time. For instance, one can expect a result like "There is a likelihood of 50% that all the applications will be done after N time units, 25% after $2N$ time units, 15% after $3N$ time units and 10% that machines will crash too often for the applications to complete".

Besides, keeping some parameters unknown would allow to dimension the system with respect to some requirements. For instance, for a given number of resources, how many jobs can be executed and finish on time? Or the other way around, for a given number of jobs, how many resources does the system need to have to make sure that all the jobs will be executed and finish on time?

5 Conclusion

In this paper, we have presented a distributed algorithm for resource discovery and reservation. This algorithm was verified using model checking techniques (P/T Petri nets, Colored Petri nets and Timed Petri nets) in a previous paper, but these tools did not take into account the fact that the system contains several parameters, including on times and probabilities.

Parameter synthesis techniques such as behavioral cartography would permit to exhibit values for these parameters that would guarantee that the system follows a certain behavior. However, the complexity of the model, which is in the same time parametric, timed and probabilistic, makes it challenging for current parameter synthesis techniques.

Therefore, we hope that the system presented and modeled in this paper will be a challenge for the parameter synthesis community to find fitting parameter synthesis, maybe by hybridizing or combining several existing techniques, or developing new, specific ones.

One approach would be to consider parts of the problem as smaller instances and to decompose it as models of increasing complexity. A finite-state model, with no probability, can be used to perform a worst-case analysis. A real-time model, with parameterized delays, timeouts and number of processes and resources but no probability, can be used to do a verification and quantification of the behavior of the system without failures. Last, the full model allows to perform verification and behavior analysis of the complete system.

Moreover, some assumptions can be made to perform a worst-case analysis and verify some properties. For instance, the failure probability can be simplified by a global failure rate, such as “at most 10% of the resources fail during the execution of a job”. Under this assumption, the model loses its probabilistic nature.

References

[1] *LSF: Load Sharing Facility*. WWW Page: http://www.platform.com/Products/platform-lsf. Available at `http://www.platform.com/Products/platform-lsf`.

[2] *Portable Batch System*. WWW Page: http://www.pbsgridworks.com/. Available at `http://www.pbsgridworks.com/`.

[3] *Torque Resource Manager*. WWW Page: http://www.clusterresources.com/products/torque-resource-manager.php. Available at `http://www.clusterresources.com/products/torque-resource-manager.php`.

[4] *Zero Configuration Networking (Zeroconf)*. WWW Page: http://www.zeroconf.org. Available at `http://www.zeroconf.org`.

[5] Wil M. P. van der Aalst (1997): *Verification of Workflow Nets*. In: *ICATPN*, LNCS, Springer-Verlag, London, UK, pp. 407–426.

[6] Marcos Kawazoe Aguilera, Wei Chen & Sam Toueg (1997): *Heartbeat: A Timeout-Free Failure Detector for Quiescent Reliable Communication*. In Marios Mavronicolas & Philippas Tsigas, editors: *Proceedings of the 11th Workshop on Distributed Algorithms (WDAG'97)*, *Lecture Notes in Computer Science* 1320, Springer, pp. 126–140.

[7] Étienne André & Laurent Fribourg (2010): *Behavioral Cartography of Timed Automata*. In Antonín Kučera & Igor Potapov, editors: *Proceedings of the 4th Workshop on Reachability Problems in Computational Models (RP'10)*, *Lecture Notes in Computer Science* 6227, Springer, Brno, Czech Republic, pp. 76–90, doi:10.1007/978-3-642-15349-5. Available at `http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/AF-rp10.pdf`.

[8] Gerd Behrmann, Alexandre David & Kim G Larsen: *A tutorial on UPPAAL 4.0 (Updated November 28, 2006)*.

[9] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cyrille Martin, Grégory Mounié, Pierre Neyron & Olivier Richard (2005): *A batch scheduler with high level components*. In: *Proceedings of the 5th International Symposium on Cluster Computing and the Grid (CCGRID'05)*, IEEE Computer Society, Cardiff, UK, pp. 776–783.

[10] Camille Coti, Sami Evangelista & Kais Klai (2015): *Queue-less, Uncentralized Resource Discovery: Formal Specification and Verification*. In Daniel Moldt, Heiko Rölke & Harald Störrle, editors: *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'15), including the International Workshop on Petri Nets for Adaptive Discrete Event Control Systems (ADECS 2015) A satellite event of the conferences: 36th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2015 and 15th International Conference on Application of Concurrency to System Design ACSD 2015, Brussels, Belgium, June 22-23, 2015.*, *CEUR Workshop Proceedings* 1372, CEUR-WS.org, pp. 315–316. Available at `http://ceur-ws.org/Vol-1372/paper19.pdf`.

[11] Camille Coti, Sami Evangelista & Kais Klai (2015): *Time Petri Net Models for a New Queueless and Uncentralized Resource Discovery System*. *CoRR* abs/1502.03431. Available at `http://arxiv.org/abs/1502.03431`.

Weighted Branching Simulation Distance for Parametric Weighted Kripke Structures

Louise Foshammer, Kim Guldstrand Larsen and Anders Mariegaard
Department of Computer Science, Aalborg University
Selam Lagerlöfs Vej 300, DK-9220 Aalborg, Denmark.
{foshammer,kgl,am}@cs.aau.dk

This paper concerns branching simulation for weighted Kripke structures with parametric weights. Concretely, we consider a weighted extension of branching simulation where a single transition can be matched by a sequence of transition while preserving the branching behavior. We relax this notion to allow for a small degree of deviation in the matching of weights, inducing a directed distance on states. The distance between two states can be used directly to relate properties of the states within a sub-fragment of weighted CTL. The problem of relating systems thus changes to minimizing the distance which, in the general parametric case, corresponds to finding suitable parameter valuations such that one system can approximately simulate another. Although the distance considers a potentially infinite set of transition sequences we demonstrate that there exists an upper bound on the length of relevant sequences, thereby establishing the computability of the distance.

1 Introduction

In recent years within the area of embedded and distributed systems, a significant effort has been made to develop various formalisms for modeling and specification that address non-functional properties. Examples include extensions of classical Timed Automata [2] with cost and resource consumption/production in Priced Timed Automata [6] and Energy Automata [8]. For quantitative analysis of these systems, a generalization of bisimulation equivalence by Milner [17] and Park [19] as behavioral distances [21, 16, 1] between system, has been studied.

In parallel, *parametric* extensions of various formalism have been intensively studied. Instead of requiring exact specification of e.g probabilities, cost or timing constraints, these formalisms allow for the use of *parameters* representing unknown or unspecified values. This can be used to encode multiple configurations of the same system as a system being parametric in the configurable quantities. The problem is then to find "good" parameter values such that the instantiated system (configuration) performs as expected. For real-time systems, Parametric Timed Automata [3, 4] and Parametric Stateful Timed CSP [5] have been developed. Parametric probabilistic models [14, 13] have also been developed as well as parametric analysis for weighted Kripke structures [9, 10, 15]. [10] provides an efficient model-checking algorithm for a parametric extension of real-time CTL on timed Kripke structures. [15] extends [10] to full parameter synthesis by demonstrating that model-checking a finite subset of the entire set of parameter values is sufficient.

In this paper we revisit (parametric) weighted Kripke structures with the purpose of lifting the behavioral distance defined in [11] to the parametric setting, demonstrate its fixed point characterization and prove computability of the distance between any two systems. The distance is a generalization of a weighted extension of branching simulation [12]. Consider the following two processes s,t both ending in the inactive process 0:

$$s \rightarrow_5 0 \text{ and } t \rightarrow_3 t_1 \rightarrow_2 0$$

Submitted to:
SynCoP 2016,
3rd International Workshop on Synthesis of Complex Parameters

If s, t, t_1 satisfy the same atomic proposition, t_1 may be deemed unobservable and t may simulate s as they both evolve into the process 0 with the same overall weight. [11] captures this situation in generality by extending branching simulation with weights. Consider a similar scenario, where the process t is now parametrized by the parameter p:

$$s \rightarrow_5 0 \text{ and } t \rightarrow_p t_1 \rightarrow_2 0$$

If $p \neq 3$ we know that t can no longer simulate s. However, it should be intuitive that $p = 6$ is somehow worse than $p = 2$ as the latter is closer to 3. Thus, instead of considering pre-orders and Boolean answers we develop a parametric distance between states such that as the value of p approaches 3, the distance between s and t decreases towards 0. The distance will also give us a direct relation between the properties satisfied by s and t and a distance of 0 implies that any formula satisfied by s is satisfied by t. In this way one can reason about how "close" a given implementation is to the specification and compare different configurations that are not necessarily able to fully simulate s.

The structure of this paper is as follows: in Section 2 we introduce preliminaries and recall results from [11], Section 3 concerns the fixed point characterization of the distance for weighted systems, Section 4 lifts the distance to the parametric setting and finally Section 5 concludes the paper and describes future work.

2 Preliminaries

A weighted Kripke Structure (WKS) extends the classical Kripke structure by associating to each transition a non-negative rational transition weight.

Definition 1 (Weighted Kripke Structure)**.** A weighted Kripke Structure is a tuple $\mathcal{K} = (S, AP, \mathcal{L}, \rightarrow)$ where S is a finite set of states, AP is a set of atomic propositions, $\mathcal{L} : S \rightarrow \mathcal{P}(AP)$ is a labelling function, associating to each state a set of atomic propositions and $\rightarrow \subseteq S \times \mathbb{Q}_{\geq 0} \times S$ is the finite transition relation.

A transition from s to s' with weight w will be denoted by $s \rightarrow_w s'$ instead of $(s, w, s') \in \rightarrow$.

Example 1. *Figure 1 depicts the WKS* $\mathcal{K} = (S, AP, \mathcal{L}, \rightarrow)$ *where* $S = \{s, s_1, s_2, s_3, s_4, t, t_1, t_2\}$, $AP = \{a, b\}$, $\mathcal{L}(s) = \mathcal{L}(s_1) = \mathcal{L}(s_2) = \mathcal{L}(t) = \mathcal{L}(t_2) = \{a\}$, $\mathcal{L}(s_3) = \mathcal{L}(s_4) = \mathcal{L}(t_1) = \{b\}$ *and* $\rightarrow = \{(s, 1, s_1), (s, 2, s_2), (s_1, 2, s_2), (s_1, 1, s_3), (s_1, 3, s_4), (s_2, 5, s_4), (t, 2, t_1), (t, 1, t_2), (t_2, 2, t_2), (t_2, 1, t_1)\}$.

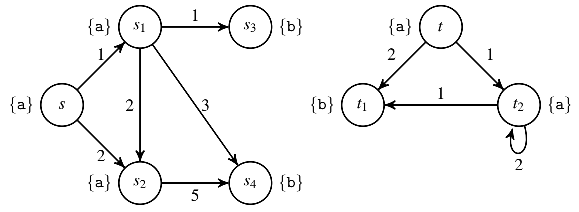


Figure 1: WKS $\mathcal{K}$ where $s \not\leq t$ and $t \not\leq s$ but $s \leq_{0.5} t$.

To reason about behavior of WKSs, we introduce a weighted variant of the classical notion of branching simulation [12]. The basic idea is to let a transition $s \rightarrow_5 s'$ be matched by a sequence of transitions $t \rightarrow_2 t_1 \rightarrow_2 t_2 \rightarrow_1 t_3$, if t_3 can simulate s', as the accumulated weight equals 5. In addition, each intermediate state passed through in the matching transition sequence must be able to simulate s. In this way the branching structure of systems is preserved. Instead of always requiring exact weight matching we allow small relative deviations. These small deviations will in Section 3 induce a directed distance between WKS states.

Definition 2 (Weighted Branching ε-Simulation [11]). Given a WKS $\mathscr{K} = (S, AP, \mathscr{L}, \rightarrow)$ and an $\varepsilon \in \mathbb{R}_{\geq 0}$, a binary relation $R^\varepsilon \subseteq S \times S$ is a weighted branching ε-simulation relation if whenever $(s,t) \in R^\varepsilon$:

- $\mathscr{L}(s) = \mathscr{L}(t)$
- for all $s \rightarrow_w s'$ there exists $t \rightarrow_{v_1} t_1 \rightarrow_{v_2} \cdots \rightarrow_{v_k} t_k$ such that $\sum_{i=1}^{k} v_i \in [w(1-\varepsilon), w(1+\varepsilon)], (s', t_k) \in R^\varepsilon$ and $\forall i < k.(s, t_i) \in R^\varepsilon$.

If there exists a weighted branching ε-simulation relating s to t we write $s \leq_\varepsilon t$. If $\varepsilon = 0$ we write $s \leq t$ instead of $s \leq_0 t$. Note that in this case $\sum_{i=1}^{k} v_i = w$.

Example 2. *Consider again Figure 1 and the pair (s,t). It is clear that $t \not\leq s$ because of the loop $t_2 \rightarrow_2 t_2$. We can also observe that $s \not\leq t$ as the transition $s \rightarrow_2 s_2$ can only be matched by $t \rightarrow_2 t_1$ but $s_2 \not\leq t_1$ as $\mathscr{L}(s_2) \neq \mathscr{L}(t_1)$. If we relax the matching requirements by 50%, we get that s can be simulated by t i.e $s \leq_{0.5} t$; $s \rightarrow_2 s_2$ can be matched by $t \rightarrow_1 t_2$ as $[2(1-0.5), 2(1+0.5)] = [1,3]$ and $1 \in [1,3]$ (another legal match would be $t \rightarrow_1 t_2 \rightarrow_2 t_2$). Now, $s_2 \rightarrow_5 s_4$ can be matched exactly by $t_2 \rightarrow_2 t_2 \rightarrow_2 t_2 \rightarrow_1 t_1$. It follows that $\varepsilon \geq 0.5 \iff s \leq_\varepsilon t$.*

If we restrict weighted CTL to only encompass the existential quantifier and remove the next-operator and we know that $s \leq_\varepsilon t$, then for any property ϕ of s, there exists a related property ϕ^ε of t.

Definition 3 (Existential Fragment of Weighted CTL without next). The syntax of $EWCTL_{-X}$ is given by the following abstract syntax:

$$\phi ::= a \mid \neg a \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid E(\phi_1 U_I \phi_2),$$

where $a \in AP$, $I = [l,u]$ and $l,u \in \mathbb{Q}_{\geq 0}$ such that $l \leq u$. For a WKS $\mathscr{K} = (S, AP, \mathscr{L}, \rightarrow)$ and an arbitrary state $s \in S$, the semantics of $EWCTCL_{-X}$ formulae is given by a satisfiability relation, inductively defined on the structure of formulae in $EWCTL_{-X}$. For existential until; $\mathscr{K}, s \models E(\phi_1 U_I \phi_2) \iff$ there exists a sequence $s \rightarrow_{w_1} s_1 \rightarrow_{w_2} \cdots \rightarrow_{w_k} s_k \rightarrow_{w_{k+1}} \ldots$ where $s_k \models \phi_2, \forall i < k. s_i \models \phi_1$ and $\sum_{i=1}^{k} w_i \in I$. Let the *ε-expansion* of a formula $\phi = E(\phi_1 U_{[l,u]} \phi_2)$ be given by $\phi^\varepsilon = E(\phi_1^\varepsilon U_{[l(1-\varepsilon), u(1+\varepsilon)]} \phi_2^\varepsilon)$ where ϕ_1^ε and ϕ_2^ε are defined inductively by relaxing any interval by ε percent in both directions (just as for $[l,u]$).

Theorem 1. *[11] Let $\mathscr{K} = (S, AP, \mathscr{L}, \rightarrow)$ be a WKS. Then for all $s,t \in S, \varepsilon \in \mathbb{R}_{\geq 0}$:*

$$s \leq_\varepsilon t \quad \textit{iff} \quad \forall \varepsilon' \in \mathbb{Q}_{\geq 0}, \varepsilon \leq \varepsilon'.[\forall \phi \in EWCTL_{-X}. s \models \phi \implies t \models \phi^{\varepsilon'}].$$

3 Weighted Branching Simulation Distance for WKSs

We now define a directed distance between WKS states as a least fixed point to a set of equations. The distance from s to t, $d(s,t)$, represents the minimal ε such that $s \leq_\varepsilon t$. Thus, if $d(s,t) = 0$ then $s \leq t$. As the distance is based upon weighted branching ε-similarity and its relative deviation in weight matching, it will not satisfy the triangle inequality and is therefore not a hemi-metric [1].

The distance definition follows intuitively weighted branching ε-simulation. If $s \leq_\varepsilon t$ then no matter what transition s chooses, t has a matching transition sequence with a relative difference of at most ε. In order words, for a given transition $s \rightarrow_w s'$, the goal of t is to find a matching sequence $t \rightarrow_{v_1} t_1 \cdots \rightarrow_{v_n} t_n$ that *minimizes* the relative difference $\left|\frac{\sum_{i=0}^{n} v_i}{w} - 1\right|$ as well as ensuring that any intermediate state t_i has as small a distance to s as possible. The strategy of s is then to find a *maximal* move, given the minimization strategy of t. In the remainder of this section we assume a fixed WKS $\mathscr{K} = (S, AP, \mathscr{L}, \rightarrow)$.

[1]absolute deviations would lead to a hemi-metric but is insufficient when the weights have different magnitudes.

Definition 4 (Weighted Branching Simulation Distance)**.** For an arbitrary pair of states $s, t \in S$ we define the weighted branching simulation distance from s to t, $d(s,t)$, as the least fixed point ($\overset{\min}{=}$) of the following set of equations:

$$d(s,t) \overset{\min}{=} \begin{cases} \infty & \text{if } \mathscr{L}(s) \neq \mathscr{L}(t) \\ \max_{s \to_w s'} \left\{ \min_{t \to_{v_1} t_1 \cdots \to_{v_n} t_n} \left\{ \max \left\{ \begin{array}{l} \left| \frac{\sum_{i=0}^{n} v_i}{w} - 1 \right|, d(s', t_n), \\ \max\{d(s,t_i) \,|\, i < n\} \end{array} \right\} \right\} \right\} & \text{o.w} \end{cases}$$

We assume the empty transition sequence to have accumulated weight 0 and let $\mathbb{R}_{\geq 0} = \{w \,|\, w \in \mathbb{R}, w \geq 0\} \cup \{\infty\}$ denote the extended set of non-negative reals. For any $d_1, d_2 \in \mathbb{R}_{\geq 0}^{S \times S}$ let $d_1 \leq d_2$ iff $\forall (s,t) \in S \times S. d_1(s,t) \leq d_2(s,t)$. Then $(\mathbb{R}_{\geq 0}^{S \times S}, \leq)$ constitutes a complete lattice. We now define a monotone function on $(\mathbb{R}_{\geq 0}^{S \times S}, \leq)$ that iteratively refines the distance:

Definition 5. Let $\mathscr{F} : \mathbb{R}_{\geq 0}^{S \times S} \to \mathbb{R}_{\geq 0}^{S \times S}$ be defined for any $d \in \mathbb{R}_{\geq 0}^{S \times S}$:

$$\mathscr{F}(d)(s,t) = \begin{cases} \infty & \text{if } \mathscr{L}(s) \neq \mathscr{L}(t) \\ \max_{s \to_w s'} \left\{ \min_{t \to_{v_1} t_1 \cdots \to_{v_n} t_n} \left\{ \max \left\{ \begin{array}{l} \left| \frac{\sum_{i=0}^{n} v_i}{w} - 1 \right|, d(s', t_n), \\ \max\{d(s,t_i) \,|\, i < n\} \end{array} \right\} \right\} \right\} & \text{o.w} \end{cases}$$

By Tarski's fixed point theorem [20] we are guaranteed the existence of a least (pre-)fixed point. Thus, the weighted branching simulation distance is well-defined. Note that any transition $s \to_w s'$, t may have an infinite set of possible transition sequence matches in the presence of cycles in the system. To this end we demonstrate an upper bound, N, on the length of relevant matching sequences. As the set of sequences of length at most N is finite (the WKS is finite) computability of the distance follows. The first step is proving that any sequence exercising a loop with accumulated weight 0 can be ignored. We refer to these cycles as *0-cycles*.

Lemma 1. *For a given move $s \to_w s'$, any transition sequence $t \to_{v_1} t_1 \cdots \to_{v_n} t_n$ with a 0-cycle can be removed without affecting the distance $d(s,t)$.*

Proof. A transition sequence with one or more 0-cycles has the exact same accumulating weight as the corresponding sequence with no 0-cycles. Furthermore, exercising the loop (once) can only introduce new states, leading to a potentially larger value of $\max\{d(s,t_i) \,|\, i < n\}$. Thus, 0-cycles can be ignored. □

Given that 0-cycles can be removed, we now prove an upper bound N on the length of sequences that affect the distance $d(s,t)$. Thus, any sequence longer than N can be safely ignored.

Lemma 2. *Given that $\mathscr{K}$ has no 0-cycles, it is the case that whenever $s \to_w s'$:*

$$\begin{array}{l} \exists N. \forall \pi = t \to_{v_1} t_1 \ldots \to_{v_n} t_n, n \geq N. \\ \quad \exists \pi^* = t \to_{u_1} t'_1 \ldots \to_{u_m} t'_m, m \leq N. \\ \quad t_n = t'_m \wedge \left| \dfrac{\sum_{i=0}^{m} u_i}{w} - 1 \right| \leq \left| \dfrac{\sum_{i=0}^{n} v_i}{w} - 1 \right| \wedge \\ \quad \{t'_1, \ldots, t'_{m-1}\} \subseteq \{t_1, \ldots, t_{n-1}\} \end{array}$$

Proof. Let $w_{\min} = \min\{w \,|\, s \to_w s'\}$ be the minimum weight in the WKS and let $s_{w_{\max}} = \max\{w \,|\, s \to_w s'\}$ be the maximum weight out of s. We now demonstrate that $N \geq \frac{2 \cdot s_{w_{\max}}}{w_{\min}} \cdot |S|$ is sufficient. Any sequence of length $|S|$ must have a loop which, by assumption, cannot have accumulated weight 0. Thus, after $|S|$

transitions, the accumulated weight must be at least $w_{\min}$. Without loss of generality, assume that it is *exactly* $w_{\min}$. If the sequence exercises the loop a number of time, the accumulated weight will at some point reach $2 \cdot s_{w_{\max}}$. Let this sequence be $\pi = t \rightarrow_{v_1} t_1 \cdots \rightarrow_{v_k} t_k$ and let x denote the number of times the loop is exercised i.e $x \cdot w_{\min} \geq 2 \cdot s_{w_{\max}}$. Consider now the corresponding sequence $\pi^* = t \rightarrow_{u_1} t'_1 \cdots \rightarrow_{u_l} t'_l$ where the loop is removed. As $\sum_{i=0}^{k} v_i \geq 2 \cdot s_{w_{\max}}$ it follows that $\left|\frac{\sum_{i=0}^{k} v_i}{s_{w_{\max}}} - 1\right| > 1$. By assumption, removing the loop results in a strictly lower accumulated weight implying $\left|\frac{\sum_{i=0}^{l} u_i}{s_{w_{\max}}} - 1\right| < \left|\frac{\sum_{i=0}^{k} v_i}{s_{w_{\max}}} - 1\right|$. We also directly have $t_k = t'_l$ and $\{t_1, \ldots, t'_l\} \subseteq \{t_1, \ldots, t_k\}$. We will now derive N from the inequality $x \cdot w_{\min} \geq 2 \cdot s_{w_{\max}}$. The number of times the loops is exercised must be equal to the length of the entire sequence divided by $|S|$ as we are sure to exercise the loop every $|S|$ states. Thus, $x = \frac{N}{|S|} \implies \frac{N}{|S|} \cdot w_{\min} \geq 2 \cdot s_{w_{\max}}$ and finally,

$$N \geq \frac{2 \cdot s_{w_{\max}}}{w_{\min}} \cdot |S|.$$

□

Theorem 2 (Computability). *For two states $s, t \in S$, the weighted branching simulation distance is computable.*

Proof. Lemma 2 provides an upper bound on the length of transition sequence that we need to consider in the computation of $d(s,t)$ for any states $s, t \in S$ under the assumption that there are no 0-cycles. By Lemma 1 we know that any 0-cycles can be removed without affecting the distance. Thus when computing the distance we know for the sub-expression

$$\min_{t \rightarrow_{v_1} t_1 \cdots \rightarrow_{v_n} t_n} \left\{ \max \left\{ \begin{array}{l} \left|\frac{\sum_{i=0}^{n} v_i}{w} - 1\right|, d(s', t_n), \\ \max\{d(s, t_i) \mid i < n\} \end{array} \right\} \right\}$$

that $n \leq \frac{2 \cdot s_{w_{\max}}}{w_{\min}} \cdot |S|$. As the WKS has a finite number of states and a finite transition relation, only a finite number of sequences of finite length exist. Thus we can modify the distance function to only consider these without affecting the computed distance. Thus, the distance must at some point converge as only a finite number of relative distances on the form $\left|\frac{\sum_{i=0}^{n} v_i}{w} - 1\right|$ exists. □

We leave the exact complexity of computing $d(s,t)$ open but note that deciding $d(s,t) = 0$ is NP-complete [11].

Example 3. *Consider again Figure 1 and the computation of $d(s,t)$. For the transition $s \rightarrow_1 s_1$ only one sequence is considered instead of the entire infinite set arising from the loop; $t \rightarrow_1 t_2$. As $\left|\frac{3}{1} - 1\right| > \left|\frac{1}{1} - 1\right|$, even the sequence that only exercises the loop once is worse than just transitioning to t_2 directly. This happens because the accumulated matching weight exceeds the weight being matched and the same states are involved in both sequences. Therefore any sequence involving the loop can be ignored. Note that we in this example consider fewer sequences than implied by the upper bound given in Lemma 2. For $s \rightarrow_1 s_1$ the bound would be $\frac{2 \cdot 2}{2} \cdot 8 = 16$ but it should be clear that the loop can be safely ignored. For the transition $s \rightarrow_2 s_2$, there are two relevant matching sequences; $t \rightarrow_1 t_2$ and $t \rightarrow_1 t_2 \rightarrow_2 t_2$. Thus,*

$$d(s,t) \overset{\min}{=} \max \left\{ \begin{array}{l} \max\left\{\left|\frac{1}{1} - 1\right|, d(s_1, t_2)\right\}, \\ \min \left\{ \begin{array}{l} \max\left\{\left|\frac{1}{2} - 1\right|, d(s_2, t_2)\right\}, \\ \max\left\{\left|\frac{3}{2} - 1\right|, d(s_2, t_2), d(s, t_2)\right\} \end{array} \right\} \end{array} \right\}$$

It is easily shown that $d(s_2,t_2)=0$ as $s_2 \rightarrow_5 s_4$ can be matched exactly by $t_2 \rightarrow_2 t_2 \rightarrow_2 t_2 \rightarrow_1 t_1$. Thus,

$$d(s,t) \overset{\min}{=} \max\left\{\frac{1}{2}, d(s_1,t_2), d(s,t_2)\right\}$$

where

$$d(s_1,t_2) \overset{\min}{=} \max\left\{\begin{matrix}\max\left\{\left|\frac{2}{2}-1\right|, d(s_2,t_2)\right\}, \\ \max\left\{\left|\frac{1}{1}-1\right|, d(s_3,t_1)\right\}, \\ \max\left\{\left|\frac{3}{3}-1\right|, d(s_1,t_2), d(s_4,t_1)\right\}\end{matrix}\right\} \quad \text{and} \quad d(s,t_2) \overset{\min}{=} \max\left\{\begin{matrix}\max\left\{\left|\frac{2}{1}-1\right|, d(s_2,t_2)\right\}, \\ \max\left\{\left|\frac{2}{2}-1\right|, d(s_2,t_2)\right\}\end{matrix}\right\}.$$

As $s_4 \not\rightarrow$, $s_3 \not\rightarrow$ and $t_1 \not\rightarrow$ it follows that $d(s_4,t_1)=d(s_3,t_1)=0$, hence

$$d(s_1,t_2) \overset{\min}{=} \max\left\{\frac{1}{2}, d(s_1,t_2)\right\}.$$

The least solution to this equation is $\frac{1}{2}$ hence $d(s_1,t_2)=d(s,t)=\frac{1}{2}$. From Example 2 we know that $s \leq_\varepsilon t$ for any $\varepsilon \geq 0.5$ i.e for any $\varepsilon \geq d(s,t)$.

Now that we have established the computability of the distance we prove its relation to weighted branching ε-simulation.

Theorem 3. *For two states $s,t \in S$ and $\varepsilon \in \mathbb{R}_{\geq 0}$:*

$$d(s,t) \leq \varepsilon \textit{ iff } s \leq_\varepsilon t$$

Proof. ($\Longrightarrow$) For this direction we prove that $R^\varepsilon = \{(s,t) \mid s,t \in S, d(s,t) \leq \varepsilon\}$ is a weighted branching ε-simulation relation. Suppose $(s,t) \in R^\varepsilon$. Then $d(s,t) \leq \varepsilon$ and by the fixed point property of d,

$$d(s,t) = \max_{s \rightarrow_w s'}\left\{\min_{t \rightarrow_{v_1} t_0 \cdots \rightarrow_{v_n} t_n}\left\{\max\left\{\begin{matrix}\left|\frac{\sum_{i=0}^n v_i}{w}-1\right|, \\ \max\{d(s',t_n)\} \cup \{d(s,t_i) \mid i<n\}\end{matrix}\right\}\right\}\right\}$$

We immediately have that for any transition $s \rightarrow_w s'$ there exists a matching transitions sequence $t \rightarrow_{v_1} t_0 \cdots \rightarrow_{v_n} t_n$ such that $\left|\frac{\sum_{i=0}^n v_i}{w}-1\right| \leq \varepsilon$, $d(s',t_n) \leq \varepsilon$ and $\forall i<n.d(s,t_i) \leq \varepsilon$. Thus, by definition of R^ε, for any transition $s \rightarrow_w s'$ there exists a sufficient matching sequence from t such that $(s',t_n) \in R^\varepsilon$ and $(s,t_i) \in R^\varepsilon$ for any $i<n$.

($\Longleftarrow$) Let

$$d^*(s,t) = \begin{cases} \varepsilon & \text{if } s \leq_\varepsilon t \\ \infty & \text{otherwise} \end{cases}$$

We now prove that d is a pre-fixed point of $\mathscr{F}$ i.e $\mathscr{F}(d^*)(s,t) \leq d^*(s,t)$ for any pair $(s,t) \in S$. If $s \not\leq_\varepsilon t$ then $d^*(s,t)=\infty$ and there is nothing to prove. If $s \leq_\varepsilon t$ then for any transition $s \rightarrow_w s'$ there exists a matching sequence $t \rightarrow_{v_1} t_0 \cdots \rightarrow_{v_n} t_n$ such that $\sum_{i=1}^n v_i \in [w(1-\varepsilon), w(1+\varepsilon)]$, $s' \leq_\varepsilon t_n$ and $s \leq_\varepsilon t_i$ for any $i<n$. We can now argue that

$$\max_{s \rightarrow_w s'}\left\{\min_{t \rightarrow_{v_1} t_0 \cdots \rightarrow_{v_n} t_n}\left\{\max\left\{\begin{matrix}\left|\frac{\sum_{i=0}^n v_i}{w}-1\right|, \\ \max\{d^*(s',t_n)\} \cup \{d^*(s,t_i) \mid i<n\}\end{matrix}\right\}\right\}\right\} \leq \varepsilon$$

as $\sum_{i=1}^n v_i \in [w(1-\varepsilon), w(1+\varepsilon)]$ is equivalent to $\left|\frac{\sum_{i=0}^n v_i}{w}-1\right| \leq \varepsilon$, $s' \leq_\varepsilon t_n$ implies $d^*(s',t_n)=\varepsilon$ and similarly $d^*(s,t_i)=\varepsilon$ for any $i<n$. As d^* is a pre-fixed point of $\mathscr{F}$ and $d^*(s,t)=\varepsilon$ it must be the case that $d(s,t) \leq \varepsilon$ as d is the *smallest* pre-fixed point of $\mathscr{F}$. □

Combining Theorem 1 and Theorem 3 we immediate get a relation between the distance from one state s to another state t and their $EWCTL_{-X}$ properties:

$$d(s,t) \leq \varepsilon \quad \text{iff} \quad \forall \varepsilon' \in \mathbb{Q}_{\geq 0}, \varepsilon \leq \varepsilon'.[\forall \phi \in EWCTL_{-X}.s \models \phi \implies t \models \phi^{\varepsilon'}.$$

4 Weighted Branching Simulation Distances for Parametric WKSs

We now extend WKS with parametric weights. The lifted parametric distance will be from a WKS to a parametric system and is represented as a parametric expression that can be evaluated to a rational by a *parameter valuation*. If one abstracts multiple configurations of the same system as one parametric system and calculate the parametric distance, evaluating the distance with respect to a parameter valuation then corresponds to calculating the exact distance from a specific configuration (given by the valuation) to the WKS. Thus, instead of working with multiple WKS configurations, one can use a parametric system and compute the parametric distance once.

A parametric weighted Kripke structure (PWKS) extends WKS by allowing transitions to have parametric weights. Let $\mathscr{P} = \{p_1, \ldots, p_n\}$ be a fixed finite set of parameters. A *parameter valuation* is a function mapping each parameter to a non-negative rational; $v : \mathscr{P} \to \mathbb{Q}_{\geq 0}$. The set of all such valuation will be denote by $\mathscr{V}$.

Definition 6 (Parametric Weighted Kripke Structure)**.** A *parametric weighted Kripke structure* is a tuple $\mathscr{K}_{\mathscr{P}} = (S, AP, \mathscr{L}, \to)$, where S is a finite set of states, AP is a set of atomic propositions, $\mathscr{L} : S \to \mathscr{P}(AP)$ is a mapping from states to sets of atomic propositions and $\to \subseteq S \times \mathscr{P} \cup \mathbb{Q}_{\geq 0} \times S$ the finite transition relation.

Unless otherwise specified, we assume a fixed PWKS $\mathscr{K}_{\mathscr{P}} = (S, AP, \mathscr{L}, \to)$ in the remainder of this section. One can instantiate a PWKS to a WKS by applying a parameter valuation. A PWKS thus represents an infinite set of WKSs.

Definition 7. Given a parameter valuation $v \in \mathscr{V}$, we define the *instantiated WKS* of $\mathscr{K}_{\mathscr{P}}$ under v to be $\mathscr{K}_{\mathscr{P}}^{v} = (S, AP, \mathscr{L}, \to_v)$ where

$$\to_v = \{(s, v(p), s') \mid (s, p, s') \in \to, p \in \mathscr{P}\} \cup \{(s, w, s') \mid (s, w, s') \in \to, w \in \mathbb{Q}_{\geq 0}\}$$

For a state s in $\mathscr{K}_{\mathscr{P}}$ let $s[v]$ be the corresponding state in the WKS $\mathscr{K}_{\mathscr{P}}^{v}$ and let $\leq_\varepsilon$ be lifted to disjoint unions of WKSs in the natural way.

Given a WKS state s, a PWKS state t and $\varepsilon \geq 0$ we now state three interesting problems:

1. Does there exist a $v \in \mathscr{V}$ such that $s \leq_\varepsilon t[v]$?
2. Can we characterize the set of "good" parameter valuation $V = \{v \mid v \in \mathscr{V}, s \leq_\varepsilon t[v]\}$?
3. Can we synthesize a valuation $v \in \mathscr{V}$ that minimizes ε for $s \leq_\varepsilon t[v]$?

We will show how to solve (2) by fixed point computations. The result will be a set of linear inequalities over parameters and ε which has as solution a set of parameter valuations. Instead of considering a concrete $\varepsilon \in \mathbb{R}_{\geq 0}$, one can let ε be an extra parameter. Thus, (1) and (3) can be solved by first solving (2) and applying e.g $Z3$ [18] and νZ [7] or similar tools to solve the inequalities and search for solutions that minimize ε.

Example 4. *Consider Figure 2. From Example 2 we know that $s \leq_{0.5} t[v]$ if $v(p) = 1$. Both $v(p) = 0$ and $v(p) = 2$ imply $s \leq_1 t[v]$. It turns out that $v(p) = 1$ is the valuation that minimizes ε for $s \leq_\varepsilon t[v]$.*

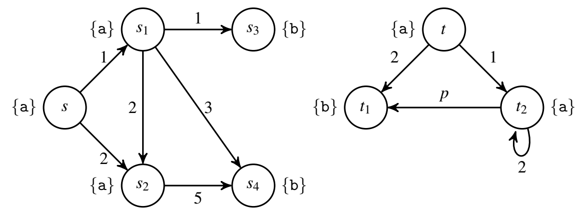


Figure 2: A WKS (left) and a PWKS (right)

If we were to validate a given parameter valuation we could simply apply the valuation to the PWKS and use $\mathscr{F}$ directly to decide if the distance is below some ε. As we want a full characterization of the good parameter valuation we will instead represent the distance as a parametric expression. For any two states s,t we will associate an expression $E_{s,t}$ such that the solution set to the inequality $E_{s,t} \leq \varepsilon$ characterizes the set of good parameter valuations i.e applying a parameter valuation to $E_{s,t}$ yields a concrete weighted distance. The syntactic elements for the expressions can be derived directly from $\mathscr{F}$; we need syntax for describing minimums of maximums of basic elements $\left|\frac{v}{w} - 1\right|$ where w is rational and v a linear expression in the parameters. Hence, we define the following abstract syntax:

$$E_1, E_2 ::= \left|\frac{v}{w} - 1\right| \mid \min\{E_1, E_2\} \mid \max\{E_1, E_2\}$$

where $w \in \mathbb{Q}_{\geq 0}$ and v is on the form $\sum_{i=0}^{n} a_i p_i + b$ s.t $a_i \in \mathbb{N}$ for all $i < n$ and $b \in \mathbb{Q}_{\geq 0}$. We extend parameter valuations to expressions in the obvious way and denote by $[\![E]\!](v)$ the value of E under $v \in \mathscr{V}$. Similar to disjunctive normal form for logical formulae, we can convert any expression to an equivalent expression being a min expression of max expressions of simple expressions $\left|\frac{v}{w} - 1\right|$. To convert an expression, note that for any $v \in \mathscr{V}$

$$[\![\max\{\min\{E_1, E_2\}, E_3\}]\!](v) = [\![\min\{\max\{E_1, E_3\}, \max\{E_2, E_3\}\}]\!](v)$$

The set of expression on this normal form will be denoted by $\mathscr{E}$. The following definitions concern the ordering of parametric distance function from the set $\mathscr{E}^{S \times S_{\mathscr{P}}}$. We define both a syntactic ordering and semantic ordering. The syntactic ordering is based on syntactically ordering elements from $\mathscr{E}$.

Definition 8. The syntactic ordering $\sqsubseteq_{\mathscr{E}} \subseteq \mathscr{E} \times \mathscr{E}$ is defined inductively on the structure of $\mathscr{E}$:

$$\left|\frac{\sum_{i=0}^{n} a_i p_i + b}{w} - 1\right| \sqsubseteq_{\mathscr{E}} \left|\frac{\sum_{i=0}^{n} a'_i p_i + b'}{w} - 1\right| \quad \text{iff} \quad \begin{cases} \forall i.a_i \leq a'_i \wedge b \leq b' & \text{if } \frac{b'}{w}, \frac{b}{w} \geq 1 \\ \forall i.a_i = a'_i \wedge b = b' & \text{otherwiese} \end{cases}$$

$$\max\{E_{1.1}, E_{1.2}\} \sqsubseteq_{\mathscr{E}} \max\{E_{2.1}, E_{2.2}\} \quad \text{iff} \quad \forall i.\exists j.E_{1.i} \sqsubseteq_{\mathscr{E}} E_{2.j}$$

$$\min\{E_{1.1}, E_{1.2}\} \sqsubseteq_{\mathscr{E}} \min\{E_{2.1}, E_{2.2}\} \quad \text{iff} \quad \forall j.\exists i.E_{1.i} \sqsubseteq_{\mathscr{E}} E_{2.j}$$

We extend this ordering to distance functions

Definition 9. The *syntactic* ordering on distance functions $\sqsubseteq_{\mathscr{E}}$ is defined for any $d^1_{\mathscr{P}}, d^2_{\mathscr{P}} \in \mathscr{E}^{S \times S_{\mathscr{P}}}$:

$$d^1_{\mathscr{P}} \sqsubseteq_{\mathscr{E}} d^2_{\mathscr{P}} \quad \text{iff} \quad \forall s,t \in S.d^1_{\mathscr{P}}(s,t) \sqsubseteq_{\mathscr{E}} d^2_{\mathscr{P}}(s,t).$$

We now define the semantic ordering of distance functions.

Definition 10. The *semantic* ordering on distance functions $\sqsubseteq_{[\![\mathscr{E}]\!]}$ is defined for any $d^1_{\mathscr{P}}, d^2_{\mathscr{P}} \in \mathscr{E}^{S \times S_{\mathscr{P}}}$:

$$d^1_{\mathscr{P}} \sqsubseteq_{[\![\mathscr{E}]\!]} d^2_{\mathscr{P}} \quad \text{iff} \quad \forall s,t \in S, v \in \mathscr{V}.[\![d^1_{\mathscr{P}}(s,t)]\!](v) \leq [\![d^2_{\mathscr{P}}(s,t)]\!](v).$$

By design of the ordering on expressions, the syntactic ordering implies the semantic ordering.

Lemma 3. *For arbitrary $d^1_{\mathscr{P}}, d^2_{\mathscr{P}} \in \mathscr{E}^{S \times S_{\mathscr{P}}}$:*

$$d^1_{\mathscr{P}} \sqsubseteq_{\mathscr{E}} d^2_{\mathscr{P}} \implies d^1_{\mathscr{P}} \sqsubseteq_{[\![\mathscr{E}]\!]} d^2_{\mathscr{P}}$$

When lifting the distance to the parametric setting, we consider disjoint unions of systems and require that only the simulating system can be parametric. Let $\mathscr{K}_{\mathscr{P}} = (S_{\mathscr{P}}, AP, \mathscr{L}_{\mathscr{P}}, \rightarrow^{\mathscr{P}})$ be a PWKS and $\mathscr{K} = (S, AP, \mathscr{L}, \rightarrow)$ a WKS. To compute the distance between states we now consider the function $\mathscr{F}_{\mathscr{P}} : \mathscr{E}^{S \times S_{\mathscr{P}}} \rightarrow \mathscr{E}^{S \times S_{\mathscr{P}}}$ being the trivial parametric analogue to $\mathscr{F}$ of Definition 5 for disjoint unions of WKSs and PWKSs. $\mathscr{F}_{\mathscr{P}}$ is monotone on the complete lattice $(\mathscr{E}^{S \times S_{\mathscr{P}}}, \sqsubseteq_{[\![\mathscr{E}]\!]})$ and therefore has a least fixed point computable by repeated application of $\mathscr{F}_{\mathscr{P}}$ on the bottom element $d^0_{\mathscr{P}}(s,t)$ where $\forall s \in S, t \in S_{\mathscr{P}}.d^0_{\mathscr{P}}(s,t) = 0$. We refer to this fixed point as the *semantic* least fixed point of $\mathscr{F}_{\mathscr{P}}$ and denote it by $d^{\min}_{\mathscr{P}}$. It follows that $[\![d^{\min}_{\mathscr{P}}(s,t)]\!](v) \leq \varepsilon \iff d(s,t[v]) \leq \varepsilon$ but the computation involves a quantification over all valuations in the semantic ordering. Instead we wish to compute an equivalent syntactic fixed point over expressions.

For WKS we assumed no 0-cycles to give an upper bound on the length of relevant transition sequences. Similarly for PWKS, we require all loops to have at least one strictly positive non-parametric weight in order to show computability of a syntactic fixed point of $\mathscr{F}_{\mathscr{P}}$.

Lemma 4. *Let $\mathscr{K} = (S, AP, \mathscr{L}, \rightarrow)$ be a WKS with state $s \in S$ such that $s \rightarrow_w s'$ and let $\mathscr{K}_{\mathscr{P}} = (S_{\mathscr{P}}, AP, \mathscr{L}_{\mathscr{P}}, \rightarrow^{\mathscr{P}})$ be a PWKS with the following property:*

- *There exists a $w_{\min} > 0$ such that for any valuation, the accumulated weight of every loop in $\mathscr{K}_{\mathscr{P}}$ is at least $w_{\min}$ (strongly cost non-zeno).*

Then for any $t \in S_{\mathscr{P}}$:

$$\begin{aligned}
&\exists N. \forall \pi = t \rightarrow^{\mathscr{P}}_{v_1} t_1 \ldots \rightarrow^{\mathscr{P}}_{v_n} t_n, n \geq N. \\
&\quad \exists \pi^* = t \rightarrow^{\mathscr{P}}_{u_1} t'_1 \ldots \rightarrow^{\mathscr{P}}_{u_m} t'_m, m \leq N. \\
&\quad t_n = t'_m \wedge \left| \frac{\sum_{i=0}^m u_i}{w} - 1 \right| \sqsubseteq_{\mathscr{E}} \left| \frac{\sum_{i=0}^n v_i}{w} - 1 \right| \wedge \\
&\quad \{t'_1, \ldots, t'_{m-1}\} \subseteq \{t_1, \ldots, t_{n-1}\}
\end{aligned}$$

Proof. Let the maximum weight out of s be $s_{w_{\max}}$. Any sequence of length $|S_{\mathscr{P}}|$ must have a loop which, by assumption, cannot have accumulated weight 0 w.r.t any parameter valuation. Thus, the accumulated weight w.r.t any valuation is at least $w_{\min}$. Without loss of generality we assume it to be exactly $w_{\min}$. Exercising the loop a number of times will at some point result in the accumulated weight being greater than $2 \cdot s_{w_{\max}}$ w.r.t any valuation. Let this sequence be $\pi^* = t \rightarrow^{\mathscr{P}}_{v_1} t_1 \cdots \rightarrow^{\mathscr{P}}_{v_k} t_k$ and let x denote the number of times the loop is exercised i.e $x \cdot w_{\min} \geq 2 \cdot s_{w_{\max}}$. Let $\sum_{i=0}^k v_i = \sum_{i=0}^n a_i p_i + b$. Then it is clear that $\frac{b}{s_{w_{\max}}} > 1$. Now consider the corresponding non-looping sequence $\pi_1 = t \rightarrow^{\mathscr{P}}_{u_1} t'_1 \cdots \rightarrow^{\mathscr{P}}_{u_l} t'_l$ and let $\sum_{i=0}^l u_i = \sum_{i=0}^n a'_i p_i + b'$. We would like it to be the case that

$$\left| \frac{\sum_{i=0}^n a'_i p_i + b'}{w} - 1 \right| \sqsubseteq_{\mathscr{E}} \left| \frac{\sum_{i=0}^n a_i p_i + b}{w} - 1 \right|$$

but it might be the case that $\frac{b'}{s_{w_{\max}}} < 1$. Consider a third sequence $\pi = t \rightarrow^{\mathscr{P}}_{x_1} t''_1 \cdots \rightarrow^{\mathscr{P}}_{x_m} t''_m$, being π^* modified to exercise the loop one more time and let $\sum_{i=0}^m x_i = \sum_{i=0}^n a''_i p_i + b''$. Now we know that $\frac{b''}{s_{w_{\max}}} > 1$

as $b'' > b'$ and furthermore $\left|\frac{\sum_{i=0}^{n} a'_i p_i + b'}{w} - 1\right| \sqsubseteq_{\mathscr{E}} \left|\frac{\sum_{i=0}^{n} a''_i p_i + b''}{w} - 1\right|, t_k = t''_m$ and $\{t'_1, \ldots, t_k\} \subseteq \{t''_1, \ldots, t''_m\}$. We can now derive N. For π^* we have the inequality $x \cdot w_{\min} \geq 2 \cdot s_{w_{\max}}$ and by Lemma 2 this leads to the bound $\frac{2 \cdot s_{w_{\max}}}{w_{\min}} \cdot |S_{\mathscr{P}}|$. As π is at most $|S_{\mathscr{P}}|$ longer than π^* we get

$$N \geq \frac{2 \cdot s_{w_{\max}}}{w_{\min}} \cdot |S_{\mathscr{P}}| + |S_{\mathscr{P}}|$$

□

Lemma 5. *There exists a natural number n such that $\mathscr{F}^n_{\mathscr{P}}(d^0_{\mathscr{P}})$ is a fixed point w.r.t $\sqsubseteq_{\mathscr{E}}$.*

The following theorem states that we can, by computing the syntactic fixed point, get a characterization of the good parameter valuations for $s \leq_\varepsilon t[v]$. Thus, the syntactic fixed point implies the semantic fixed point $d^{\min}_{\mathscr{P}}$.

Theorem 4. *$\mathscr{F}^n_{\mathscr{P}}(d^0_{\mathscr{P}})(s,t) = E_{s,t}$ has the following property:*

$$s \leq_\varepsilon t[v] \quad \textit{iff} \quad [\![E_{s,t}]\!](v) \leq \varepsilon$$

By computing the syntactic fixed point we thus get a characterization of the “good” parameter valuations as the solution set for the inequality $E_{s,t} \leq \varepsilon$.

Example 5. *Consider the WKS and PWKS from Figure 2. To compute $E_{s,t}$, let $d^i_{\mathscr{P}}(s,t) = \mathscr{F}^i_{\mathscr{P}}(d^0_{\mathscr{P}})(s,t)$. We now show how the distance from s to t is updated after each iteration. Note that although $d^1_{\mathscr{P}}(s,t) = d^2_{\mathscr{P}}(s,t)$, the fixed point is not reached as the distance between dependent pairs is still being refined.*

$$d^1_{\mathscr{P}}(s,t) = \max\left\{\begin{array}{l} \max\left\{\left|\frac{1}{1} - 1\right|, d^0_{\mathscr{P}}(s_1,t_2)\right\}, \\ \max\left\{\left|\frac{3}{2} - 1\right|, d^0_{\mathscr{P}}(s_1,t_2), d^0_{\mathscr{P}}(s,t_2)\right\} \end{array}\right\} = \frac{1}{2}$$

$$d^2_{\mathscr{P}}(s,t) = \max\left\{\begin{array}{l} \max\left\{\left|\frac{1}{1} - 1\right|, 0\right\}, \\ \max\left\{\left|\frac{3}{2} - 1\right|, 0, \frac{1}{2}\right\} \end{array}\right\} = \frac{1}{2}$$

$$d^3_{\mathscr{P}}(s,t) = \max\left\{\begin{array}{l} \frac{1}{2}, \\ \min\left\{\left|\frac{p}{5} - 1\right|, \left|\frac{p+2}{5} - 1\right|, \left|\frac{p+4}{5} - 1\right|\right\}, \\ \min\left\{\left|\frac{p}{3} - 1\right|, \left|\frac{p+2}{3} - 1\right|\right\}, \\ \min\left\{\left|\frac{p}{1} - 1\right|, \left|\frac{p+2}{1} - 1\right|\right\} \end{array}\right\}$$

$$d^4_{\mathscr{P}}(s,t) = d^3_{\mathscr{P}}(s,t)$$

We immediately see that any solution to $E_{u,v} \leq \varepsilon$ is bounded from below by $\frac{1}{2}$. This implies that there exists no valuation $v \in \mathscr{V}$ such that $s \leq_\varepsilon t[v]$ for $\varepsilon < \frac{1}{2}$. If we consider the valuation $v_{\min}(p) = 1$ we get that $[\![E_{s,t}]\!](v_{\min}) = \frac{1}{2}$ i.e $v_{\min}$ is the valuation that induces the minimal distance $d(s,t[v_{\min}]) = \frac{1}{2}$.

5 Conclusion and Future Work

We have characterized the distance from [11] between weighted Kripke structures (WKS) as a least fixed point. The distance between any pair of states can thus be computed by first assuming the distance between any pair to be 0 and then applying a step-wise refinement of the distance. The computability of the distance is guaranteed as a finite number of the (potentially) infinite transition sequences of the system is sufficient. This we proved by demonstrating an upper bound on the relevant sequences. We furthermore lifted the distance to parametric WKS (PWKS), where transition weights can be parametric. The parameters can be used to abstract multiple configurations of the same system as one parametric system. In this case the distance is from a WKS to a PWKS and is concretely a parametric expression that one can evaluate to get an exact distance from the WKS to a specific WKS instance of the PWKS. The question is then which configuration (parameter valuation) is "best" i.e minimizes the induced distance. For computability we again demonstrate an upper bound on the length of relevant distances. To do this we assume all cycles to be cost non-zeno i.e any loop must include a transition with a positive rational weight.

For future work, the actual complexity of computing the distance is open. From [11] we know that checking whether the distance is 0 is NP-complete but the general complexity of checking whether the distance is less than some $\varepsilon \in \mathbb{R}_{\geq 0}$ is open. One could also investigate whether the distance has a polynomial approximation scheme.

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. Available at `http://dx.doi.org/10.1109/TSE.2008.106`.

[2] Rajeev Alur & David L. Dill (1990): *Automata For Modeling Real-Time Systems*. In: *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, July 16-20, 1990, Proceedings*, pp. 322–335, doi:10.1007/BFb0032042. Available at `http://dx.doi.org/10.1007/BFb0032042`.

[3] Rajeev Alur, Thomas A. Henzinger & Moshe Y. Vardi (1993): *Parametric real-time reasoning*. In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pp. 592–601, doi:10.1145/167088.167242. Available at `http://doi.acm.org/10.1145/167088.167242`.

[4] Étienne André, Thomas Chatain, Laurent Fribourg & Emmanuelle Encrenaz (2009): *An Inverse Method for Parametric Timed Automata*. *Int. J. Found. Comput. Sci.* 20(5), pp. 819–836, doi:10.1142/S0129054109006905. Available at `http://dx.doi.org/10.1142/S0129054109006905`.

[5] Étienne André, Yang Liu, Jun Sun & Jin Song Dong (2014): *Parameter synthesis for hierarchical concurrent real-time systems*. *Real-Time Systems* 50(5-6), pp. 620–679, doi:10.1007/s11241-014-9208-6. Available at `http://dx.doi.org/10.1007/s11241-014-9208-6`.

[6] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn & Frits W. Vaandrager (2001): *Minimum-Cost Reachability for Priced Timed Automata*. In: *Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Rome, Italy, March 28-30, 2001, Proceedings*, pp. 147–161, doi:10.1007/3-540-45351-2_15. Available at `http://dx.doi.org/10.1007/3-540-45351-2_15`.

[7] Nikolaj Bjørner, Anh-Dung Phan & Lars Fleckenstein (2015): *νZ - An Optimizing SMT Solver*. In: *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, pp. 194–199, doi:10.1007/978-3-662-46681-0_14. Available at `http://dx.doi.org/10.1007/978-3-662-46681-0_14`.

[8] Patricia Bouyer, Ulrich Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey & Jirí Srba (2008): *Infinite Runs in Weighted Timed Automata with Energy Constraints*. In: *Formal Modeling and Analysis of Timed Systems, 6th International Conference, FORMATS 2008, Saint Malo, France, September 15-17, 2008.*

Proceedings, pp. 33–47, doi:10.1007/978-3-540-85778-5_4. Available at `http://dx.doi.org/10.1007/978-3-540-85778-5_4`.

[9] Peter Christoffersen, Mikkel Hansen, Anders Mariegaard, Julian Trier Ringsmose, Kim Guldstrand Larsen & Radu Mardare (2015): *Parametric Verification of Weighted Systems*. In Étienne André & Goran Frehse, editors: *2nd International Workshop on Synthesis of Complex Parameters (SynCoP'15)*, *OpenAccess Series in Informatics (OASIcs)* 44, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 77–90, doi:10.4230/OASIcs.SynCoP.2015.77. Available at `http://drops.dagstuhl.de/opus/volltexte/2015/5611`.

[10] E. Allen Emerson & Richard J. Trefler (1999): *Parametric Quantitative Temporal Reasoning*. In: *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pp. 336–343, doi:10.1109/LICS.1999.782628. Available at `http://dx.doi.org/10.1109/LICS.1999.782628`.

[11] Louise Foshammer, Kim Guldstrand Larsen & Bingtian Xue (2016): *Logical Characterization and Complexity of Weighted Branching Preorders and Distances*. In: *Proceedings of The Seventh International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking, COMPUTATION TOOLS 2016, Rome, Italy, March 20-24, 2016.*, IARIA XPS Press, p. To Appear.

[12] Rob J. van Glabbeek & W. P. Weijland (1996): *Branching Time and Abstraction in Bisimulation Semantics*. *J. ACM* 43(3), pp. 555–600, doi:10.1145/233551.233556. Available at `http://doi.acm.org/10.1145/233551.233556`.

[13] Ernst Moritz Hahn, Tingting Han & Lijun Zhang (2011): *Synthesis for PCTL in Parametric Markov Decision Processes*. In: *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*, pp. 146–161, doi:10.1007/978-3-642-20398-5_12. Available at `http://dx.doi.org/10.1007/978-3-642-20398-5_12`.

[14] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter & Lijun Zhang (2010): *PARAM: A Model Checker for Parametric Markov Models*. In: *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, pp. 660–664, doi:10.1007/978-3-642-14295-6_56. Available at `http://dx.doi.org/10.1007/978-3-642-14295-6_56`.

[15] Michal Knapik & Wojciech Penczek (2014): *Parameter Synthesis for Timed Kripke Structures*. *Fundam. Inform.* 133(2-3), pp. 211–226, doi:10.3233/FI-2014-1072. Available at `http://dx.doi.org/10.3233/FI-2014-1072`.

[16] Kim G. Larsen, Uli Fahrenberg & Claus R. Thrane (2011): *Metrics for weighted transition systems: Axiomatization and complexity*. *Theor. Comput. Sci.* 412(28), pp. 3358–3369, doi:10.1016/j.tcs.2011.04.003. Available at `http://dx.doi.org/10.1016/j.tcs.2011.04.003`.

[17] Robin Milner (1989): *Communication and concurrency*. PHI Series in computer science, Prentice Hall.

[18] Leonardo Mendonça de Moura & Nikolaj Bjørner (2008): *Z3: An Efficient SMT Solver*. In: *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pp. 337–340, doi:10.1007/978-3-540-78800-3_24. Available at `http://dx.doi.org/10.1007/978-3-540-78800-3_24`.

[19] David Park (1981): *Theoretical Computer Science: 5th GI-Conference Karlsruhe, March 23–25, 1981*, chapter Concurrency and automata on infinite sequences, pp. 167–183. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/BFb0017309. Available at `http://dx.doi.org/10.1007/BFb0017309`.

[20] Alfred Tarski et al. (1955): *A lattice-theoretical fixpoint theorem and its applications*. *Pacific journal of Mathematics* 5(2), pp. 285–309.

[21] Claus R. Thrane, Uli Fahrenberg & Kim G. Larsen (2010): *Quantitative analysis of weighted transition systems*. *J. Log. Algebr. Program.* 79(7), pp. 689–703, doi:10.1016/j.jlap.2010.07.010. Available at `http://dx.doi.org/10.1016/j.jlap.2010.07.010`.