IST AUSTRIA

*Institute of Science and Technology*

# From model checking to model measuring

Thomas A Henzinger, Jan Otop

https://doi.org/10.15479/AT:IST-2014-172-v1-1

Deposited at: 12 Dec 2018 11:53 ISSN: 2664-1690

IST Austria (Institute of Science and Technology Austria)
Am Campus 1
A-3400 Klosterneuburg, Austria

# From Model Checking to Model Measuring*

Thomas A. Henzinger and Jan Otop

IST Austria

**Abstract.** We define the *model-measuring problem*: given a model $M$ and specification $\varphi$, what is the maximal distance $\rho$ such that all models $M'$ within distance $\rho$ from $M$ satisfy (or violate) $\varphi$. The model measuring problem presupposes a distance function on models. We concentrate on *automatic distance functions*, which are defined by weighted automata. The model-measuring problem subsumes several generalizations of the classical model-checking problem, in particular, quantitative model-checking problems that measure the degree of satisfaction of a specification, and robustness problems that measure how much a model can be perturbed without violating the specification. We show that for automatic distance functions, and $\omega$-regular linear-time and branching-time specifications, the model-measuring problem can be solved. We use automata-theoretic model-checking methods for model measuring, replacing the emptiness question for standard word and tree automata by the *optimal-weight question* for the weighted versions of these automata. We consider weighted automata that accumulate weights by maximizing, summing, discounting, and limit averaging. We give several examples of using the model-measuring problem to compute various notions of robustness and quantitative satisfaction for temporal specifications.

## 1 Introduction

Model-checking techniques have proved to be very useful in automatic verification. Typically, the verified system is modeled as a transition system, the desired properties are specified by a formula in a temporal language (Linear Temporal Logic [LTL], Computation Tree Logic[CTL]) or an $\omega$-automaton, and a model-checking algorithm decides whether the model is correct with respect to the specification. However, knowing whether the model is correct or not, is often insufficient.

Consider the TCP handshake protocol, which is used to establish a connection between a client and a server. First, the client sends a SYN packet to the server, which replies with a SYN-ACK packet. Then, the client responds with an ACK packet. A TCP connection is established, provided that the protocol terminated.

Termination of the protocol can be verified by the standard model-checking techniques, when the communication channel is assumed to be reliable, that is, every sent packet is delivered in the next step. Certain faults, such as "the first server response SYN-ACK gets lost" can be encoded in the model. But, this raises doubts whether the model includes all communication faults. Another approach would be to use fairness assumptions, for example "if infinitely many packets are sent, infinitely many packets will be delivered". But, such assumptions may be too weak to guarantee termination of the protocol. We propose a more refined, quantitative approach.

We assume that any packet may get lost, but we ask quantitative questions: What is the maximal number of lost packets tolerated by the protocol? What is the maximal ratio of lost packets that guarantees liveness of the system? Such questions are instances of the model-measuring problem.

The *model-measuring problem* asks, given a model $M$ and specification $\varphi$, what is the maximal distance $\rho$ such that all models $M'$ within that distance from $M$ satisfy (or violate) $\varphi$. That distance $\rho$ is called the *stability radius*. Figure 1 presents a geometric interpretation of the stability radius in two cases, a model $M$ that satisfies the specification and a model $N$ that violates it.

To determine the stability radius, it suffices to have a unary function that, for a given transition system $M'$, specifies its distance from $M$. Such a function, called a *similarity measure*, is a sole input to the model-measuring problem. As inputs are required to be finite, we are interested in *automatic* similarity measures that are represented by weighted automata.

In the TCP handshake protocol example, a model $N$ encodes all executions of the protocol over a reliable channel. Next, we define a similarity measure $d_N$ so that $d_N(M) = k$ if $M$ encodes the TCP handshake protocol that loses (up to) $k$ packets during its execution. Then, for the

* This work was supported in part by the Austrian Science Fund NFN RiSE (Rigorous Systems Engineering) and by the ERC Advanced Grant QUAREM (Quantitative Reactive Modeling).

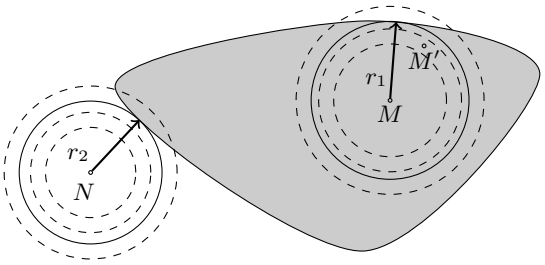


**Fig. 1.** A geometric interpretation of stability radii $r_1$ of a model $M$ and $r_2$ of a model $N$. The shaded area represents the family of models satisfying the specification.

specification "the protocol terminates", the model-measuring problem answers the question, what is the maximal number of lost packets that guarantees termination of the protocol?

We represent similarity measures by weighted automata; the representation depends on the type of the specification. For example, in the branching-time case, every transition system (model) $M'$ admits the unique unrolling to a tree $t_{M'}$. Then, a weighted automaton $\mathcal{A}_{dist}$ represents a similarity measure $d_M$, if for every transition system $M'$, $d_M(M')$ equals to $\mathcal{A}_{dist}(t_{M'})$, the weight of $t_{M'}$ assigned by $\mathcal{A}_{dist}$.

Similarity measures represented by weighted automata are invariant with respect to bisimilarity. This design choice is not accidental as we think that two systems should be considered similar when their outputs are similar rather than when the internal structures are similar. After all, we would consider two different implementations of the same algorithm as similar rather than two similar programs that implement different algorithms.

Having an automatic representation of $d_M$, we can solve the model-measuring problem. Returning to the branching-time case, a (qualitative) automata-theoretic CTL model-checking procedure works as follows. It translates $\neg\varphi$ and $M$ to $\omega$-tree automata $\mathcal{A}_{\neg\varphi}, \mathcal{A}_M$, where $\mathcal{A}_{\neg\varphi}$ recognizes the set of all trees that satisfy $\neg\varphi$ and $\mathcal{A}_M$ accepts only a single tree, the unrolling of $M$. Then, it asks for emptiness of $\mathcal{L}(\mathcal{A}_{\neg\varphi} \times \mathcal{A}_M) = \mathcal{L}(\mathcal{A}_{\neg\varphi}) \cap \mathcal{L}(\mathcal{A}_M)$. In our approach, we replace $\mathcal{A}_M$ by a weighted $\omega$-tree automaton $\mathcal{A}_{dist}$ representing $d_M$, and generalize the emptiness question to its weighted counterpart, the *optimal-weight question*. That question asks for the infimum over weights of all $\omega$-trees ($\omega$-words) accepted by a weighted $\omega$-tree automaton. Now, let $\rho$ be the answer to the optimal weight question for $\mathcal{A}_{\neg\varphi} \times \mathcal{A}_{dist}$. It follows that for every $\rho' > \rho$, there is a tree accepted by $\mathcal{A}_{\neg\varphi}$ of weight at most $\rho'$, and every $M'$, whose distance from $M$ is less than $\rho$, satisfies $\varphi$. Thus, $\rho$ is the stability radius of $\varphi$ in $M$. Virtually the same argument can be repeated in the linear-time case using $\omega$-automata and $\omega$-words.

The contribution of this paper is two-fold. First, we define the model-measuring framework (Section 3) and show that several problems studied in the literature are special cases of the model-measuring problem. Second, we give a systematic approach to modeling similarity measures using weighted automata, and corresponding algorithms based on the optimal-weight question for computing them.

The paper is organized as follows. In Section 2 we recall the standard notions of weighted and unweighted automata, define the optimal weight question and discuss its complexity in various cases. In Section 3 we define the stability radius of a model w.r.t. a specification and the model-measuring problem. We start with general definitions of these notions, which are then specialized to the $\omega$-regular linear-time and branching-time settings, based on weighted automata. Finally, in Section 4 we discuss in depth the modeling of similarity measures and give several examples in each case.

*Related work.* In recent years, much attention has been given to quantitative[1] generalizations of the Boolean notion of correctness and the corresponding quantitative verification questions [2, 3, 14, 15, 18]. Here we attempt to define a unifying automata-theoretic framework to capture and compute various ways of measuring model quantities. In particular, we have succeed in subsuming the following approaches.

The robust satisfaction of an open system has been studied in [14, 18]. An open system $M$ *robustly satisfies* a CTL specification $\varphi$ (according to [14]) if and only if for every environment,

[1] Note that we use the attribute "quantitative" in a non-probabilistic sense. We therefore restrict ourselves to list only non-probabilistic references.

given as an open system $M'$, the composition $M \parallel M'$ satisfies $\varphi$ (refer to [14] for the formal definition of composition). The model-measuring problem subsumes this notion of robustness (cf. Section 4).

The model-measuring problem can express *mutations on circuits* [17]. Indeed, all mutations considered in [17] just modify transition relations of automata, therefore they can be expressed by our *hypervisor* approach (cf. Example 24). In consequence, the model-measuring problem subsumes vacuity [19], coverage [10], and certain cases of fault tolerance [11].

Another approach to robustness of discrete systems has been presented in [3], where the *robustness distance* has been defined. This robustness distance can be expressed in our framework as well (cf. Proposition 25).

## 2 Preliminaries

A tree ($\omega$-tree) $t$ over $\Gamma$ labeled by $\Sigma$ is a pair $(\tau, L)$, where $\tau$ is a finite (infinite for $\omega$-trees) prefix-closed subset of $\Gamma^*$ and $L : \tau \mapsto \Sigma$ is a labeling function. For $\sigma \in \tau$, every extension $\sigma \cdot g$ of $\sigma$, where $g \in \Gamma$ and $\sigma \cdot g \in \tau$, is a *successor* of $\sigma$ in $(\tau, L)$. We write $\sigma \in t$ and $t(\sigma)$ instead of $\sigma \in \tau$ and $L(\sigma)$. We usually omit $\Gamma$.

A *labeled transition system* is a quadruple $\langle S, \Sigma, E, s_0\rangle$, where $S$ is a (finite or infinite) set of states, $\Sigma$ is an alphabet, $E$ is a relation on $S \times \Sigma \times S$ and $s_0$ is an initial state. All models considered in this paper are (finite or infinite) transition systems. A word (or $\omega$-word) $w = a_1 a_2 \dots$ is a *trace* of a labeled transition system $M$ if there is an (unlabeled) path $s_0 s_1 \dots$ in $M$ such that for every $i \in [1, |w|]$, $(s_{i-1}, a_i, s_i) \in E$. We say that an ($\omega$-)tree $(\tau, L)$ (over $S \times (\Sigma \cup \{\epsilon\})$) labeled by $\Sigma \cup \{\epsilon\}$ is the *unrolling* of a transition system $M = \langle S, \Sigma, E, s_0\rangle$ if $\tau$ is the union of all finite labeled paths $\langle s_0, \epsilon\rangle\langle s_1, a_1\rangle \dots \langle s_k, a_k\rangle$ through $M$ such that for every $i \in [1, k]$, $(s_{i-1}, a_i, s_i) \in E$, and $L(\langle s_0, \epsilon\rangle \dots \langle s_k, a_k\rangle) = a_k$.

### 2.1 Automata

A *(nondeterministic) automaton* is a tuple $(\Sigma, Q, Q_0, \delta, F)$, where $\Sigma$ is an alphabet, $Q$ is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation and $F$ is an acceptance condition (finite, Büchi, ...).

A *run* $\pi$ of an automaton $\mathcal{A}$ on $w = a_1 a_2 \dots$ is a sequence of states such that $\pi(0) \in Q_0$ and for every $i \in [1, |w|]$, $(q_{i-1}, a_i, q_i) \in \delta$. A run $\pi$ is *accepting* if it satisfies the acceptance condition $F$, e.g., in the Büchi case: there is $q \in F$ that occurs infinitely often in $\pi$.

A *(nondeterministic) ($\omega$-)tree automaton with varying degree* (bounded by $N$) [20] is a tuple $(\Sigma, Q, Q_0, \delta, F)$, where $\Sigma$ is an alphabet, $Q$ is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta \subseteq \bigcup_{k=1}^{N} (Q \times \Sigma) \times Q^k$ is a transition relation and $F$ is an acceptance condition (finite, Büchi, parity, ...).

A *run* $\pi$ of an automaton $\mathcal{A}$ on an ($\omega$-)tree $t = (\tau, L)$ is an ($\omega$-)tree $(\tau, L')$ labeled by $Q$ such that $\pi(\epsilon) \in Q_0$ and for every $\sigma \in t$, if $deg(\sigma) = k$ and $\sigma \cdot g_1, \dots, \sigma \cdot g_k$ are all successors of $\sigma$ in $t$, then $(\pi(\sigma), t(\sigma), \langle \pi(\sigma \cdot g_1), \dots, \pi(\sigma \cdot g_k)\rangle) \in \delta$. A run $\pi$ is *accepting* if it satisfies the acceptance condition $F$, e.g., in the Büchi case: along every infinite path there is a state from $F$ that occurs infinitely often.

A weighted (Büchi, parity, Büchi-tree, ...) automaton is an automaton whose transitions are labeled by natural numbers called weights. Formally, a weighted automaton $\mathcal{A}$ is a tuple $(\Sigma, Q, Q_0, \delta, F, C)$ such that $(\Sigma, Q, Q_0, \delta, F)$ is an automaton and $C : \delta \mapsto \mathbb{N}$.

A *weighting scheme* is a function that maps runs to real numbers, called *weights*. The weight of an $\omega$-word $w$ ($\omega$-tree $t$) assigned by the automaton $\mathcal{A}$ according to a weighting scheme $f$, denoted by $L^f_{\mathcal{A}}(w)$, is the infimum of the set of weights of all accepting runs of $\mathcal{A}$ on the $\omega$-word $w$ (the $\omega$-tree $t$) weighted by $f$. $\omega$-words ($\omega$-trees) that are rejected by $\mathcal{A}$ have infinite weight. Often, a particular weighting scheme is irrelevant in reasoning, as long as it is fixed through a proof; in such cases we shall omit it.

The emptiness question for non-weighted automata extends to the following question in the weighted case:

**Definition 1.** *Let $f$ be a weighting scheme. The* optimal-weight question *for $f$ asks, given a weighted automaton $\mathcal{A}$, to compute the infimum of $L^f_{\mathcal{A}}(w)$ over all $\omega$-words ($\omega$-trees).*

*Remark 2.* The dual to the optimal-weight question is to find the supremum of $L^f_{\mathcal{A}}(w)$ over all $\omega$-words $w$. Its decision versions have been referred to as the limitedness problem [21] or the universality problem for weighted automata [7]. They are usually much harder than the optimal-weight problem (see [5] for undecidability results).

### 2.2 Weighting schemes for $\omega$-words

Let $\mathcal{A}$ be a weighted automaton and $\pi$ be its run. Denote by $wt(\pi, i)$ the weight of the $i$th transition in $\pi$. We consider the following weighting schemes:

1. $\textsc{Sum}(\pi) = \sum_{i=1}^{\infty} wt(\pi, i)$, the sum,
2. $\textsc{Max}(\pi) = \max_{i=1}^{\infty} wt(\pi, i)$, the maximum,
3. $\textsc{Disc}_\lambda(\pi) = (1-\lambda)\sum_{i=1}^{\infty} \lambda^i wt(\pi, i)$, the discounted sum, where $\lambda \in (0, 1)$ is a *discount factor*,
4. $\textsc{LimAvg}(\pi) = \liminf_{k\to\infty} \frac{1}{k} \sum_{i=1}^{k} wt(\pi, i)$, the limit average.

These weighting schemes admit efficient algorithms computing the optimal-weight question:

**Theorem 3.** *([13, 22]) Let $f$ be one of* Sum, Max, $\textsc{Disc}_\lambda$, LimAvg. *The optimal-weight question for $f$ and a weighted Büchi automaton $\mathcal{A}$ can be computed in polynomial time in $|\mathcal{A}|$.*

### 2.3 Weighting schemes for $\omega$-trees

In the $\omega$-tree case, we consider two families of weighting schemes, Sup and Acc. The Sup weighing schemes are derived from $\omega$-words weighting schemes; every path in a run on an $\omega$-tree is weighted according to an $\omega$-words weighting scheme, and the weight of the run is supremum over weights of its all paths. We consider the following Sup weighting schemes: SupSum, SupMax, $\textsc{SupDisc}_\lambda$ and SupLimAvg.

The Acc family is obtained by accumulating weights over all paths. Given a run $\pi$ over an $\omega$-tree $t$ and $\sigma \in t$, we define: ***(i)*** $wt(\pi, \sigma)$ as the weight of a transition at $\sigma$ in the run $\pi$, ***(ii)*** the contribution of $\sigma$ in $\pi$, denoted by $\mu(\pi, \sigma)$, as follows: $\mu(\pi, \epsilon) = 1$, and for every successor $\sigma \cdot g$ of $\sigma$, $\mu(\pi, \sigma \cdot g) = \frac{1}{deg(\sigma)}\mu(\pi, \sigma)$.

We define the following weighting schemes:

1. $\textsc{AccSum}(\pi) = \sum_{\sigma\in\pi} \mu(\pi, \sigma) wt(\pi, \sigma)$, the accumulated sum,
2. $\textsc{AccDisc}_\lambda(\pi) = (1-\lambda)\sum_{\sigma\in\pi} \lambda^{|\sigma|}\mu(\pi, \sigma) wt(\pi, \sigma)$, the accumulated discounted sum, where $\lambda \in (0, 1)$ is a *discount factor*,
3. $\textsc{AccLimAvg}(\pi) = \liminf_{k\to\infty} \frac{1}{k} \sum_{\sigma\in\pi, |\sigma|\le k} \mu(\pi, \sigma) wt(\pi, \sigma)$, the accumulated limit average.

**Theorem 4.** *([1, 4, 6, 8, 9, 23]) Let $f$ be one of* SupSum, SupMax, $\textsc{SupDisc}_\lambda$, SupLimAvg, AccSum, $\textsc{AccDisc}_\lambda$ *or* AccLimAvg. *The optimal-weight question for $f$ and a weighted Büchi-tree automaton $\mathcal{A}$ can be computed in polynomial time in $|\mathcal{A}|$.*

| | $\omega$-words | $\omega$-trees | |
|---|---|---|---|
| | | Sup | Acc |
| Sum | $O(n \log n)$ | PTIME | PTIME |
| Max | $O(n \log n)$ | PTIME | — |
| $\textsc{Disc}_\lambda$ | PTIME | PTIME$^{(*)}$ | PTIME |
| LimAvg | PTIME | PTIME$^{(*)}$ | PTIME |

**Table 1.** The complexity of the optimal-weight question for weighted Büchi and Büchi-tree automata. $(*)$ indicates that the algorithm work in polynomial time under assumption that the weights are given in unary notation.

*Remark 5.* The optimal-weight question can be solved for parity $\omega$-word and $\omega$-tree automata with all weighting schemes from Theorems 3 and 4. However, its complexity in the parity case increases from PTIME to the complexity of solving parity games.

### 2.4 Automatic (weighted) relations

The *convolution* of $\omega$-words $w_1, w_2$, denoted by $w_1 \otimes w_2$, is an $\omega$-word over $\Sigma \times \Sigma$ such that the $i$th letter of $w_1 \otimes w_2$ is a pair of the $i$th letters of $w_1, w_2$.

A *weighted relation* is a generalization of the usual relation by allowing the characteristic function to range over $\mathbb{R}^+ \cup \{\infty\}$. A (binary) weighted relation $S$ is an *automatic weighted relation* if there is a weighted automaton $\mathcal{A}_S$ that *computes* $S$, i.e., for all $w_1, w_2 \in \Sigma^*$, $w_1 S w_2 = \mathcal{A}_S(w_1 \otimes w_2)$.

The notion of automatic weighted relations straightforwardly extends on $\omega$-trees.

## 3 The Model-Measuring Framework

Correctness of a system w.r.t. a specification is, like membership, a qualitative property; the system is correct or not. However, membership of a point $p$ in a region $R$ has a natural quantitative extension called the *stability radius*. It is defined as the distance between $p$ and the border of $R$ (cf. Fig. 1). It has been widely used in the decision-making community [16]. Assuming that we are given a distance function $d$ defined on transition systems, we adapt the stability radius to the model-checking setting. Basically, we ask for stability radius of a transition system in the region of all transition systems satisfying a specification.

The definitions in this section are independent of a particular logic. They refer to a *specification* which is not yet instantiated. It will be instantiated in Sections 3.1 and 3.2.

**Definition 6.** *Let $d$ be a distance defined on transition systems. For a transition system $M$ and a specification $P$, the* stability radius *of $P$ in $M$ (w.r.t. the distance $d$), denoted by $sr_d(M, P)$, is defined as follows:*

*(i)* *if $M \models P$, $sr_d(M, P) = \sup\{\rho \geq 0 : \forall M'(d(M, M') < \rho \Rightarrow M' \models P)\}$,*
*(ii)* *if $M \models \neg P$, $sr_d(M, P) = sr_d(M, \neg P)$,*
*(iii)* *otherwise, $sr_d(M, P) = 0$.*

In order to determine the stability radius of $P$ in $M$ we only need to know distances between a (fixed) $M$ and other transition systems; it suffices to have a unary function $d_M$, defined as $d_M(M') = d(M, M')$, which encodes essential information about $M$ and $d$. We call such a function $d_M$ a *similarity measure*. Observe that any function satisfying $d_M(M) = 0$ and $d_M(M') \geq 0$ is a valid similarity measure as we can find a distance $d$ defining it. However, we are interested only in similarity measures that are semantically defined, i.e., those that depend only on the behavior of the transition system (the set of traces), not on its structure.

We define the stability radius of $P$ in $M$ w.r.t. a similarity measure $d_M$, $sr_{d_M}(M, P)$, as the stability radius w.r.t. any distance compatible with $d_M$.

We define the *model-measure* on the basis of the stability radius by scaling the value the stability radius from $[0, \infty]$ to $[\frac{1}{2}, 1]$ if $M \models P$, and $[0, \frac{1}{2}]$ otherwise.

**Definition 7.** *The* model-measuring problem *is defined as follows: given a similarity measure $d_M$ and a specification $P$, compute $[P]_{d_M}$ defined as follows:*

*(i)* *if $M \models P$, $[P]_{d_M} = 1 - 2^{-sr_{d_M}(M,P)-1}$* $(\in [\frac{1}{2}, 1])$,
*(ii)* *if $M \models \neg P$, $[P]_{d_M} = 1 - [\neg P]_{d_M}$* $(\in [0, \frac{1}{2}])$,
*(iii)* *otherwise, $[P]_{d_M} = \frac{1}{2}$.*

Consider a specification given by a temporal (LTL or CTL) formula $\varphi$. The model-measure is compatible with conjunction and implication, i.e., $[\varphi_1 \wedge \varphi_2]_{d_M} = min([\varphi_1]_{d_M}, [\varphi_2]_{d_M})$ and $\varphi_1 \Rightarrow \varphi_2$ implies $[\varphi_1]_{d_M} \leq [\varphi_2]_{d_M}$. Observe that for every similarity measure $d_M$, $[\varphi]_{d_M} = 1$ if $\varphi$ is a tautology, as $sr_{d_M}(M, \varphi) = \infty$ and $1 - 2^{-\infty-1} = 1$, and $[\varphi]_{d_M} = 0$ if $\varphi$ is inconsistent. Values of formulae that are neither tautologies nor inconsistent depend on the choice of a similarity measure.

*Example 8.* Consider a transition system $M$ modeling two parties communicating through a channel, where every sent packet is delivered in the next state. We define a similarity measure $d_M$, such that $d_M(M') = k$ if $M'$ models two parties that follow the same protocol as in $M$, but up to $k$ packets sent through the channel get lost. We shall return to this example in Section 4.

In the following, we discuss specialization of the model-measuring problem for $\omega$-regular linear-time and branching-time specifications.

### 3.1 Model measuring $\omega$-regular linear-time specifications

An *$\omega$-regular linear-time specification* $P$ is a subset of $\Sigma^\omega$, the set of all correct traces. We assume that $P$ is given by a Büchi automaton $\mathcal{A}_P$ recognizing its complement, i.e., an $\omega$-word $w$ violates $P$ iff $\mathcal{A}_P$ accepts $w$. E.g. a Linear Temporal Logic (LTL) formula $\varphi$ can be translated to a Büchi automaton $\mathcal{A}_{\neg\varphi}$ recognizing $\omega$-words that satisfy $\neg\varphi$. The automaton $\mathcal{A}_P$ can be regarded as a weighted automaton with all weights 0. Next, we say that a transition system $M$ satisfies a linear-time specification $P$ if all its traces satisfy $P$, or equivalently, the language of all traces of $M$ and $\mathcal{L}(\mathcal{A}_P)$ are disjoint.

We proceed alike with similarity measures. We define similarity measures on $\omega$-words, then we extend the definition to transition systems.

**Definition 9.** *A (linear-time) similarity measure $d_M$ is* automatic *iff there is a weighted automaton $\mathcal{A}_{dist}$ and a weighting scheme $f \in \{\textsc{Sum}, \textsc{Max}, \textsc{Disc}_\lambda, \textsc{LimAvg}\}$ such that for every transition system $M'$, $d_M(M') = \sup\{\mathcal{A}^f_{dist}(w) : w \text{ is a trace of } M'\}$.*

*Example 10.* Consider a finite transition system $M$. Let $\mathcal{A}_{dist}$ be a weighted automaton that contains $M$ and has a single additional state $q_\perp \notin M$. There are transitions, labeled by every letter, from every state of $\mathcal{A}_{dist}$ to $q_\perp$; each such transition has the weight 1. The state $q_\perp$ is accepting, but it has only self-loops of weight 1. All transitions of $M$ are weighted by 0. The automaton $\mathcal{A}_{dist}$ weighted by $\textsc{Disc}_\lambda$ (with $\lambda \in (0,1)$) assigns the weight 0 to all traces of $M$. If $w$ is not a trace of $M$, $\mathcal{A}^{\textsc{Disc}_\lambda}_{dist}(w) = (1-\lambda)\sum_{i=k}^{\infty} \lambda^i = \lambda^k$, where $k$ is the length of the longest common prefix of $w$ and any trace of $M$. Observe that for a transition system $M'$, $d_M(M')$ is equal to $\lambda^K$, where $K$ is the maximal number such that every trace of $M'$ agree on the first $K$ letters with some trace of $M$.

We discuss constructions of automatic similarity measures in Section 4. Now, we assume that a weighted automaton $\mathcal{A}_{dist}$ computing $d_M$ is given and we show how to use it to compute $[P]_{d_M}$.

Consider the usual model-checking problem for LTL specifications: given a transition system $M$ and an LTL formula $\varphi$, decide whether $M \models \varphi$. An automata-based model-checking procedure constructs two $\omega$-automata: $\mathcal{A}_M$ accepting all traces of $M$, and $\mathcal{A}_{\neg\varphi}$ accepting all $\omega$-words that violate $\varphi$. $\omega$-words accepted by both, $\mathcal{A}_M$ and $\mathcal{A}_{\neg\varphi}$, are counterexamples to the statement $M \models \varphi$. Thus, the model-checking problem $M \models \varphi$ reduces to emptiness of $\mathcal{L}(\mathcal{A}_M \times \mathcal{A}_{\neg\varphi}) = \mathcal{L}(\mathcal{A}_M) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi})$. In order to compute the model-measure, we follow the same scheme.

Since the specification is already given by the automaton $\mathcal{A}_P$ recognizing its complement, we simply replace $\mathcal{A}_M$ with $\mathcal{A}_{dist}$ and compute the optimal-weight of the cross product $\mathcal{A}_{dist} \times \mathcal{A}_P$. This automaton is defined as the usual cross product of Büchi automata, but the weight of every transition is the weight of its first component in $\mathcal{A}_{dist}$. Observe that $\mathcal{A}_{dist} \times \mathcal{A}_P(w) = \mathcal{A}_{dist}(w)$ if $w \in \mathcal{L}(\mathcal{A}_P)$ and $\mathcal{A}_{dist} \times \mathcal{A}_P(w) = \infty$ otherwise. The optimal-weight of $\mathcal{A}_{dist} \times \mathcal{A}_P$ is precisely the value of $sr_{d_M}(M, P)$. Indeed, assume that $M \models P$ and consider, for every $\rho > 0$, an (infinite) transition systems $M_\rho$, such that the traces of $M_\rho$ are all $\omega$-words $w$ with $\mathcal{A}_{dist}(w) \leq \rho$. Clearly, $d_M(M_\rho) = \rho$. Observe that if an $\omega$-word $w$ has the weight $\rho$ assigned by $\mathcal{A}_{dist}$, i.e., $\mathcal{A}_{dist}(w) = \rho$, and $\mathcal{A}_P$ accepts $w$, then $M_\rho$ violates $P$ and $sr_{d_M}(M, P) \leq \rho$. Conversely, if $\mathcal{A}_P$ rejects all $\omega$-words $w$ with $\mathcal{A}_{dist}(w) < \rho$, then for every $\rho' < \rho$, $M_{\rho'} \models P$ and $sr_{d_M}(M, P) \geq \rho$. The case where $M \models \neg P$ is symmetric, and the case $M \not\models P$ and $M \not\models \neg P$ is trivial.

**Theorem 11.** *Assume that (linear-time) similarity measures and $\omega$-regular linear-time specifications are given by weighted Büchi automata with one of the weighting schemes* $\textsc{Sum}, \textsc{Max}, \textsc{Disc}_\lambda$ *or* $\textsc{LimAvg}$. *Then, the model-measure $[P]_{d_M}$ can be computed in polynomial time in the size of both automata representing $d_M$ and $P$.*

The size of $\mathcal{A}_{dist} \times \mathcal{A}_P$ is quadratic in $|\mathcal{A}_{dist}| + |\mathcal{A}_P|$. Since the optimal-weight question for $\textsc{Disc}_\lambda, \textsc{LimAvg}$ weighting schemes is equivalent to computing the value of the optimal strategy in a Markov decision process, and the latter is solved by linear programming, the optimal-weight questions for $\textsc{Disc}_\lambda$ and $\textsc{LimAvg}$ are solved in polynomial time assuming that arithmetical operations have constant costs. The question, whether linear programming, and in consequence the optimal-weight questions for $\textsc{Disc}_\lambda$ and $\textsc{LimAvg}$, admit polynomial-time algorithms when costs of arithmetic operations are proportional to lengths of their arguments, is still open.

### 3.2 Model measuring $\omega$-regular branching-time specifications

An *$\omega$-regular branching-time specification* $P$ is a subset of $\omega$-trees labeled by $\Sigma$, the set of all valid computation trees. We assume that $P$ is given by a Büchi-tree automaton $\mathcal{A}_P$ recognizing the set of all $\omega$-trees that violate $P$. The automaton $\mathcal{A}_P$ is an automaton over trees with varying (but bounded) degree. It can be regarded as a weighted automaton with all weights 0. Next, we proceed as in the linear-time case.

Surprisingly, the definition of similarity measure is simpler in the branching-time than in the linear-time case. Since every transition system $M'$ has the unique unrolling to an $\omega$-tree $t_{M'}$, the similarity measure of a transition system $M'$ is defined directly as the weight of its unrolling $t_{M'}$.

**Definition 12.** *A (branching-time) similarity measure $d_M$ is* automatic *iff there is a weighted $\omega$-tree automaton $\mathcal{A}_{dist}$ and a weighting scheme $f$ from Theorem 4 such that for every transition system $M'$, $d_M(M') = \mathcal{A}^f_{dist}(t_{M'})$.*

Again, by virtually the same argument as in the linear-time case, the optimal weight of $\mathcal{A}_{dist} \times \mathcal{A}_P$ is equal to $sr_{d_M}(M, P)$.

**Theorem 13.** *Assume that (branching-time) similarity measures and $\omega$-regular branching-time specifications are given by weighted Büchi-tree automata. Then, the model-measure $[P]_{d_M}$ can be computed in polynomial time in the size of both automata representing $d_M$ and $P$.*

*Remark 14.* Recall that we assume that weights are given in unary notation. It is an open problem whether mean-payoff and discounted-payoff games, and in consequence the optimal-weight question for SupDisc$_\lambda$ and SupLimAvg, admit polynomial-time algorithms if weights are given in binary notation.

*Remark 15.* Let $\varphi$ be a CTL formula. In order to compute the model-measure of $\varphi$, $\varphi$ has to be translated to a non-deterministic Büchi-tree automaton $\mathcal{A}_{\neg\varphi}$ recognizing all $\omega$-trees (of bounded degree) that violate it. Such an automaton has exponential size in $|\varphi|$. Thus, Theorem 13 yields an exponential-time algorithm computing the model-measure of a CTL formula, whereas CTL model checking has a linear-time algorithm. Unfortunately, the exponential blow-up cannot be avoided as the satisfiability problem for CTL, which is $EXPTIME$-complete, reduces to model-measuring for CTL (even with a fixed $d_M$). Consider a similarity measure $d_M$, which is finite for every transition system based on the full binary $\omega$-tree. Observe that $[\varphi]_{d_M} < 1$ iff $\neg\varphi$ is satisfiable over the class of models based on the full binary $\omega$-tree. The satisfiability problem for CTL, even restricted to models based on the full binary $\omega$-tree, is $EXPTIME$-complete.

*Remark 16.* Theorem 13 can be generalized to parity tree automata. The optimal-weight question can be solved for parity automata over the same weighting schemes as in the Büchi case. The complexity of those algorithms is higher, but it matches the complexity of solving parity games.

### 3.3 Undecidable model measuring

We have shown that the model-measure of a linear (or branching-time) specification can be computed for automatic similarity measures. It may seem to be a narrow class of similarity measures, but even slight extensions of this class make the model-measuring problem undecidable.

Let $\Sigma$ be an alphabet and let $S$ be a relation on $\Sigma \times \Sigma$ denoting *admissible* pairs of letters. Consider a function $f^S_M$ such that $f^S_M(w)$ is the minimal number of transpositions of admissible adjunct letters of $w$ necessary to transform $w$ to a trace of $M$. The function $f^S_M$ is a variant of sorting, where some letters cannot be swapped. Although $f^S_M$ cannot be computed by an automaton, as it requires unbounded memory, for every $\omega$-word $w$, $f^S_M(w)$ is computable in polynomial time.

**Theorem 17.** *There exist $M, S$ such that for the similarity measure defined as $d_M(M') = \sup\{f^S_M(u) : u \text{ is a trace of } M'\}$, the problem: given an LTL formula $\varphi$, decide whether $[\varphi]_{d_M} = 1$, is undecidable.*

## 4 Similarity Measures for $\omega$-regular Specifications

In this section we present a systematic approach to the construction of automatic similarity measures. They will be constructed from the transition system $M$ by relatively simple adjustments rather than modifications of $M$ itself. The system $M$ is usually complex, therefore modifying its internal structure is a complicated and error-prone task.

One way to construct similarity measures without modifying $M$ itself is to employ automatic weighted relations. Let $\mathcal{A}_M$ be an automaton that recognizes the set of traces of $M$ and let $R$ be an automatic weighted relation computed by $\mathcal{A}_R$. A similarity measure $d_M$, defined by $d_M(w) = \inf\{vRw : v \text{ is a trace of } M\}$ on $\omega$-words, and $d_M(M') = \sup\{d_M(w) : w \text{ is a trace of } M'\}$, is an automatic similarity measure. Indeed, consider a weighted automaton $\mathcal{A}_{dist}$ that, while running on the $\omega$-word $w$, guesses a trace $w$ of $M$ on the fly and computes $R$ by simulating $\mathcal{A}_R$. Since the weight of an $\omega$-word is the infimum over the weights of its runs, $\mathcal{A}_{dist}(w) = \inf\{vRw : v \text{ is a trace of } M\}$ and $\mathcal{A}_{dist}$ computes $d_M$.

Observe that $\mathcal{A}_{dist}$ can be constructed from automata $\mathcal{A}_M$ and $\mathcal{A}_R$ in a uniform way, i.e., independently of their internal structure. This is the main advantage of that approach, but this also makes it unsuitable. To see that, suppose that an automatic weighted relation $R^{E8}$ computes the similarity measure from Example 8. After the first packet is lost, the system (from Example 8) is in the state that is not reachable in a valid execution and a corrupt trace is not related to any valid trace of $M$. Thus, an automaton computing $R^{E8}$ would have to simulate $M$. In consequence, it would have to remember all states of $M$, which is precisely what we want to avoid.

We suggest a compromise between uniformity and expressiveness. In our approach the structure of $\mathcal{A}_M$ is unaffected, but its execution is governed by an external component, called the *hypervisor*.

**Definition 18.** *Let $\mathcal{A}_M = (\Sigma, Q_M, Q_{0,M}, \delta_M, F_M, C_M)$ be a weighted automaton. A* hypervisor $H$ *for $\mathcal{A}_M$ is a triple $(\mathcal{A}_H, \tau_H, \Gamma_H)$ such that*

- *$\mathcal{A}_H = (\Sigma, Q_H, Q_{0,H}, \delta_H, F_H, C_H)$ is a weighted automaton,*
- *$\tau_H : Q_H \mapsto 2^{Q_M \times \Sigma \times Q_M}$,*
- *$\Gamma_H : Q_H \mapsto \mathbb{N}^{Q_M \times \Sigma \times Q_M}$,*
- *$\mathcal{A}_H$ has an initial $q_I \in Q_{0,H}$, an* idle state*, such that $\tau_H[q_I] = \delta_M$, $\Gamma_H[q_I] = C_M$ and for every $a \in \Sigma$, $\mathcal{A}_H$ has a transition $(q_I, a, q_I)$ of weight 0.*

The functions $\tau_H, \Gamma_H$ determine the transition relation and cost function for $\mathcal{A}_M$ at each step. Intuitively, they should encode modifications applied to the transition relation and cost function of $\mathcal{A}_M$ rather than their complete descriptions. E.g. blind $a$-transitions $\tau_H[q_a] = \{(q, b, q') : (q, a, q') \in \delta_M, b \in \Sigma\}$, i.e., the automaton moves as it would read $a$, regardless of the actual letter. Having $\delta_M$, $\tau_H[q_a]$ can be simply defined regardless of complexity of $\delta_M$.

Let $\mathcal{A}_M = (\Sigma, Q_M, Q_{0,M}, \delta_M, F_M, C_M)$ be a weighted automaton. For a hypervisor $H = (\mathcal{A}_H, \tau_H, \Gamma_H)$ with $\mathcal{A}_H = (\Sigma, Q_H, Q_{0,H}, \delta_H, F_H, C_H)$, the *semi-direct product* $\mathcal{A}_M \ltimes H$ is a weighted generalized Büchi automaton $(\Sigma, Q_H \times Q_M, Q_{0,H} \times Q_{0,M}, \delta, C, \{F_1, F_2\})$ defined as follows:

- $\delta = \{(\langle q_1, q_2\rangle, a, \langle q_1', q_2'\rangle) : a \in \Sigma, (q_1, a, q_1') \in \delta_H, (q_2, a, q_2') \in \tau_H[q_1]\}$,
- $F_1 = F_H \times Q_M$ and $F_2 = Q_H \times F_M$, that is the automaton should visit infinitely often accepting states of $\mathcal{A}_M$ and those of $\mathcal{A}_H$,
- $C(\langle q_1, q_2\rangle, a, \langle q_1', q_2'\rangle) = C_H(q_1, a, q_1') + \Gamma_H[q_1](q_2, a, q_2')$.

Observe that for every hypervisor $H$, and every $\mathcal{A}_M$ recognizing the set of traces of $M$, $\mathcal{A}_M \ltimes H$ defines an automatic similarity measure related to $M$. Indeed, due to existence of the idle state, the automaton $\mathcal{A}_M \ltimes H$ can just simulate $\mathcal{A}_M$, therefore for every trace of $M$, $\mathcal{A}_M \ltimes H(w) = 0$. Conversely, the hypervisor method is complete, i.e., every automatic similarity measure $d_M$ is computed by an automaton which is the semi-product of $\mathcal{A}_M$ and some $H$.

Let $d_M$ be an automatic similarity measure and let $\mathcal{A}_{dist}$ be an automaton computing it. Consider a hypervisor $H = (\mathcal{A}_H, \tau_H, \Gamma_H)$ such that $\mathcal{A}_H$ can either begin in $q_I$ and stay there forever, or it can begin in $q_{0,dist}$, simulate the execution of $\mathcal{A}_{dist}$, and neglect the automaton $\mathcal{A}_M$, i.e., for every $q \in Q_H \setminus \{q_I\}$, $\tau_H[q]$ is the full relation and $\Gamma_H[q]$ is always 0. Then, for every $w$, $\mathcal{A}_M \ltimes H(w) = \mathcal{A}_{dist}(w)$, therefore $\mathcal{A}_M \ltimes H$ computes $d_M$.

However, this is a degenerate case. We rather focus on showing that the hypervisor-based approach is a convenient and reasonably uniform (w.r.t. $\mathcal{A}_M$) way of modeling similarity measures. In the following we shall give several examples supporting this thesis.

Observe that using aforementioned blind $a$-transitions one can simply define a similarity measure $d_M(w) = \inf\{vRw : v \text{ is a trace of } M\}$ based on an automatic weighted relation $R$. We leave that as an exercise for the reader.

Since the semi-direct product of a weighted automaton $\mathcal{A}_M$ and $H$ is again a weighted automaton, this construction can be iterated $((\mathcal{A}_M \ltimes H) \ltimes H')$. Although iteration can be avoided (Lemma 19), iterated definitions are often simpler (cf. Example 20).

**Lemma 19.** *Let $\mathcal{A}$ be a weighted automaton and $H_1, H_2$ be hypervisors for $\mathcal{A}$ and $\mathcal{A} \ltimes H_1$. There effectively exists a hypervisor $H_3$ such that for every $w$, $(\mathcal{A} \ltimes H_1) \ltimes H_2(w) = \mathcal{A} \ltimes H_3(w)$.*

*Example 20.* **(Edit distance)** We define the hypervisor $Del = (\mathcal{A}_{Del}, \tau_{Del}, \Gamma_{Del})$ computing deletions of letters. The automaton $\mathcal{A}_{Del}$ has two states, the idle state $q_I$, and the state $q_D$ responsible for deletion. In the deletion state $q_D$, the automaton $\mathcal{A}_M$ ignores the input letter and remains in its current state, i.e., $\tau_{Del}[q_D] = \{(q, a, q) : q \in Q, a \in \Sigma\}$. The cost functions $\Gamma_H[q_I], \Gamma_H[q_L]$ assign 0 weight to every transition of $\mathcal{A}_M$. Both, $q_I, q_D$, are initial states in $\mathcal{A}_H$ and $\delta_H$ is the full relation, i.e., $\delta_H = \{(q_1, a, q_2) : q_1, q_2 \in Q_H, a \in \Sigma\}$. Transitions from $q_I$ have weight 0 (in $\mathcal{A}_H$), whereas those from $q_D$ have weight 1. Clearly, $\mathcal{A}_M \ltimes Del$ computes a similarity measure such that the weight of an $\omega$-word $w$ is the least number of deletions necessary to transform $w$ to a trace of $M$. $\mathcal{A}_M \ltimes Del$ extends from $\omega$-words to transition systems by taking supremum over all traces of a transition system.

Similarly, one can define hypervisors $Ins, Sub, Tra$ computing insertions, a single letter substitutions or transpositions of adjacent letters necessary to transform a given $\omega$-word to an $\omega$-word accepted by the hypervised automaton. Then, the automaton $\mathcal{A}_{edit}$, defined as $(((M \ltimes Del) \ltimes Sub) \ltimes Tra) \ltimes Ins$, computes the edit distance between $w$ and the set of traces of $M$. Indeed, if $v$, a trace of $M$, can be obtained from $w$ by applying deletions, substitution, transpositions and insertions, it can be obtained by applying the same number of these operations in precisely that order, i.e., first deletions, next substitutions etc.

*Example 21.* **(An unreliable channel)** Consider the similarity measure $d_M$ from Example 8. Assume that those parties, $P_1, P_2$, communicate through a sheared variable $a$. Suppose that the transition relation for $M$ is given symbolically by a propositional formula $\mathcal{N}(\boldsymbol{p_1 p_2 a}, \boldsymbol{p'_1 p'_2 a'})$, where $\boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{a}$ are vectors of propositional variables that represent the current state of $P_1, P_2$ and $a$, and $\boldsymbol{p'_1}, \boldsymbol{p'_2}, \boldsymbol{a'}$ represent their next state. All transitions in $M$ have weight 0.

Consider a hypervisor $H = (\mathcal{A}_H, \tau_H, \Gamma_H)$ such that $\mathcal{A}_H$ has two states: the idle state $q_I$, and $q_L$, the state of a packet being lost. The cost functions $\Gamma_H[q_I], \Gamma_H[q_L]$ assign 0 weight to every transition, which can be easily expressed symbolically. Then $\tau[q_I], \tau[q_L]$ are represented by $\mathcal{N}$, and $\mathcal{N}_L(\boldsymbol{p_1 p_2 a}, \boldsymbol{p'_1 p'_2 a'}) \equiv \exists \boldsymbol{a''}(\mathcal{N}(\boldsymbol{p_1 p_2 a}, \boldsymbol{p'_1 p'_2 a''}) \wedge \boldsymbol{a} = \boldsymbol{a'})$. Thus, when $\mathcal{A}_H$ is in the state $q_D$, $\mathcal{A}_M$ executes the usual transition, but immediately after that $a$ is being reset to its previous value. Clearly, $\mathcal{A}_M \ltimes H$ defines the similarity measure from Example 8.

Now, by employing different weighting schemes, we can ask a whole range of questions. For Sum weighting scheme, the stability radius is the maximal number of lost packets tolerated by the system, whereas for LimAvg weighting scheme it gives the maximal *average ratio* of lost packets tolerated by the system.

*Example 22.* **(Active environment)** We can extend the idea from Example 21 to many processes where content of packets may be altered during communication. It is possible, as in the Dolev-Yao model for verification of cryptographic protocols [12], to simulate a scenario where all communication channels are controlled by the *intruder* who can intercept and forge packets. As it is unlikely that the system is immune to arbitrary actions of the intruder, the model-measure tells us *how* vulnerable the system is. E.g. the system works correctly as long as no more than 5 packets are forged.

The hypervisor approach can be straightforwardly adapted to the branching-time case. A (tree) hypervisor $H$ is a triple $(\mathcal{A}_H, \tau_H, \Gamma_H)$ such that $\mathcal{A}_H$ is a weighted automaton over $\omega$-trees with varying (but bounded) degree and $\tau_H, \Gamma_H$ associate with each state of $\mathcal{A}_H$ a (tree) transition relation and cost function.

Now, we shall present examples of branching-time similarity measures. The first example is a class of measures inherited from the linear-time case. In the linear-time case, $d_M(M')$ is defined as the supremum over weights of all traces of $M'$, therefore linear-time similarity measures naturally translate to branching-time similarity measures over Sup weighting schemes. Indeed, consider an

$\omega$-word automaton $\mathcal{A}^w_M$. By extending the labeling of $M$ to $\Sigma \times Q$, we can assume that $\mathcal{A}^w_M$ is deterministic. Then, it can be transformed to an $\omega$-tree automaton $\mathcal{A}^t_M$ that accepts precisely those $\omega$-trees whose paths are accepted by $\mathcal{A}^w_M$. This idea can be generalized to weighted tree automata over SUP weighting schemes to get the following:

**Proposition 23.** *Let $M$ be a transition system and let $\mathcal{A}^w_M, \mathcal{A}^t_M$ be a word and tree automata representing $M$. Every word hypervisor $H^w$ can be translated to a tree hypervisor $H^t$ such that the similarity measures defined by $\mathcal{A}^w_M \ltimes H^w$ and by $\mathcal{A}^t_M \ltimes H^t$ agree on every transition system $M'$ labeled by $\Sigma \times Q^w_H \times Q^w_M$.*

In particular, Examples 20, 21, 22 can be adapted to the branching-time case.

Another feature of tree hypervisors, which is incompatible with the linear-time case, is the ability to clone (or prune) a transition. Transition cloning at a state can be easily implemented as follows. For every $j \in \{1, \ldots, k\}$, the hypervisor has a cloning state $q_{c,j}$ such that $\tau_H[q_{c,j}]$ changes $q_0 \xrightarrow{a} \langle q_1, \ldots, q_k \rangle$, an original transition of $\mathcal{A}_M$, to $q_0 \xrightarrow{a} \langle q_1, \ldots, q_k, q_j \rangle$. In a similar way one can define transition pruning. By combining cloning and pruning one can implement the robustness notion from [14]. Indeed, the language of all execution trees of $M \parallel M'$, where branching degree of $M'$ is bounded by $B$, can be obtained by the combination of transition cloning (where each transition is cloned at most $B$ times), and arbitrary pruning. Thus, robustness of open systems defined in [14] is a special case of model measuring.

*Example 24.* **(Mutations)** Removal of behaviors according to [17] is a special case of our transition pruning. Generally, mutations that modify or add behaviors can be straightforwardly implemented using the hypervisor approach. Thus, all mutations considered in [17] can be expressed by similarity measures.

Finally, the model-measuring problem subsumes the robustness distance [3]:

**Proposition 25.** *Let $M$ be a transition system. There (effectively) exists a similarity measure $d_M$ such that for every transition system $M'$, the value of the robustness distance from $M$ to $M'$ equals to $1 - [M']_{d_M}$.*

## 5 Conclusions

We have defined the model-measuring problem, which generalizes several previously studied notions of robustness in verification. We have shown a way to express several distances (edit distance; semantic distance: the number of lost packets; etc.) in a convenient way, based on weighted automata, which admits a succinct symbolic representation.

The algorithms computing the model measure follow the same basic scheme as standard automata-based model-checking algorithms. This suggests that our method can be implemented on the basis of existing model-checking tools.

The model-measuring problem can be extended to the real-time case. It remains to construct a variety of similarity measures in the timed case.

## References


1. Franz Baader and Rafael Peñaloza. Automata-based axiom pinpointing. *J. Autom. Reasoning*, 45(2):91–129, 2010.
2. Udi Boker, Krishnendu Chatterjee, Thomas A. Henzinger, and Orna Kupferman. Temporal specifications with accumulative values. In *LICS*, pages 43–52. IEEE Computer Society, 2011.
3. Pavol Cerný, Thomas A. Henzinger, and Arjun Radhakrishna. Simulation distances. *Theor. Comput. Sci.*, 413(1):21–35, 2012.
4. Krishnendu Chatterjee and Laurent Doyen. Energy parity games. In *ICALP (2)*, volume 6199 of *LNCS*, pages 599–610. Springer, 2010.
5. Krishnendu Chatterjee, Laurent Doyen, Herbert Edelsbrunner, Thomas A. Henzinger, and Philippe Rannou. Mean-payoff automaton expressions. *CoRR*, abs/1006.1492, 2010.
6. Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. In *CSL*, pages 385–400, 2008.
7. Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Alternating weighted automata. In *FCT'09*, pages 3–13. Springer-Verlag, 2009.

8. Krishnendu Chatterjee, Thomas A. Henzinger, Barbara Jobstmann, and Rohit Singh. Measuring and synthesizing systems in probabilistic environments. In *CAV*, volume 6174 of *LNCS*, pages 380–395. Springer, 2010.
9. Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdzinski. Mean-payoff parity games. In *LICS*, pages 178–187. IEEE Computer Society, 2005.
10. Hana Chockler, Orna Kupferman, and Moshe Y. Vardi. Coverage metrics for temporal logic model checking. In Tiziana Margaria and Wang Yi, editors, *TACAS*, volume 2031 of *LNCS*, pages 528–542. Springer, 2001.
11. Sayantan Das, Ansuman Banerjee, Prasenjit Basu, Pallab Dasgupta, P. P. Chakrabarti, Chunduri Rama Mohan, and Limor Fix. Formal methods for analyzing the completeness of an assertion suite against a high-level fault model. In *VLSI Design*, pages 201–206. IEEE Computer Society, 2005.
12. D. Dolev and A. C. Yao. On the security of public key protocols. In *FOCS '81*, pages 350–357, Washington, DC, USA, 1981. IEEE Computer Society.
13. Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer-Verlag New York, Inc., New York, USA, 1996.
14. Orna Grumberg and David E. Long. Model checking and modular verification. In *CONCUR*, volume 527 of *LNCS*, pages 250–265. Springer, 1991.
15. Thomas A. Henzinger. From boolean to quantitative notions of correctness. In *POPL '10*, pages 157–158, New York, USA, 2010. ACM.
16. D. Hinrichsen and N. K. Son. Stability radii of linear discrete-time systems and symplectic pencils. *Int. J. of Robust and Nonlinear Control*, 1(2):79–97, 1991.
17. Orna Kupferman, Wenchao Li, and Sanjit A. Seshia. A theory of mutations with applications to vacuity, coverage, and fault tolerance. In *FMCAD*, pages 1–9. IEEE, 2008.
18. Orna Kupferman and Moshe Y. Vardi. Robust satisfaction. In *CONCUR*, volume 1664 of *LNCS*, pages 383–398. Springer, 1999.
19. Orna Kupferman and Moshe Y. Vardi. Vacuity detection in temporal model checking. *STTT*, 4(2):224–233, 2003.
20. Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, March 2000.
21. Hing Leung. Limitedness theorem on finite automata with distance functions: an algebraic proof. *Theor. Comput. Sci.*, 81(1):137 – 145, 1991.
22. Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.
23. Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158(1&2):343–359, 1996.

## A The proof of Theorem 3

*Proof.* SUM: The automaton $\mathcal{A}$ has a fixed set of weights, thus it has the least non-zero weight $\rho$. It follows that every run of $\mathcal{A}$ on an $\omega$-word that has finite weight contains only finitely many non-zero weighted transitions. Thus, every run of finite weight consists of a finite (possibly empty) prefix of non-zero weight and infinite suffix of weight 0.

By transformation of $\mathcal{A}$ we can compute the set of states $Q^{\text{inf}}$ that admit infinite run of weight 0; it is sufficient to remove transitions of nonzero weights. Next, consider a finite automaton $\mathcal{A}_2$ derived from $\mathcal{A}$ by replacing $\omega$-accepting condition with the set of accepting states $F = Q^{\text{inf}}$. Thus, the least weight of a word accepted by $\mathcal{A}_2$ is equal the least weight of any accepting run of $\mathcal{A}$. The automata $\mathcal{A}_2$ and subsequently $\mathcal{A}_1$ can be constructed in $O(n)$.

We can assume that there is a single initial state and a single final state. If there are many initial states, we can create additional initial state $s_I$ and connect it by $\epsilon$-transitions of weight 0 to all old initial states. We can do the same with final states. Then, the least weight of any accepting run of a finite automaton $\mathcal{A}_2$ can by found by applying Dijkstra's algorithm (working in $O(n \log n)$) to a weighted graph $(Q, E, C)$ resulting from $\mathcal{A}_2$ by connecting states that are reachable by direct transition and assign to edges the least (over all letters) transition weight.

MAX: As in the case of SUM use Dijkstra algorithm, but use *max* function instead of + for updating the distances.

DISC$_\lambda$: Consider a Markov Decision Process(MDP) $M^\lambda$ obtained from $\mathcal{A}$ by selecting all states that admit accepting run and removing labels corresponding to letters. Let $\pi$ be a path of minimal weight in $M^\lambda$. Observe that the infimum over the weights of words accepted by $\mathcal{A}$ is equal the weight of $\pi$. Indeed, an accepting run of $\mathcal{A}$ contains only states from $M^\lambda$, thus the least weight of an accepting run of $\mathcal{A}$ does not exceed the weight of $\pi$. On the other hand, for every $\epsilon > 0$, there is a position an accepting run of $\mathcal{A}$ whose weight differs from the weight of $\pi$ by $\epsilon$. It suffice to take

sufficiently long prefix $\pi[N]$ of $\pi$ which is concatenated with an accepting run of $\mathcal{A}$ starting in the final state of $\pi[N]$.

LimAvg: We show the result for the parity objective. Consider a family of MDPs $M^{2p,C}$ obtained by selecting all states with priority not exceeding $2p$ such that selecting in the result a maximal strongly connected component $C$. Let $\pi^{2p,C}$ be a path of the least weight in $M^{2p,C}$. Then, every accepting run of $\mathcal{A}$ is finally contained in one of $M^{2p,C}$, for some $2p$ and $C$. As the weight of the run does not depend on finite prefix, the optimal weight of $\mathcal{A}$ does not exceed the minimum over $2p, C$ of the least weights of $M^{2p,C}$. On the other hand, every path of the least weight in $M^{2p,C}$ can be altered in such a way that the priority $2p$ occurs infinitely often. The original path can be altered seldom enough that limit average is unaffected. Thus, the infimum over weights of all words accepted by $\mathcal{A}$ equal minimum over $2p, C$ of the least weights of $M^{2p,C}$.

## B The proof of Theorem 4

*Proof.* SupSum and AccSum: Again, as in the word case, every run $\pi$ of finite weight can be partitioned into a finite tree of finite weight and finitely many trees of weight zero. As in the word problem, first compute the set of states $Q_0$, such that $q \in Q_0$ if there is an accepting run of weight zero whose root is labeled by $q$. Next, use dynamic programming to compute a finite run of minimal weight whose leaves are labeled by states from $Q_0$. The solution to the optimal weight problem for SupSum has been also presented in

SupLimAvg: The problem is equivalent to mean payoff parity games [9]. It has been shown that the parity and the mean payoff are orthogonal. In particular, when a parity condition is replaced by a Büchi condition, the algorithm works in time proportional to solving mean-payoff games, which can be solved in polynomial time assuming that all weights are bounded by a constant [23].

AccDisc$_\lambda$ and SupDisc$_\lambda$: First, select a set of all states of the automata for which there is an accepting run according to the Büchi condition. Denote this set by $Q_w$.

For SupDisc$_\lambda$, construct a Markov Decision Process $\mathcal{P}$ of the states from $Q_w$. Observe that the optimal value of $\mathcal{P}$ is equal to the optimal value of the automata. Indeed, for every $\epsilon > 0$, we can construct an accepting run that differs from the optimal run of $\mathcal{P}$ by at most $\epsilon$. Basically, we construct a run play on $\mathcal{P}$ sufficiently long, and when the discount factor makes the contribution of the further play smaller than $\epsilon$, we play to satisfy the Büchi condition.

For SupDisc$_\lambda$, construct a discount-payoff game ([23]) on $Q_w$ and transitions from $Q_w$ into states of $Q_w$. The minimizer plays on $Q_w$; he picks a letter $a$ and a weighted transition $q \xrightarrow{a} (q_1, \ldots, q_k)$. Next, the maximizer picks one of the states $q_1, \ldots, q_k$ in the transition and that state is the next position of the minimizer. Observe that, for every $\epsilon > 0$, the minimizer can play sufficiently long according to the optimal strategy, so that the value of the game is $\epsilon$-close to the optimal value. After that he can play to satisfy the Büchi condition.

AccLimAvg: The problem is equivalent to determining the optimal value of an MDP. For a tree automaton $\mathcal{A} = (\Sigma, Q, q_o, \delta, F)$, consider an MDP whose states are $Q \cup \delta$. The states in $Q$ are *player* states and the states in $\delta$ are *random states* (cf. [8]). Basically, in a state $q$ the player can choose an appropriate transition $q \xrightarrow{a} (q_1, \ldots, q_k)$ available in $q$. Then, the random player in $q \xrightarrow{a} (q_1, \ldots, q_k)$ mimics branching by choosing every successor state $q_1, \ldots, q_k$ with equal probability $\frac{1}{k}$. Every play in this MDP corresponds to a run of $\mathcal{A}$ on some tree. Observe that the mean-payoff value of the play defined as in [8] coincides with the value of the run according to AccLimAvg. Solving MDP with mean-payoff and parity objective has been discussed in [8]. As in the case of mean-payoff parity games, it has been shown that the parity and the mean payoff are orthogonal. In particular, when a parity condition is replaced by a Büchi condition, the algorithm works in polynomial time.

## C The proof of Theorem 17

*Proof.* Let $k > 0$, let $\Sigma = (\{a, b, 1, \ldots, k\} \times \{+, -\}) \cup \{\#\}$[2]. The letters $a^+, 2^-$ denote $(a, +), (2, -)$. For a word $w$ over $\{a, b, 1, \ldots, k\}$, the words $w^-, w^+$ are defined by replacing every letter $\alpha \in \{a, b, 1 \ldots, k\}$, by $\alpha^-$ or $\alpha^+$ respectively. For $w \in \Sigma^*$, define $\pi_1(w), \ldots, \pi_4(w)$ as projections of $w$ on the alphabets $\{a^-, b^-\}, \{a^+, b^+\}, \{1^-, \ldots, k^-\}$ and $\{1^+, \ldots, k^+\}$.

[2] The letter # is used only to maintain that all words are infinite.

Consider a model $M$ whose traces are finite concatenations of the following words: $a^-a^+$, $b^-b^+$, $1^-1^+, \ldots, k^-k^+$, followed by infinitely many # symbols. E.g. $2^-2^+b^-b^+1^-1^+\#^\omega$. Thus, if $w$ is a trace of $M$, then there are words $v, u$ over $\{a, b\}$ and $\{1, \ldots, k\}$ such that

$$\pi_1(w) = v^- \qquad \pi_2(w) = v^+ \tag{1}$$

$$\pi_3(w) = u^- \qquad \pi_4(w) = u^+ \tag{2}$$

Define a function $f^S_M$ on words over $\Sigma$ in the following way: if $w\#^\omega$ is a trace of $M$, then $f^S_M(w) = 0$, otherwise $f^S_M(w)$ the least number of allowed transpositions of adjacent letters that applied to $w$ that transforms $w$ to $u$ such that $u\#^\omega$ is a trace from $M$. A transposition of letter $\alpha, \beta$ is allowed, $(\alpha, \beta) \in S$, if $\alpha$ and $\beta$ have different polarities, or they have the same polarity $p$ but one belongs to $\{a^p, b^p\}$ and the other belongs to $\{1^p, \ldots, k^p\}$. Define $d_M(M')$ as $d_M(M') = \sup\{f(w) : w\#^\omega \text{ is a trace of } M'\}$. Then, $d_M(M') = k$ if every trace of $M'$ can be obtained from a trace of $M$ by at most $k$ allowed transpositions. In particular, $d_M(M) = 0$.

Observe that the projections of an $\omega$-word over $\Sigma$ on each of the following four sets $\{a^-, b^-\}, \{a^+, b^+\}, \{1^-, \ldots, k^-\}$ and $\{1^-, \ldots, k^-\}$ are invariant under the allowed transpositions. Thus, $d_M(w)$ is finite iff there is $w' \in (\Sigma \setminus \{\#\})^*$, such that $w = w'\#^\omega$, and there are words $v, u$ over $\{a, b\}$ and $\{1, \ldots, k\}$ such that $\pi_1(w') = v^-$, $\pi_2(w') = v^+$, and $\pi_3(w') = u^-$, $\pi_4(w') = u^+$.

It can be easily decided what is a weight of a given word. On the other hand, we show an LTL formula $\varphi_\mathcal{P}$ such that $[\varphi_\mathcal{P}] < 1$ if and only if the PCP instance $\varphi_\mathcal{P}$ has a solution.

Let $\Gamma = \{(w_1, v_1), \ldots, (w_k, v_k)\}$ be an instance of the Post Correspondence Problem. The automaton $\mathcal{A}_{P_\Gamma}$ for the specification $P_\Gamma$ accepts all $\omega$-words that are finite concatenations of words $i^-w_i^-$ or $i^+v_i^+$, for $i \in \{1, \ldots, k\}$, followed by infinite number of # letters. In other words, $\mathcal{A}_{P_\Gamma}$ accepts $w$ iff

$$\pi_1(w) = w^-_{i_1} \ldots w^-_{i_m} \qquad \text{where } \pi_3(w) = i^-_1 \ldots i^-_m \tag{3}$$

$$\pi_2(w) = v^+_{i_1} \ldots v^+_{i_p} \qquad \text{where } \pi_4(w) = i^+_1 \ldots i^+_p \tag{4}$$

Now, by combining Equations (1), (2), (3) and (4) we have $[P_\Gamma]_M < \infty$ if and only if $\Gamma$ has a solution.

The PCP problem is undecidable even if $k$ is fixed, therefore the problem given $\varphi$, decide whether $[\varphi_\mathcal{P}] < 1$, is undecidable even for a fixed similarity measure.

## D The proof of Proposition 23

*Proof.* Recall that the unrolling of the transition system $M'$ is the tree $(\tau, L)$ such that $\tau$ consists of all finite labeled paths through $M'$. Since we consider transition systems labeled by $\Sigma \times Q^w_M \times Q^w_H$, every path of $M'$ uniquely determines the run of $\mathcal{A}^w_M \ltimes H^w$. Thus, we can consider that automaton to be deterministic.

A deterministic word automaton $\mathcal{A}_w$ can be straightforwardly transformed to a tree automaton $\mathcal{A}_t$ that accepts trees whose all paths are accepted by the word automaton. Indeed, the set of states of $\mathcal{A}_t$ is $(Q_w \cup \{\bot\}) \times Q_w \times (\Sigma \cup \{\epsilon\})$ and the initial is $(\bot, q_0, \epsilon)$. Then, for every state $q$ of a word automaton, if $q \xrightarrow{\mathrm{a_1}} q_1, \ldots, q \xrightarrow{\mathrm{a_k}} q_k$ are all transitions in the state $q$, then for all $x \in \Sigma, q' \in Q_w$, $(q', q, x) \xrightarrow{\mathrm{x}} \langle (q, q_1, a_1), \ldots, (q, q_k, a_k)\rangle$ in a transition of $\mathcal{A}_t$. The automaton $\mathcal{A}_t$ remembers in its state the letter of the previous transition (which should label the current node) and the previous state. The weight of the transition $(q', q, x) \xrightarrow{\mathrm{x}} \langle (q, q_1, a_1), \ldots, (q, q_k, a_k)\rangle$, is the weight of the transition $q' \xrightarrow{\mathrm{x}} q$ in $\mathcal{A}^w$.

Observe that sequence of weights alone every path $\pi$ is the same as the sequence of weights of the run ($\mathcal{A}_w$ is deterministic) of $\mathcal{A}_w$ on the word $\pi$. Thus, the result follows.

## E The proof of Proposition 25

*Proof.* Consider two transitions systems $M_1, M_2$, a specification system and a correct implementation. The robustness distance [3] has been defined as the value in the following game.

There are two players, the specification and the implementation, playing a game on their systems $M_1, M_2$. The objective of the specification is to simulate moves of the implementation. At every step, first the specification either allows the implementation to cheat, i.e., to take a transition

that is not present $M_2$, or prohibits that. Then, the implementation chooses a transition and the specification tries to find a corresponding transition in $M_1$. The robustness of system is determined by value, how often the specification may allow the implementation to cheat and still win the game.

We define a similarity measure $d_{M_2}$ is over binary trees that represent strategies for the implementation in the following way. Let $t$ be a tree. Every node in $t$ has degree 1 or 2. If a node has degree 1, the implementation is forced to play faithfully. Otherwise, the implementation has a choice, it can play faithfully or cheat. The weight of each node in a tree is 0, if its degree is 1, and 1 otherwise. Then, the weight of the whole tree is computed according to $\textsc{SupDisc}_\lambda$ or $\textsc{SupLimAvg}$ weighting scheme. The specification $\varphi$ of the model measuring problem is a CTL formula that states that no path of a tree is a trace of $M_1$, i.e., the implementation has the winning strategy.

The tree $t$ of weight $k$ that satisfies $\varphi$ certifies that regardless of allowing or disallowing cheating by the specification, the implementation wins the game. In consequence, the stability radius of $\varphi$ corresponds to the value in the robustness game.