Theoretical Computer Science 413 (2012) 21–35

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Simulation distances☆,☆☆

Pavol Černý, Thomas A. Henzinger, Arjun Radhakrishna *

IST Austria, Klosterneuburg, A-3400, Austria

ARTICLE INFO

Keywords:
Simulation
Quantitative analysis
Games

ABSTRACT

Boolean notions of correctness are formalized by preorders on systems. Quantitative measures of correctness can be formalized by real-valued distance functions between systems, where the distance between implementation and specification provides a measure of "fit" or "desirability". We extend the simulation preorder to the quantitative setting by making each player of a simulation game pay a certain price for her choices. We use the resulting games with quantitative objectives to define three different simulation distances. The *correctness distance* measures how much the specification must be changed in order to be satisfied by the implementation. The *coverage distance* measures how much the implementation restricts the degrees of freedom offered by the specification. The *robustness distance* measures how much a system can deviate from the implementation description without violating the specification. We consider these distances for safety as well as liveness specifications. The distances can be computed in polynomial time for safety specifications, and for liveness specifications given by weak fairness constraints. We show that the distance functions satisfy the triangle inequality, that the distance between two systems does not increase under parallel composition with a third system, and that the distance between two systems can be bounded from above and below by distances between abstractions of the two systems. These properties suggest that our simulation distances provide an appropriate basis for a quantitative theory of discrete systems. We also demonstrate how the robustness distance can be used to measure how many transmission errors are tolerated by error correcting codes.

1. Introduction

Standard verification systems return a boolean answer that indicates whether a system satisfies its specification. However, not all correct implementations are equally good, and not all incorrect implementations are equally bad. There is thus a natural question whether it is possible to extend the standard specification frameworks and verification algorithms to capture a finer and more quantitative view of the relationship between specifications and systems.

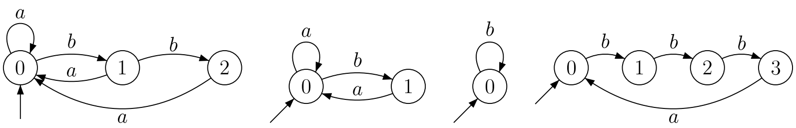
We focus on extending the notion of simulation to the quantitative setting. For reactive systems, the standard correctness requirement is that all executions of an implementation have to be allowed by the specification. Requiring that the specification simulates the implementation is a stricter condition, but it is computationally less expensive to check. The simulation relation defines a preorder on systems. We extend the simulation preorder to a distance function that, given two systems, returns a real-valued distance between them.

☆ This work was partially supported by the ERC Advanced Grant QUAREM, the FWF NFN Grant S11402-N23 (RiSE), the European Union project COMBEST and the European Network of Excellence Artist Design.

☆☆ Preliminary version of this work appeared in Černý et al. (2010) [5].

* Corresponding author. Tel.: +43 224390003503.
E-mail addresses: cernyp@ist.ac.at (P. Černý), tah@ist.ac.at (T.A. Henzinger), aradha@ist.ac.at (A. Radhakrishna).

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.08.002



(a) S_1. (b) I_2. (c) I_3. (d) I_4.

Fig. 1. Example systems.

Let us consider the definition of simulation of an implementation I by a specification S as a two-player game, where Player 1 (the implementation) chooses moves (transitions) and Player 2 (the specification) tries to match each move. The goal of Player 1 is to prove that simulation does not hold, by driving the game into a state from which Player 2 cannot match the chosen move; the goal of Player 2 is to prove that there exists a simulation relation, by playing the game forever. In order to extend this definition to capture how "good" (or how "bad") the simulation is, we make the players pay a certain price for their choices. The goal of Player 1 is then to maximize the cost of the game, and the goal of Player 2 is to minimize it. The cost is given by an objective function, such as the limit average of transition prices. For example, for incorrect implementations, i.e., those for which the specification S does not simulate the implementation I, we might be interested in how often the specification (Player 2) cannot match an implementation move. We formalize this using a game with a limit-average objective between modified systems. The specification is allowed to "cheat", by following a non-existing transition, while the implementation is left unmodified. More precisely, the specification is modified by giving the transitions, from the original system, a weight of 0, and adding new "cheating" transitions with a non-zero positive weight. As Player 2 is trying to minimize the value of the game, she is motivated not to cheat. The value of the game measures how often the specification can be forced to cheat by the implementation, that is, how often the implementation violates the specification (i.e., commits an error) in the worst case. We call this distance function *correctness*.

Let us consider the examples in Fig. 1. We take the system S_1 as the specification. The specification allows at most two symbols b to be output in the row. Now let us consider the two incorrect implementations I_3 and I_4. The implementation I_3 outputs an unbounded number of b's in a row, while the implementation I_4 can output three b's in a row. The specification S_1, thus, will not be able to simulate either I_3 or I_4, but I_4 is a "better" implementation in the sense that it violates the requirement to a smaller degree. We capture this by allowing S_1 to cheat in the simulation game by taking an existing edge while outputting a different symbol. When simulating the system I_3, the specification S_1 will have to output a b when taking the edge from state 2 to state 0. This cheating transition will be taken every third move while simulating I_3. The correctness distance from S_1 to I_3 will therefore be $1/3$. When simulating I_4, the specification S_1 needs to cheat only one in four times—this is when I_4 takes a transition from its state 2 to state 3. The distance from S_1 to I_4 will be $1/4$.

Considering the implementation I_2 from Fig. 1, it is easy to see that it is correct with respect to the specification S_1. The correctness distance would thus be 0. However, it is also easy to see that I_2 does not include all behaviors allowed by S_1. Our second distance function, *coverage*, is the dual of the correctness distance. It measures how many of the behaviors allowed by the specification are actually implemented by the implementation. This distance is obtained as the value for the implementation in a game in which I is required to simulate S, with the implementation being allowed to cheat. Our third distance function is called *robustness*. It measures how robust the implementation I is with respect to the specification S in the following sense: we measure how often the implementation can make an unexpected error (i.e., it performs a transition not present in its transition relation), with the resulting behavior still being accepted by the specification. Unexpected errors could be caused, for example, by a hardware problem, by a wrong environment assumption, or by a malicious attack. Robustness measures how many such unexpected errors are tolerated.

In addition to safety specifications, we consider liveness specifications given by weak (Büchi) fairness constraints or strong (Streett) fairness constraints. In order to define distances to liveness specifications, the notion of quantitative simulation is extended to *fair* quantitative simulation. We study variations of the correctness, coverage, and robustness distances using limit-average and discounted objective functions. Limit-average objectives measure the long-run frequency of errors, whereas discounted objectives count the number of errors and give more weight to earlier errors than later ones.

The correctness, coverage, and robustness distances can be calculated by solving the value problem in the corresponding games. Without fairness requirements, we obtain limit-average games or discounted games with constant weights. The values of such games can be computed in polynomial time [22]. We obtain polynomial complexity also for distances between systems with weak-fairness constraints, whereas for strong-fairness constraints, the best known algorithms require exponential time.

We present composition and abstraction techniques that are useful for computing and approximating simulation distances between large systems. We prove that distance from a composite implementation $I_1 \parallel I_2$ to a composite specification $S_1 \parallel S_2$ is bounded by the sum of distances from I_1 to S_1 and from I_2 to S_2. Furthermore, we show that the distance between two systems can be bounded from above and below by distances between abstractions of the two systems.

Finally, we present an application of the robustness distance as well as an application of the coverage distance. First, we consider error correction systems for transmitting data over noisy channels and show that the robustness distance measures how many transmission errors can be tolerated by an implementation. Three implementations are analyzed, one based on the Hamming code, one based on triple modular redundancy, and an implementation without any error correction. Second, a specification of a reactive system with inputs and outputs is considered, and we use the coverage metric to determine what part of the input words, for which a specification defines an output, is covered by different implementations.

Related work. Weighted automata [2,12] provide a way to assign values to words, and to languages defined by finite-state systems. Distances between systems can be defined using weighted automata, analogically to boolean language inclusion. However, the complexity of computation of such distance is not known [6]. Our solution of using a quantitative version of simulation games corresponds, in the boolean case, to the choice of using simulation instead of language inclusion. There have been several attempts to give a mathematical semantics to reactive processes which is based on quantitative metrics rather than boolean preorders [20,8]. In particular, for probabilistic processes, it is natural to generalize bisimulation relations to bisimulation metrics [11,21], and similar generalizations can be pursued if quantities enter not through probabilities but through discounting [9] or continuous variables [4] (this work uses the Skorohod metric on continuous behaviors to measure the distance between hybrid systems). We consider distances between purely discrete (nonprobabilistic, untimed) systems, and our distances are directed rather than symmetric (based on simulation rather than bisimulation). Software metrics measure properties such as lines of code, depth of inheritance (in an object-oriented language), number of bugs in a module or the time it took to discover the bugs (see for example [13,17]). These functions measure syntactic properties of the source code, and are fundamentally different from our distances that capture the difference in the behavior (semantics) of programs.

2. Quantitative simulation games

Transition systems. A *transition system* is a tuple $\langle S, \Sigma, E, s_0\rangle$ where S is a finite set of states, Σ is a finite alphabet, $E \subseteq S \times \Sigma \times S$ is a set of labeled transitions, and s_0 is the initial state. We require that for every $s \in S$, there exists a transition from s. The set of all transition systems is denoted by $\mathcal{S}$. A *weighted transition system* is a transition system along with a weight function v from E to $\mathbb{Q}$. A run in a transition system T is an infinite path $\rho = \rho_0\sigma_0\rho_1\sigma_1\rho_2\sigma_2 \ldots \in (S \cdot \Sigma)^\omega$ where $\rho_0 = s_0$ and for all i, $(\rho_i, \sigma_i, \rho_{i+1}) \in E$.

Fairness conditions. A *Büchi (weak fairness) condition* for a (weighted) transition system is set of states $F \subseteq S$. Given a Büchi condition F and a run $\rho = \rho_0\sigma_0\rho_1\sigma_1 \ldots$ of a transition system, the run ρ is *fair* iff $\forall n \geq 0 : (\exists i > n : \rho_i \in F)$. A *Streett (strong fairness) condition* for a (weighted) transition system is a set of request-response pairs $F = \{\langle E_1, F_1\rangle, \langle E_2, F_2\rangle, \ldots, \langle E_d, F_d\rangle\}$ where each $E_i, F_i \in 2^S$. Given a Streett condition, a run $\rho = \rho_0\sigma_0\rho_1\sigma_1 \ldots$ is *fair* iff $\forall k \leq d : \big((|\{i \mid \rho_i \in E_k\}| = \infty) \Rightarrow (|\{i \mid \rho_i \in F_k\}| = \infty)\big)$. We denote a transition system A with a fairness condition F as A^F.

Game graphs. A *game graph* G is a tuple $\langle S, S_1, S_2, \Sigma, E, s_0\rangle$ where S, Σ, E and s_0 are as in transition systems and (S_1, S_2) is a partition of S. The choice of the next state is made by Player 1 (Player 2) when the current state is in S_1 (respectively, S_2). A *weighted game graph* is a game graph along with a weight function v from E to $\mathbb{Q}$. A run in the game graph G is called a *play*. The set of all plays is denoted by Ω.

When the two players represent the choices internal to a system, we call the game graph an *alternating transition system*. We only consider alternating transition systems where the transitions from Player 1 states go only to Player 2 states and vice-versa. We use A^F to denote an alternating transition system A with fairness condition F.

Strategies. Given a game graph G, a *strategy* for Player 1 is a function $\pi : (S \cdot \Sigma)^*S_1 \to S \times \Sigma$ such that $\forall \rho_0\sigma_0\rho_1\sigma_1 \ldots \rho_i \in (S \cdot \Sigma)^*S_1$, we have that if $\pi(\rho_0\sigma_0\rho_1\sigma_1 \ldots \rho_i) = (\rho, \sigma)$, then $(\rho_i, \sigma, \rho) \in E$. A strategy for Player 2 is defined in a similar way. The set of all strategies for Player p is denoted by Π_p. A play $\rho = \rho_0\sigma_0\rho_1\sigma_1\rho_2\sigma_2 \ldots$ *conforms* to a player p strategy π if $\forall i \geq 0 : (\rho_i \in S_p \implies : (\rho_{i+1}, \sigma_{i+1}) = \pi(\rho_0\sigma_0\rho_1\sigma_1 \ldots \rho_i))$. The *outcome* of a Player 1 strategy π_1 and a Player 2 strategy π_2 is the unique play $out(\pi_1, \pi_2)$ that conforms to both π_1 and π_2.

Two restricted notions of a strategy are sufficient for many classes of games. A *memoryless strategy* is one where the value of the strategy function depends solely on the last state in the history, whereas a *finite-memory strategy* is one where the necessary information about the history can be summarized by a bounded amount of information. For formal definitions, refer to any standard work on graph games.

Games and objectives. A *game* is a game graph and a boolean or quantitative objective. A *boolean objective* is a function $\Phi : \Omega \to \{0, 1\}$ and the goal of Player 1 in a game with objective Φ is to choose a strategy so that, no matter what Player 2 does, the outcome maps to 1; the goal of Player 2 is to ensure that the outcome maps to 0. A *quantitative objective* is a *value* function $f : \Omega \to \mathbb{R}$ and the goal of Player 1 is to maximize the value f of the play, whereas the goal of Player 2 is to minimize it. Given a boolean objective Φ, a play ρ is *winning* for Player 1 (Player 2) if $\Phi(\rho) = 1$ ($\Phi(\rho) = 0$). A strategy π is a *winning strategy* for Player p if every play conforming to π is winning for Player p.

For a quantitative objective f, the value of the game for a Player 1 strategy π_1, denoted by $\nu_1(\pi_1)$, is defined as the minimum value of the outcome of the play resulting from a Player 2 strategy, i.e., $\nu_1(\pi_1) = \inf_{\pi_2 \in \Pi_2} f(out(\pi_1, \pi_2))$. The value of the game for Player 1 is defined as the supremum of the values of all Player 1 strategies, i.e., $\sup_{\pi_1 \in \Pi_1} \nu_1(\pi_1)$. The value of a Player 2 strategy π_2 and the value of the game for Player 2 are defined analogously as $\nu_2(\pi_2) = \sup_{\pi_1 \in \Pi_1} f(out(\pi_1, \pi_2))$ and $\inf_{\pi_2 \in \Pi_2} \nu_2(\pi_2)$. A strategy is an *optimal strategy* for a player if the value of the strategy for that player is equal to the value of the game. Similarly, a strategy is an *ϵ-optimal strategy* for a maximizing (resp. minimizing) player if the value of the strategy for that player is no more that ϵ is smaller (resp. larger) than the value of the game.

We consider ω-regular boolean objectives and the following quantitative objectives. Given a game graph with the weight function v and a play $\rho = \rho_0\rho_1\rho_2 \ldots$, for all $i \geq 0$, let $v_i = v((\rho_i, \sigma_i, \rho_{i+1}))$.

- $LimAvg(\rho) = \liminf_{n\to\infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} v_i$
- $Disc_\lambda(\rho) = \lim_{n\to\infty}(1-\lambda) \cdot \sum_{i=0}^{n-1} \lambda^i \cdot v_i$ where $0 < \lambda < 1$.

LimAvg is the long-run average of the weights occurring in a play, whereas *Disc* is the discounted sum of the weights. Therefore, *LimAvg* gives more importance to the infinite suffix of a play whereas *Disc* gives more importance to the finite prefix of a play.

Note that for *LimAvg* and *Disc* objectives, optimal memoryless strategies exist for both players [22]. Also, for qualitative objectives specified as Büchi conditions, memoryless winning strategies exist for both players, and for other ω-regular conditions, finite-memory winning strategies exist.

Also, consider the following family of objectives where a boolean ω-regular objective and a quantitative objective f are combined as follows. If a play ρ satisfies the boolean objective, then the value of ρ is the value according to f; otherwise, the value of ρ is the maximum possible value of f (in our case, it is always 1). When $f = LimAvg$ and the ω-regular objective is a parity objective, ϵ-optimal finite-memory strategies exist [7]. This result can be extended to arbitrary ω-regular objectives as all ω-regular objectives can be expressed as parity objectives with the *latest appearance records* memory [14]. Such objectives are called ω-regular *LimAvg* objectives.

2.1. Qualitative simulation games

The simulation preorder [19] is a useful and polynomially computable relation to compare two transition systems. In [1], this relation was extended to alternating simulation between alternating transition systems. For systems with fairness conditions, the simulation relation was extended to fair simulation in [16]. These relations can be computed by solving games with boolean objectives.

Simulation and alternating simulation. Consider two transition systems $A = \langle S, \Sigma, E, s_0\rangle$ and $A' = \langle S', \Sigma, E', s_0'\rangle$. The system A' *simulates* the system A if there exists a relation $H \subseteq S \times S'$ such that (a) $(s_0, s_0') \in H$; (b) $\forall s, t \in S, s' \in S' : (s, s') \in H \wedge (s, \sigma, t) \in E \Rightarrow (\exists t' : (s', \sigma, t') \in E' \wedge (s', t') \in H)$.

For two alternating transition systems $A = \langle S, S_1, S_2, \Sigma, E, s_0\rangle$ and $A' = \langle S', S_1', S_2', \Sigma, E', s_0'\rangle$, *alternating simulation* of A by A' holds if there exists a relation $H \subseteq S \times S'$ such that $(s_0, s_0') \in H$ and $\forall s \in S, s' \in S' : (s, s') \in H \Rightarrow \big(s \in S_1 \Leftrightarrow s' \in S_1'\big)$; and:

- $\forall s \in S, s' \in S' : ((s, s') \in H \wedge s \in S_1) \Rightarrow \forall (s, \sigma, t) \in E : (\exists (s', \sigma, t') \in E' : (t, t') \in H)$.
- $\forall s \in S, s' \in S' : ((s, s') \in H \wedge s \in S_2) \Rightarrow \exists (s', \sigma, t') \in E' : (\forall (s, \sigma, t) \in E : (t, t') \in H)$.

Simulation and alternating simulation games. Given two (alternating) transition systems, A and A', we can construct a game $\mathcal{G}_{A,A'}$ ($\mathcal{H}_{A,A'}$) such that, (alternating) simulation of A by A' holds if and only if Player 2 has a winning strategy in $\mathcal{G}_{A,A'}$ ($\mathcal{H}_{A,A'}$). We define quantitative simulation game graphs. The quantitative version of these game graphs are not necessary to define the classical simulation and alternating simulation games. However, they are introduced here as they will be used later to define quantitative simulation games.

Given two weighted transition systems A and A' with the same alphabet, we define the corresponding *quantitative simulation game graph* $G_{A,A'}$ as $\langle S \times (\Sigma \cup \{\#\}) \times S' \cup \{s_{\mathrm{err}}\}, S_1^G, S_2^G, \Sigma, E^G, (s_0, \#, s_0')\rangle$ where $\# \notin \Sigma$. Here, $S_1^G = (S \times \{\#\} \times S') \cup \{s_{\mathrm{err}}\}$ and $S_2^G = (S \times \Sigma \times S')$. Each transition of the game graph corresponds to a transition in either A or A' as follows:

- $((s, \#, s'), \sigma, (t, \sigma, s')) \in E^G \Leftrightarrow (s, \sigma, t) \in E$
- $((s, \sigma, s'), \sigma, (s, \#, t')) \in E^G \Leftrightarrow (s', \sigma, t') \in E'$.

For each of the above transitions, the weight is the same as the weight of the corresponding transition in A or A'. If there is no outgoing transition from a particular state, transitions to s_{err} are added with all symbols. The state s_{err} is a sink with transitions to itself on all symbols. Each of these transitions has weight 1 (as 1 is the maximum possible value of the quantitative games we consider).

For classical simulation games, we consider the same game graph without weights. Now, the boolean objective for the simulation game is as follows. If the play can proceed ad infinitum without reaching s_{err}, then Player 2 wins. If the play arrives at the s_{err} state, then Player 1 wins. We denote this classical simulation game as $\mathcal{G}_{A,A'}$. Intuitively, in every state, Player 1 chooses a transition of A and Player 2 has to match it by picking a transition of A'. If Player 2 cannot match at some point, Player 1 wins that play. It is easy to see that A' simulates A iff there is a winning strategy for Player 2 in $\mathcal{G}_{A,A'}$.

We can extend the simulation game to an alternating simulation game. We informally define the *quantitative alternating simulation game graph*. Given two weighted alternating transition systems A and A' with the same alphabet and having the initial states s_0 and s_0' such that $s_0 \in S_1 \Leftrightarrow s_0' \in S_1'$, the quantitative alternating simulation game graph $H_{A,A'}$ intuitively works as follows. If A is at state s and $s \in S_1$, Player 1 chooses a transition of A and Player 2 has to match it with a transition of A'; if A is at s and $s \in S_2$, Player 2 chooses a transition of A' and Player 1 has to choose a transition of A to match it. If there cannot be a match, the control moves to the error state s_{err}. As before, the transitions have the same weight as in the individual systems.

Formally, given two weighted alternating transition systems $A = \langle S, S_1, S_2, \Sigma, E, s_0, v\rangle$ and $A' = \langle S', S'_1, S'_2, \Sigma, E', s'_0, v'\rangle$ with the same alphabet, the *alternating simulation game graph* $H_{A,A'} = \langle S^H, S^H_1, S^H_2, \Sigma, E^H, s^H_0, v^H\rangle$ is defined as follows:

- The alphabet is the same as the alphabet of A and A'. The initial state is $(s_0, \#, s'_0, p)$ where p is 1 (2) if s_0 and s'_0 are both Player 1 (respectively, Player 2) states. Note that if one of them is a Player 1 state and the other is a Player 2 state, then alternating simulation of A by A' cannot hold and hence, we do not define the game graph for such cases.
- Player 1 states of the graph are $S^H_1 = \{(s, \#, s', 1) \mid s \in S_1 \wedge s' \in S'_1\} \cup \{(s, \sigma, s', 1) \mid s \in S_2 \wedge s' \in S'_1 \wedge \sigma \in \Sigma\} \cup \{s_{\mathrm{err}}\}$. The first set of the union represents the states where Player 1 chooses a transition for Player 2 to match and the second set represents the states where Player 2 has already chosen a transition with the symbol σ and Player 1 has to match it. State s_{err} is an error state.
- Player 2 states of the graph are $S^H_2 = \{(s, \#, s', 2) \mid s \in S_2 \wedge s' \in S'_2\} \cup \{(s, \sigma, s', 2) \mid s \in S_2 \wedge s' \in S'_1 \wedge \sigma \in \Sigma\}$. The sets in this union are analogous to the ones in Player 1 states.
- The transitions correspond to A or A' transitions as follows:
  - Suppose (s, σ, t) $[(s', \sigma, t')]$ is transition of A $[A']$ and $(s, \#, s', 1)$ $[(s, \#, s', 2)]$ is a Player 1 [Player 2] state. We have the corresponding transition $((s, \#, s', 1), \sigma, (t, \sigma, s', 2))$ $[((s, \#, s', 2), \sigma, (s, \sigma, t', 1))]$ in E^H, i.e., in states where Player 1 [Player 2] has to choose a transition of A $[A']$, the A $[A']$ component of the state and the symbol are changed to the destination and the symbol of the A $[A']$ transition, respectively.
  - If (s, σ, t) $[(s', \sigma, t')]$ is a transition in A $[A']$ and $(s, \sigma, s', 1)$ $[(s, \sigma, s', 2)]$ is a Player 1 [Player 2] state, we have the corresponding transition $((s, \sigma, s', 1), \sigma, (t, \#, s', 1))$ $[((s, \sigma, s', 2), \sigma, (s, \#, t', 2))]$ in E^H. Here, Player 1 [Player 2] chooses a transition to match the previous move of Player 2 [Player 1]. The A component of the state is changed accordingly and the symbol is reset to #.

  The weight of each transition is equal to the weight of the corresponding A or A' transition.
- If there is no outgoing transition from a particular state, we add weight 1 transitions to s_{err} as in the previous game graph.

We consider the game graph without weights to define the alternating simulation game $\mathcal{H}^{A,A'}$ and the objective of the Player 1 is to ensure that the play reaches s_{err}. It can be seen that alternating simulation holds iff there exists a winning strategy for Player 2 .

Fair simulation. Given two (alternating) transition systems with fairness conditions A^F and $A'^{F'}$, the fair simulation game is played in the same game graph $G_{A,A'}$ $(H_{A,A'})$ as the simulation game. However, in addition to matching the symbol in each step, Player 2 has to ensure that if the sequence of transitions of A chosen by Player 1 satisfies the fairness condition F, then the sequence of A' transitions chosen satisfies the fairness condition F'.

2.2. Quantitative simulation games

We define a generalized notion of simulation games called quantitative simulation games where the simulation objectives are replaced by quantitative objectives.

Quantitative simulation games. Given two weighted (alternating) transition systems A and A', and $f \in \{LimAvg, Disc\}$, the *quantitative (alternating) simulation game* is played on the quantitative (alternating) simulation game graph $G_{A,A'}$ $(H_{A,A'})$ with the objective of Player 1 being to maximize the f value of the play. We denote this game as $\mathcal{Q}^f_{A,A'}$ $(\mathcal{P}^f_{A,A'})$.

Quantitative fair simulation games. Analogous to quantitative (alternating) simulation games, the fair versions between two transition systems with fairness conditions A^F and $A'^{F'}$ are played on the same quantitative (alternating) simulation game graph. The quantitative objective for this game is the ω-regular *LimAvg* objective which is the combination of *LimAvg* objective and the boolean fair (alternating) simulation game objective. If a play does not satisfy the boolean objective, it is given a value of 1. The fairness conditions for a quantitative (alternating) simulation game will be implicit when dealing with systems with fairness conditions.

We do not use $f = Disc$ along with fairness conditions as the two objectives are independent. The *Disc* objectives mainly consider the finite prefix of a play, whereas fairness conditions consider only the infinite suffix. Whenever a quantitative (alternating) simulation game with *Disc* objectives is mentioned, it is understood that there are no fairness conditions on the systems.

2.3. Modification schemes

We will use quantitative simulation games to measure various properties of systems. For computing these properties, we need to use small modifications of the original systems. For example, when trying to compute the distance as the number of errors an implementation commits, we add to the specification some error recovery behavior. However, we impose strict rules on these modifications to ensure that the modified system retains the structure of the original system.

A *modification scheme* is a function m from transition systems to weighted (alternating) transition systems, which can be computed using the following steps: (a) Edges may be added to the transition system and each state may be replaced by a local subgraph. All edges of the graph have to be preserved; (b) Every edge of the system is associated with a weight from $\mathbb{Q}$. We present two examples of modification schemes.

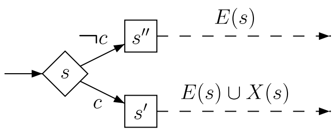
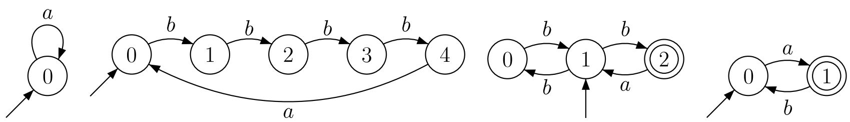


Fig. 2. Graph for *ErrMod*.



(a) I_1. (b) I_5. (c) S_L. (d) I_L.

Fig. 3. Example systems.

Output modification. This scheme is used to add behavior to a system that allows it to output an arbitrary symbol while moving to a state specified by an already existing transition. For every transition (s, σ, s'), transitions with different symbols are added to the system i.e., $\{(s, \alpha, s') \mid \alpha \in \Sigma\}$. These transitions are given a weight of 2 to prohibit their free use. All other transitions have the weight zero. Given a system T, we denote the modified system as $OutMod(T)$.

Error modification. In a perfectly functioning system, errors may occur due to unpredictable events. We model this with an alternating transition system with one player modeling the original system (Player 1) and the other modeling the controlled error (Player 2). At every state, Player 2 chooses whether or not an error occurs by choosing one of the two successors. From one of these states, Player 1 can choose the original successors of the state and from the other, she can choose either one of the original successors or one of the error transitions.

Given $T = \langle S, \Sigma, E, s_0\rangle$, we define $ErrMod(T)$ to be the weighted alternating transition system obtained by replacing each state s by the graph in Fig. 2. If an error is allowed (modeled by the c edge), then all transitions that differ from original transitions only in the symbol are added (represented by $X(s)$ in Fig. 2). All transitions are given the weight 0. We define a special case of *ErrMod* as $ErrMod^*(T)$ denotes a system where no additional transitions are introduced; only the states were replaced by a subgraph from Fig. 2 (with X being the empty set). In this special case, all the transitions from state s'' in the gadget are given the weight 4. We use $ErrMod^*$ to penalize a player if she does not allow errors.

In addition to the above schemes, we define the trivial modification scheme *NoMod* where no changes are made except to give every edge the weight 0.

3. Simulation distances

3.1. Correctness

Given a specification T_2 and an implementation T_1, such that T_1 is incorrect with respect to T_2, the correctness distance measures the degree of "incorrectness" of T_1. The boolean (fair) simulation relation is very strict in a certain way. Even a single nonconformant behavior can destroy this relation. Here we present a game which is not as strict and measures the minimal number of required errors, i.e. the minimal number of times the specification has to use nonmatching symbols when simulating the implementation.

Definition 1 (*Correctness Distance*)**.** Let $f = LimAvg$ or $f = Disc_{\sqrt{\frac{\lambda}{2}}}$. The *correctness distance* $d^f_{\mathrm{cor}}(T_1, T_2)$ from system T_1 to system T_2 is the Player 1 value of the quantitative simulation game $\mathcal{C}^f_{T_1,T_2} = \mathcal{Q}^f_{NoMod(T_1),OutMod(T_2)}$.

The game $\mathcal{C}$ can be intuitively understood as follows. Given two systems T_1 and T_2, we are trying to simulate the system T_1 by T_2, but the specification T_2 is allowed to make errors, to "cheat", but she has to pay a price for such a choice. As the simulating player is trying to minimize the value of the game, she is motivated not to cheat. The value of the game can thus be seen as measuring how often she can be forced to cheat, that is, how often on average the implementation commits an error. If the implementation is correct (T_2 simulates T_1), then the correctness distance is 0. The value of the game is either the *LimAvg* or the $Disc_\lambda$ of the number of errors. We use the value $\sqrt{\lambda/2}$ for discounting to normalize the value of the games. If the objective f is *LimAvg*, then the value is the long run average of the errors, whereas if the objective f is *Disc*, the errors which occur earlier are given more importance and the value is the discounted sum of the positions of the errors. Therefore, the *Disc* and *LimAvg* games are concerned with prefixes and infinite suffixes of the behaviors, respectively.

We present a few example systems and their distances here to demonstrate the fact that the above game measures distances that correspond to intuition. In Figs. 3 and 1, S_1 is the specification system against which we want to measure the systems I_1 through I_5. In this case, the specification says that there cannot be more than two b's in a row. Also, we have a specification with a liveness condition S_L against which we want to measure the implementation I_L. The distances between these systems according to the *LimAvg* correctness game are summarized in Table 1.

Table 1
Distances according to the correctness, coverage, and robustness games.

T_1	T_2	$d_{\mathrm{cor}}^{LimAvg}(T_1, T_2)$	$d_{\mathrm{cov}}^{LimAvg}(T_1, T_2)$	$d_{\mathrm{rob}}^{LimAvg}(T_1, T_2)$
S_1	S_1	0	0	1
I_1	S_1	0	2/3	1/3
I_2	S_1	0	1/3	2/3
I_3	S_1	1/3	1	1
I_4	S_1	1/4	1	1
I_5	S_1	1/5	1	1
I_L	S_L	1/2	1	1

Among the systems which do not satisfy the specification S_1, i.e. I_3, I_4 and I_5, we showed in the introduction that the distance from I_3 to S_1 is 1/3, while the distance from I_4 to S_1 is 1/4. However, surprisingly the distance from I_5 to S_1 is less than the distance from I_4. In fact, the distances reflect on the long run the number of times the specification has to err to simulate the implementation.

In case of the specification S_L and implementation I_L with liveness conditions, the specification can take the left branch to state 0 to get a penalty of $\frac{1}{2}$ or take the right branch to state 2 to get a penalty of 1. However, it needs to take the right branch infinitely often to satisfy the liveness condition. To achieve the distance of $\frac{1}{2}$, the specification needs infinite memory so that it can take the right branch lesser and lesser number of times. In fact, if the specification has a strategy with finite-memory of size m, it can achieve a distance of $\frac{1}{2} + \frac{1}{2m}$.

3.2. Coverage

We present the dual of the correctness distance which measures the behaviors present in specification T_2 system but not in the implementation T_1. The coverage distance corresponds to the behavior of the specification farthest from any behavior of the implementation. Hence, we have that the coverage distance from T_1 to T_2 is the correctness distance from T_2 to T_1.

Definition 2 (*Coverage Distance*)**.** Let $f = LimAvg$ or $f = Disc_{\sqrt{\frac{\lambda}{2}}}$. The *coverage distance* $d_{\mathrm{cov}}^{f}(T_1, T_2)$ from system T_1 to system T_2 is the Player 1 value of the quantitative simulation game $\mathcal{V}^{f}_{T_1,T_2} = \mathcal{Q}^{f}_{NoMod(T_2),OutMod(T_1)}$.

$\mathcal{V}$ measures the minimal number of errors that have to be committed by T_1 to cover all the behaviors of T_2. We present examples of systems and their distances according to $\mathcal{V}^{LimAvg}$. Example distances are summarized in Table 1.

3.3. Robustness

Given a specification system and a correct implementation of the specification, the notion of robustness presented here is a measure of the number of errors by the implementation that makes it nonconformant to the specification. The more such errors tolerated by the specification, the more robust the implementation is with respect to the specification. In other words, the distance measures the number of critical points, or points where an error will lead to an unacceptable behavior. The lower the value of the robustness distance to a given specification, the more robust an implementation is. In case of an incorrect implementation, the simulation of the implementation does not hold irrespective of implementation errors. Hence, in that case, the robustness distance will be 1.

Definition 3 (*Robustness Distance*)**.** Let $f = LimAvg$ or $f = Disc_{\sqrt[4]{\frac{\lambda}{4}}}$. The *robustness distance* $d_{\mathrm{rob}}^{f}(T_1, T_2)$ from system T_1 to system T_2 is the Player 1 value of the quantitative alternating simulation game $\mathcal{R}^{f}_{T_1,T_2} = \mathcal{P}^{f}_{ErrMod(T_1),ErrMod^*(T_2)}$.

As before, the value $\sqrt[4]{\frac{\lambda}{4}}$ is used for discounting to normalize the value of the games. The game $\mathcal{R}_{ErrMod(T_1),ErrMod^*(T_2)}$ is played in the following steps: (a) The specification T_2 chooses whether the implementation T_1 is allowed to make an error; (b) The implementation chooses a transition on the implementation system. It is allowed to err based on the specification choice in the previous step; (c) Specification chooses a matching move to simulate the implementation.

The specification tries to minimize the number of moves where it prohibits implementation errors (without destroying the simulation relation), whereas the implementation tries to maximize it. Intuitively, the positions where the specification cannot allow errors are the critical points for the implementation.

In the game played between S_1 and S_1, every position is critical. At each position, if an error is allowed, the system can output three b's in a row by using the error transition to return to state 0 while outputting a b. The next two moves can be b's irrespective of whether errors are allowed or not. This breaks the simulation. Now, consider I_1. This system can be allowed to err every two out of three times without violating the specification. This shows that I_1 is more robust than S_1 for implementing S_1. The list of distances is summarized in Table 1.

3.4. Computation of simulation distances

The computational complexity of computing the three distances defined here is the same as solving the value problem for the respective games. First, note that given two automata with state spaces of size $|S|$ and $|S'|$, and $|E|$ and $|E'|$ edges, the product game graph has $|S||S'|$ states and $|S||E'| + |S'||E|$ edges.

For systems without fairness conditions, the d_{cor}, d_{cov} and d_{rob} games are simple graph games with *LimAvg* or *Disc* objectives. The decision problem (deciding whether the value is greater than a given value) for these games is in NP ∩ co-NP [22], but no PTIME algorithm is known. However, for *LimAvg* objectives the existence of a pseudo-polynomial algorithm, i.e., polynomial for unary encoded weights, implies that the computation of the distances can be achieved in polynomial time. This is due to the fact that we use constant weights. Using the algorithm of [22], in the case without fairness conditions d_{cor}, d_{cov} and d_{rob} distances can be computed in time $O((|S||S'|)^3 \cdot (|E||S'| + |E'||S|))$ where S and S' are state spaces of the two transition systems; E and E' are the sets of transitions of the two systems. A variation of the algorithm in [22] gives a PTIME algorithm for the *Disc* objectives (given a fixed discounting factor).

For systems with Büchi (weak fairness) conditions, the corresponding games are graph games with *LimAvg* parity games, for which the decision problem is in NP ∩ co-NP. However, the use of constant weights and the fact that the implication of two Büchi conditions can be expressed as a parity condition, with no more than 3 priorities, leads to a polynomial algorithm. Using the algorithm presented in [7], we get a $O((|S||S'|)^3 \cdot (|E||S'| + |E'||S|))$ algorithm.

For systems with Streett (strong fairness) conditions, the corresponding games are graph games with *LimAvg* ω-regular conditions. For an ω-regular *LimAvg* game of n states, we can use the latest appearance records to convert into an equivalent parity game of $2^{O(n\log(n))}$ states and edges, and n priorities. The algorithm of [7] gives a $2^{O(n\log(n))}$ algorithm where $n = |S|\cdot|S'|$.

4. Properties of simulation distances

We present quantitative analogues of boolean properties of the simulation preorders.

4.1. Triangle inequality

Classical simulation relations satisfy the reflexivity and transitivity property which makes them preorders. In an analogous way, we show that the correctness and coverage distances satisfy the quantitative reflexivity and the triangle inequality properties. This makes them directed metrics [10].

Theorem 4. *d^f_{cor} is a directed metric for $f \in \{LimAvg, Disc\}$, i.e.,*

- $\forall S \in \mathcal{S} : d^f_{cor}(S, S) = 0$
- $\forall S_1, S_2, S_3 \in \mathcal{S} : d^f_{cor}(S_1, S_3) \leq d^f_{cor}(S_1, S_2) + d^f_{cor}(S_2, S_3)$.

Proof. We will prove the result for systems with fairness conditions. The case without fairness conditions is analogous. Consider any $\epsilon > 0$. Let τ_2 and τ_3 be $\frac{\epsilon}{2}$-optimal finite strategies for Player 2 in $\mathcal{C}_{S_1,S_2}$ and $\mathcal{C}_{S_2,S_3}$, respectively. Now, we construct a finite-memory strategy τ^* for Player 2 in $\mathcal{C}_{S_1,S_3}$. If M_2 and M_3 are the memories of τ_2 and τ_3, respectively, the memory of τ^* will be $M_2 \times S_2 \times M_3$. The strategy τ^* works as follows. Let the state of the game be $(s_1, \#, s_3)$ and the memory of τ^* be (m_2, s_2, m_3).

- Let Player 1 choose to move according to the S_1 transition (s_1, σ_1, s'_1) to the game state (s'_1, σ_1, s_3). Consider the game position (s'_1, σ_1, s_2) in $\mathcal{C}_{S_1,S_2}$ and let the τ_2 memory be at state m_2. Say τ_2 updates its memory to m'_2 and chooses the successor $(s'_1, \#, s'_2)$ with transition symbol σ_1. Let the corresponding $OutMod(S_2)$ transition be (s_2, σ_1, s'_2).
- If the transition (s_2, σ_1, s'_2) exists in S_2, then let $\sigma'_2 = \sigma_1$. Otherwise, there will exist (s_2, σ_2, s'_2) in S_2 for some σ_2. Let $\sigma'_2 = \sigma_2$. Now, consider the game position (s'_2, σ'_2, s_3) in $\mathcal{C}_{S_2,S_3}$ and the memory state m_3 of τ_3. Say τ_3 updates its memory to m'_3 and chooses the successor $(s'_2, \#, s'_3)$ and the transition symbol σ'_2. Let the corresponding $OutMod(S_3)$ transition be (s_3, σ'_2, s'_3).
- The memory of τ^* is updated to (m'_2, s'_2, m'_3) and τ^* chooses the successor $(s'_1, \#, s'_3)$ with the transition symbol σ_1. The corresponding transition (s_3, σ_1, s'_3) exists in $OutMod(S_3)$ as there exists a transition with the same source and destination as (s_3, σ'_2, s'_3).

$$\left.\begin{array}{l}
\left.\begin{array}{lll}
s_{1,0} & \xrightarrow{\sigma_1(v_{1,0})} s_{1,1} & \xrightarrow{\sigma_1(v_{1,1})} s_{1,2}\cdots \\
s_{2,0} & \xrightarrow{\sigma_1(v_{2,0})} s_{2,1} & \xrightarrow{\sigma_1(v_{2,1})} s_{2,2}\cdots
\end{array}\right\}\rho_1 \\
\left.\begin{array}{lll}
s_{2,0} & \xrightarrow{\sigma'_2(v_{2,0})} s_{2,1} & \xrightarrow{\sigma'_2(v_{2,1})} s_{2,2}\cdots \\
s_{3,0} & \xrightarrow{\sigma_1(v_{3,0})} s_{3,1} & \xrightarrow{\sigma_1(v_{3,1})} s_{3,2}\cdots
\end{array}\right\}\rho_2
\end{array}\right\}\rho.$$

If Player 2 cannot match σ_1 with a zero weight transition while playing according to τ^*, either τ_2 or τ_3 would have also taken a non-zero weight transition. Using this fact, we can easily prove the required property.

Fix an arbitrary finite-memory Player 1 strategy σ. Now, let the play proceed according to the strategy τ^*. From the moves of the game and the state of the memory of τ^*, we can extract four transitions for each round of play as above, i.e. an S_1 transition (s_1, σ_1, s_1'), an $OutMod(S_2)$ transition (s_2, σ_1, s_2'), an S_2 transition (s_2, σ_2', s_2') and an $OutMod(S_3)$ transition (s_3, σ_1, s_3'). We depict the situation in the above figure.

The play ρ in $\mathcal{C}_{S_1,S_3}$ corresponds to the transitions in the first and the last rows. This play can be decomposed into plays ρ_1 and ρ_2 in $\mathcal{C}_{S_1,S_2}$ and $\mathcal{C}_{S_2,S_3}$ by taking only the transitions in the first two and last two rows, respectively. Now, by the observation in the previous paragraph, each move in ρ has weight 2 only if one of the corresponding moves in ρ_1 or ρ_2 have weight 2. Let us denote the nth move in a play η by η^n. If both S_1 and S_3 sequence of moves in ρ are fair or if S_1 sequence is unfair, we have the following for the *LimAvg* case.

$$\begin{aligned}
\nu(\rho) &= \liminf_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho^i) \le \liminf_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} \left(v(\rho_1^i) + v(\rho_2^i)\right) \\
&= \lim_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} \left(v(\rho_1^i) + v(\rho_2^i)\right) = \lim_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho_1^i) + \lim_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho_2^i) \\
&= \liminf_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho_1^i) + \liminf_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho_2^i) \\
&\le d_{\mathrm{cor}}(S_1, S_2) + \frac{\epsilon}{2} + d_{\mathrm{cor}}(S_2, S_3) + \frac{\epsilon}{2} = d_{\mathrm{cor}}(S_1, S_2) + d_{\mathrm{cor}}(S_2, S_3) + \epsilon.
\end{aligned}$$

In the above set of equations we use limit and limit infimum interchangeably. This can be done because all the strategies we are considering are finite-memory, and hence, each sequence of weights is ultimately repeating. Hence, the limit infimum of the average of such a sequence is equal to the limit of the average of the sequence and it converges to the average weight of the repeating sequence. The case for *Disc* is much simpler and not shown here.

Hence, we have that the value of the play satisfies the required inequality for the case that both S_1 and S_3 perform fair computations. In the case that S_1 sequence is fair and S_3 sequence is not fair, the value of the play will be 1. However, by construction, the value of either ρ_1 or ρ_2 will also be 1 and hence the inequality holds.

Therefore, given an epsilon, we have demonstrated a finite-memory strategy for Player 2 such that, for every finite-memory Player 1 strategy, the value of the game is less than $d_{\mathrm{cor}}(S_1, S_2) + d_{\mathrm{cor}}(S_2, S_3) + \epsilon$ for both the *LimAvg* and *Disc* case. Hence, we have the required triangle inequality.

It can be shown by construction of a Player 2 strategy that copies every Player 1 move that $d_{\mathrm{cor}}(S, S) = 0$. Hence, we have the result. □

Theorem 5. *d^f_{cov} is a directed metric when $f \in \{LimAvg, Disc\}$, i.e.,*

- $\forall S \in \mathcal{S} : d^f_{cov}(S, S) = 0$
- $\forall S_1, S_2, S_3 \in \mathcal{S} : d^f_{cov}(S_1, S_3) \le d^f_{cov}(S_1, S_2) + d^f_{cov}(S_2, S_3)$.

Proof. The proof of this proposition follows from the fact that for any two systems S_1 and S_2, we have that $d^f_{\mathrm{cov}}(S_1, S_2) = d^f_{\mathrm{cor}}(S_2, S_1)$. □

The robustness distance satisfies the triangle inequality, but not the quantitative reflexivity. The system S_1 in Fig. 1 is a witness system that violates $d_{\mathrm{rob}}(S_1, S_1) = 0$. In fact, for *LimAvg* objectives and any rational value $v \in [0, 1]$, it is easy to construct a system S_v such that $d_{\mathrm{rob}}(S_v, S_v) = v$.

Theorem 6. *d^f_{rob} conforms to the triangle inequality for $f \in \{LimAvg, Disc\}$, i.e. $\forall S_1, S_2, S_3 \in \mathcal{S} : d^f_{rob}(S_1, S_3) \le d^f_{rob}(S_1, S_2) + d^f_{rob}(S_2, S_3)$.*

Proof. We will consider systems which have fairness conditions and the case without fairness conditions is subsumed by this.

Given any $\epsilon > 0$, we will proceed to prove this result along the same lines as the proof of Proposition 4 by constructing a strategy for Player 2 in $\mathcal{R}_{S_1,S_3}$ from $\frac{\epsilon}{2}$-optimal strategies of Player 2 in $\mathcal{R}_{S_1,S_2}$ and $\mathcal{R}_{S_2,S_3}$. Let the $\frac{\epsilon}{2}$-optimal strategies for Player 2 in $\mathcal{R}_{S_1,S_2}$ and $\mathcal{R}_{S_2,S_3}$ be τ_2 and τ_3, respectively. We construct a strategy τ^* for Player 2 in $\mathcal{R}_{S_1,S_3}$ with the memory $M_2 \times S_2 \times M_3$ where M_2 and M_3 are the finite memories of τ_2 and τ_3, respectively.

In the following description, we will denote a state of the alternating game (s_i, σ_k, s_j, p) by (s_i, σ_k, s_j). Let $(s_1, \#, s_3)$ be the state of the game and (m_2, s_2, m_3) be the state of the memory of τ^*. Let us assume for the moment that the simulation of s_1 by s_2 and s_2 by s_3 always holds. The strategy τ^* works as described below.

(1) Suppose $(s_1, \#, s_3)$ is a state where Player 2 has to choose either the c or $\neg c$ edge to decide whether Player 1 is allowed to take an error transition in the next step. If either τ_2 chooses the c edge at $(s_1, \#, s_2)$ and memory state m_2 or τ_3 chooses the c edge at $(s_2, \#, s_3)$ and memory state m_3, τ_* chooses the c edge. The memory of τ^* is updated to the corresponding updated memories of τ_2 and τ_3 and the S_2 state chosen by τ_2.

(2) Suppose $(s_1, \#, s_3)$ is a state where Player 2 has to simulate the Player 1 move. Let the Player 1 move to (s_1', σ_1, s_3) according to the transition (s_1, σ_1, s_1').
   (a) If $\neg c$ is chosen in the previous step, there can be no erroneous transitions and every Player 1 move can simulated, as we have assumed that s_1 can be simulated by s_2 and s_2 can be simulated by s_3. Suppose τ_2 updates its memory to m_2' and moves to $(s_1', \#, s_2')$ on the symbol σ_1 from the game position (s_1', σ_1, s_2) and memory state m_2. Also, suppose τ_3 updates its memory to m_3' and moves to $(s_2', \#, s_3')$ on symbol σ_1 from the game position (s_2', σ_1, s_3) and memory state m_3. τ^* updates its memory to (m_2', s_2', m_3') and choose the successor $(s_1', \#, s_2')$ and the transition symbol σ_1. The corresponding transitions for $ErrMod^*(S_3)$ is (s_3, σ_1, s_3').
   (b) Now, if c was chosen in the previous step, we have the two possibilities: either c was chosen as it was the choice of τ_2 or τ_3. We consider the two cases separately:
      (τ_2) If τ_2 chose c, it means that every move of $ErrMod(S_1)$ from s_1 (including the erroneous moves) can be simulated by $ErrMod(S_2)$. Therefore, we update the memory and choose the τ^* move as in the previous case.
      (τ_3) If τ_2 choice was $\neg c$, but τ_3 choice was c, we have the following:
         (i) For every $ErrMod(S_1)$ transition from s_1 to s_1' on σ, there is a non-erroneous S_1 transition between the same states (by definition, say on symbol σ_1'). Let τ_2 update its memory to m_2' and choose the successor $(s_1', \#, s_2')$ on transition symbol σ_1' from the game position (s_1', σ_1', s_2) in $\mathcal{R}_{S_1,S_2}$.
         (ii) Now, let τ_3 update its memory to m_3 and move to $(s_2', \#, s_3')$ on σ_1 in the game position (s_2', σ_1, s_3) and memory state m_3.
         (iii) Now, the τ^* chooses the successor $(s_1', \#, s_3')$ and the transition symbol σ_1 and updates its memory to (m_2', s_2', m_3').

As in the proof of Proposition 4, we can decompose any play of $\mathcal{R}_{S_1,S_3}$ conforming to τ^* into two plays of $\mathcal{R}_{S_1,S_2}$ and $\mathcal{R}_{S_2,S_3}$ using the memory of τ^*. Also, we have the case that there is a non-zero weight move in $\mathcal{R}_{S_1,S_3}$ if and only if there is corresponding non-zero weight move in either $\mathcal{R}_{S_1,S_2}$ or $\mathcal{R}_{S_2,S_3}$. Hence, by the same arguments as in the previous proof, we get the required inequality in the case that the simulation always holds.

Now, we just have to consider the case where the simulation in $\mathcal{R}_{S_1,S_3}$ breaks. Due to the way τ^* is defined, the simulation between $ErrMod(S_1)$ and $ErrMod^*(S_3)$ breaks in $\mathcal{R}_{S_1,S_3}$, if and only if the simulation breaks in $\mathcal{R}_{S_1,S_2}$ or $\mathcal{R}_{S_2,S_3}$. Hence, we have $d_{\mathrm{rob}}(S_1, S_3) = 1$ due to the failure of simulation if and only if $d_{\mathrm{rob}}(S_2, S_3) = 1$ or $d_{\mathrm{rob}}(S_1, S_2) = 1$, which will give us the required inequality. □

4.2. Compositionality

In the qualitative case, compositionality theorems help analyze large systems by decomposing them into smaller components. For example, if T_1 simulates S_1 and T_2 simulates S_2, we have that the composition of T_1 and T_2 simulates the composition of S_1 and S_2. We show that in the quantitative case, the distance between the composed systems is bounded by the sum of the distances between individual systems.

If A and A' are two transition systems, we define *asynchronous and synchronous composition* of the two systems, written as $A \parallel A'$ and $A \times A'$, respectively as follows: (a) The state space is $S \times S'$; (b) $((s, s'), \sigma, (t, t'))$ is a transition of $A \parallel A'$ iff (s, σ, t) is a transition of A and $s' = t'$ or (s', σ, t') is a transition of A' and $s = t$; (c) $((s, s'), \sigma, (t, t'))$ is a transition of $A \times A'$ iff (s, σ, t) is a transition of A and (s', σ, t') is a transition of A'.

The following theorems show that the simulation distances between whole systems is no more than the sum of the distances between the individual components.

Theorem 7. *The correctness, coverage, and robustness distances satisfy the following property for $f \in \{LimAvg, Disc\}$: $\forall S_1, S_2, T_1, T_2 : d^f(S_1 \times S_2, T_1 \times T_2) \leq d^f(S_1, T_1) + d^f(S_2, T_2)$.*

Proof. Let us consider the correctness game first. Let $\mathcal{C}_{S_1,T_1}$ and $\mathcal{C}_{S_2,T_2}$ be the games for computing $d_{\mathrm{cor}}(S_1, T_1)$ and $d_{\mathrm{cor}}(S_2, T_2)$. Let τ_1 and τ_2 be $\frac{\epsilon}{2}$-optimal strategies for Player 2 in the $\mathcal{C}_{S_1,T_1}$ and $\mathcal{C}_{S_2,T_2}$, with memory M_1 and M_2, respectively. We define a strategy τ^* for Player 2 in $\mathcal{C}_{S_1 \times S_2, T_1 \times T_2}$ with memory $M_1 \times M_2$. τ^* works by playing τ_1 and τ_2 component-wise.

At state (s_1, σ, t_1) and memory m_1, let τ_1 update its memory to m_1' and move to $(s_1, \#, t_1')$ with symbol σ and at state (s_2, σ, t_2) and memory m_2, let τ_2 update its memory to m_2' and move to $(s_2, \#, t_2')$ with the symbol σ. Now, at the state $((s_1, s_2), \sigma, (t_1, t_2))$ and memory (m_1, m_2), τ^* updates its memory to (m_1', m_2') and chooses to move to the successor $((s_1, s_2), \#, (t_1', t_2'))$ on symbol σ.

Now, any play ρ conforming to τ^* in the game $\mathcal{C}_{S_1 \times S_2, T_1 \times T_2}$ can be split component-wise into plays ρ_1 conforming to τ_1 and ρ_2 conforming to τ_2 in games $\mathcal{C}_{S_1,T_1}$ and $\mathcal{C}_{S_2,T_2}$, respectively. A move in ρ is a non-zero weight move if and only if at least one of the two corresponding moves in ρ_1 and ρ_2 has a non-zero weight. Using this fact, as in proof of Proposition 4, we can show that the value of the play ρ is no more than ϵ larger than the sum of the values of the two games $\mathcal{C}_{S_1,T_1}$ and $\mathcal{C}_{S_2,T_2}$. Hence, we get the result for the correctness distance. Also, the proof for the coverage distance follows from the fact that $d_{\mathrm{cov}}(A, B) = d_{\mathrm{cor}}(B, A)$ for any two systems A and B.

For the robustness distance, we can prove the result by a similar component-wise strategy construction of τ^* from $\frac{\epsilon}{2}$-optimal finite-memory strategies τ_1 and τ_2 for $\mathcal{R}_{S_1,T_1}$ and $\mathcal{R}_{S_2,T_2}$. We describe the strategy construction informally as follows: at any point where Player 2 has to choose a c or $\neg c$ transition, τ^* advises Player 2 to pick a c transition if and only if both τ_1

and τ_2 pick a c transition in their respective components. At every point where Player 2 is to simulate the move of Player 1 , τ^* advises Player 2 to move to a state with the same components as τ_1 and τ_2 moves.

As before, we have that any play ρ conforming to τ^* can be decomposed into two individual plays ρ_1 and ρ_2, conforming to τ_1 and τ_2, respectively. As any $\neg c$ transition in ρ arises from at least one of the corresponding transitions in the ρ_1 and ρ_2, we can use the same arguments as above to prove the required inequality. □

Theorem 8. *The correctness, coverage, and robustness distances satisfy the following property for $f = LimAvg$.*

$$\forall S_1, S_2, T_1, T_2 : d^f(S_1 \parallel S_2, T_1 \parallel T_2) \leq \alpha \cdot d^f(S_1, T_1) + (1-\alpha) \cdot d^f(S_2, T_2)$$

where α is the fraction of times S_1 is scheduled in $S_1 \parallel S_2$ in the long run, assuming that the fraction has a limit in the long run.

Proof. As in the proof of Theorem 7, the proof works for all cases by constructing a Player 2 strategy τ^* from the $\frac{\epsilon}{2}$-optimal strategies τ_1 and τ_2 in the games for computing $d(S_1, T_1)$ and $d(S_2, T_2)$, respectively. Let the memories of τ^*, τ_1 and τ_2 be as in the proof of Theorem 7.

For the correctness game, we define τ^* as follows: if Player 1 moves from $((s_1, s_2), \#, (t_1, t_2))$ to $((s_1', s_2), \sigma, (t_1, t_2))$ according to the S_1 transition (s_1, σ, s_2), and τ^* has the memory (m_1, m_2), it responds by playing the τ_1 strategy in the first component, i.e. if from the game position (s_1', σ, t_1) and memory m_1, τ_1 moves to $(s_1', \#, t_1')$ on symbol σ and updates memory to m_1', τ^* chooses to move to $((s_1', s_2), \#, (t_1', t_2))$ with the symbol σ and updates its memory to (m_1', m_2). The response to a Player 1 move in the second component of the system is similar. By similar arguments used in the previous proof, we can prove that τ^* is a witness to the required inequality as follows: let ρ be any play conformant to τ_*. Let $I_1 \subseteq \mathbb{Z}$ be the indices where the move is in the first component and let $I_2 = \mathbb{Z} \setminus I_1$. Now, let ρ_i be the $\mathcal{C}_{S_i, T_i}$ play obtained from ρ by taking on the positions in I_i and projecting it into component i. By construction, we have ρ_i conformant to τ_i. Hence, we get for the $f = LimAvg$ case:

$$\begin{aligned}
\nu(\rho) &= \liminf_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n} v(\rho^i) = \liminf_{n\to\infty} \frac{1}{n} \left(\sum_{i\in I_1}^{i\leq n} v(\rho_1^i) + \sum_{i\in I_2}^{i\leq n} v(\rho_2^i) \right) \\
&= \lim_{n\to\infty} \frac{1}{n} \left(\sum_{i\in I_1}^{i\leq n} v(\rho_1^i) + \sum_{i\in I_2}^{i\leq n} v(\rho_2^i) \right) \\
&= \lim_{n\to\infty} \frac{n_1}{n} \cdot \left(\frac{1}{n_1} \cdot \sum_{i\in I_1}^{i\leq n} v(\rho_1^i) \right) + \frac{n_2}{n} \cdot \left(\frac{1}{n_2} . \sum_{i\in I_2}^{i\leq n} v(\rho_2^i) \right) \quad \text{where } n_i = |\{k \mid k \in I_i \wedge k \leq n\}| \\
&= \alpha \cdot \left(\lim_{n_1\to\infty} \frac{1}{n_1} \cdot \sum_{i\in I_1}^{i\leq n} v(\rho_1^i) \right) + (1-\alpha) \cdot \left(\lim_{n_2\to\infty} \frac{1}{n_2} \cdot \sum_{i\in I_2}^{i\leq n} v(\rho_2^i) \right) \\
&\qquad \text{as } \lim_{n\to\infty} \tfrac{n_1}{n} = \alpha \text{ and } \lim_{n\to\infty} \tfrac{n_2}{n} = 1-\alpha \\
&\leq \alpha \cdot \left(d(S_1, T_1) + \frac{\epsilon}{2} \right) + (1-\alpha). \left(d(S_2, T_2) + \frac{\epsilon}{2} \right) \quad \text{as } \rho_1 \text{ conforms to } \tau_1 \text{ and } \rho_2 \text{ conforms to } \tau_2 \\
&= \alpha \cdot d(S_1, T_1) + (1-\alpha) \cdot d(S_2, T_2) + \frac{\epsilon}{2}.
\end{aligned}$$

Hence, τ^* is a witness strategy that shows the required inequality.

For the robustness game, we define τ^* as follows.

(1) From the state $((s_1, s_2), \#, (t_1, t_2), 2)$ and memory (m_1, m_2), Player 2 chooses the c transition if and only if τ_i chooses the c transition in $(s_i, \#, t_i)$ and memory m_i for $i \in \{1, 2\}$.
(2) From the state $((s_1, s_2), \sigma, (t_1, t_2), 2)$ and memory (m_1, m_2), if Player 1 has moved in the first component, τ^* copies the τ_1 move in the first component and updates the first component of the memory. The same holds for the second component.

Now, for any play ρ conformant to τ^* we can define ρ_1 and ρ_2 as in the correctness case and use the same arguments to give us the inequality. □

4.3. Abstraction

In the boolean case, properties of systems can be studied by studying the properties of over-approximations and under-approximations. In an analogous way, we prove that the distances between two systems is bounded from above and below by distances between abstractions of the two systems. We first define over-approximations and under-approximations of systems.

Given a transition system $S = \langle S, \Sigma, E, s_0 \rangle$ *existential abstraction* and *universal abstraction* of the system are systems $S^\exists = \langle Q, \Sigma, E_Q^\exists, [s_0] \rangle$ and $S^\forall = \langle Q, \Sigma, E_Q^\forall, [s_0] \rangle$, where Q is the set of equivalence classes of some equivalence relation over

Table 2
Robustness of FECS.

T_1	T_2	$d_{\mathrm{rob}}(T_1, T_2)$
None	Ideal	1
Hamming	Ideal	6/7
TMR	Ideal	2/3

S, $[s_0]$ is the equivalence class containing s_0 and $E_Q^{\exists} = \{(q, \sigma, q') \mid \exists s, s' : [s \in q \wedge s' \in q' \wedge (s, \sigma, s') \in E]\}$ for existential abstraction and $E_Q^{\forall} = \{(q, \sigma, q') \mid \forall s, s' : [s \in q \wedge s' \in q' \implies (s, \sigma, s') \in E]\}$ for universal abstraction.

Theorem 9. *Let S and I be systems. Let $S^{\exists}$ and $I^{\exists}$ be existential abstractions, and $S^{\forall}$ and $I^{\forall}$ be universal abstractions of S and I, respectively. The correctness, coverage, and robustness distances satisfy the three following properties, for $f \in \{LimAvg, Disc\}$:*

(a) $d^f_{cor}(I^{\forall}, S^{\exists}) \leq d^f_{cor}(I, S) \leq d^f_{cor}(I^{\exists}, S^{\forall})$
(b) $d^f_{cov}(I^{\exists}, S^{\forall}) \leq d^f_{cov}(I, S) \leq d^f_{cov}(I^{\forall}, S^{\exists})$
(c) $d^f_{rob}(I^{\forall}, S^{\exists}) \leq d^f_{rob}(I, S) \leq d^f_{rob}(I^{\exists}, S^{\forall})$.

Proof. The proof of the lower bound is based on the fact that every behavior of T is present in $T^{\exists}$ and the behaviors present in $T^{\forall}$ are a subset of the behaviors present in T. We prove the lower bounds and proofs for the upper bounds is similar, but considers optimal Player 1 strategies instead of optimal Player 2 strategies considered below.

- Let τ be the ϵ-optimal Player 2 strategy in $\mathcal{C}_{I,S}$ with memory M. We construct the strategy τ^* with memory M as follows: at a game position (q_I, σ, q_S) and memory $m \in M$, let s_I and s_S be I and S states such that from $s_I \in q_I \wedge s_S \in q_S$ and from (s_I, σ, s_S) and memory m in $\mathcal{C}_{I,S}$, Player 2 can ensure that the play value is less that $d_{\mathrm{cor}}(I, S) + \epsilon$ by playing according to τ. Also, let τ update its memory to m' and move to $(s_I, \#, s'_S)$ on σ. Now, at (q_I, σ, q_S) and memory m, τ^* updates its memory to m' and chooses to move to $(q_I, \sigma, [s'_S])$ with the transition symbol σ where $[s'_S]$ is the unique $S^{\exists}$ state containing s'_S.

  Now, we can easily show that if *Player* 2 plays according to τ^*, for every state (q_I, σ, q_S) that occurs in a play, we can find s_I and s_S which satisfy the condition mentioned above. Also, from every play ρ conformant to τ^*, we can extract a play ρ' in $\mathcal{C}_{S,I}$ conformant to τ such that the value of the two plays are equal. Hence, we have demonstrated a strategy that ensures a value less than $d_{\mathrm{cor}}(I, S) + \epsilon$ for every $\epsilon > 0$. This gives us the required result.
- This inequality follows from the previous one easily: $d_{\mathrm{cov}}(I^{\exists}, S^{\forall}) = d_{\mathrm{cor}}(S^{\forall}, I^{\exists}) \leq d_{\mathrm{cor}}(S, I) = d_{\mathrm{cov}}(I, S)$.
- As before, let τ be an ϵ-optimal Player 2 strategy in $\mathcal{R}_{I,S}$ with memory M. We define τ^* with memory M as follows.

  (1) At a game position $(q_I, \#, q_S, 2)$ and memory m, where Player 2 has to choose a c or $\neg c$ edge, suppose there exists $s_S \in q_S \wedge s_I \in q_I$ such that τ chooses the c edge in state $(s_I, \#, s_S, 2)$ and updates its memory to m'. τ^* does the same by choosing the c edge and updating its memory to m'.
  (2) At a game position $(q'_I, \sigma, q_S, 2)$ and memory m, where Player 2 has to simulate the transition (q_I, σ, q'_I), τ^* works by picking a convenient corresponding transition (s_I, σ, s'_I) in $\mathcal{R}_{I,S}$ and using the same move τ and memory update τ uses.

  It is fairly obvious that from any play ρ conformant to τ^*, a play ρ' conformant to τ can be extracted in $\mathcal{R}_{I,S}$ with $\nu(\rho) \leq \nu(\rho')$. As in the case for the correctness distance, this gives us the required result. □

5. Applications of simulation distances

We present two examples of application of the distances defined in Section 3 to measure interesting properties of larger systems.

5.1. Forward error-correction systems

Forward error-correction systems (FECS) are a mechanism of error control for data transmission on noisy channels. The *maximum tolerable bit-error rate* of these systems is the maximum number of errors the system can tolerate while still being able to successfully decode the message. We show that this property can be measured as the d_{rob} between a system and an ideal specification.

We examine three forward error correction systems: one with no error correction facilities, the Hamming(7,4) code [15], and triple modular redundancy [18]. Intuitively, each of these systems is at a different point in the trade-off between efficiency of transmission and the tolerable bit-error rate. By design, the system with no error correction can tolerate no errors and the Hamming(7,4) system can tolerate one error in seven bits and the triple modular redundancy system can tolerate one error in three bits. However, the overhead incurred increases with increasing error tolerance. The system with no error correction uses no extra bits while, the Hamming(7,4) system and the triple modular redundancy system use 3

No error correction

proc sender$(\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3) \equiv$
  call send$(\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3)$; .
proc receiver$() \equiv$
  call receive$(\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3)$;
  return$(\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3)$.

Hamming(7,4) error correction

proc sender$(\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3) \equiv$
  $\mathbf{P}_0 := \mathbf{B}_0 \oplus \mathbf{B}_1 \oplus \mathbf{B}_3$
  $\mathbf{P}_1 := \mathbf{B}_0 \oplus \mathbf{B}_2 \oplus \mathbf{B}_3$
  $\mathbf{P}_2 := \mathbf{B}_1 \oplus \mathbf{B}_2 \oplus \mathbf{B}_3$
  call send$(\mathbf{P}_0, \mathbf{P}_1, \mathbf{B}_0, \mathbf{P}_2, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3)$; .
proc receiver$() \equiv$
  call receive$(\mathbf{P}_0, \mathbf{P}_1, \mathbf{B}_0, \mathbf{P}_2, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3)$;
  $\mathbf{P}_0 := \mathbf{P}_0 \oplus \mathbf{B}_0 \oplus \mathbf{B}_1 \oplus \mathbf{B}_3$;
  $\mathbf{P}_1 := \mathbf{P}_1 \oplus \mathbf{B}_0 \oplus \mathbf{B}_2 \oplus \mathbf{B}_3$;
  $\mathbf{P}_2 := \mathbf{P}_2 \oplus \mathbf{B}_1 \oplus \mathbf{B}_2 \oplus \mathbf{B}_3$;
  $\mathbf{B}_0 := \mathbf{B}_0 \oplus (\neg\mathbf{P}_0.\mathbf{P}_1.\neg\mathbf{P}_2)$;
  $\mathbf{B}_1 := \mathbf{B}_1 \oplus (\mathbf{P}_0.\neg\mathbf{P}_1.\mathbf{P}_2)$;
  $\mathbf{B}_2 := \mathbf{B}_2 \oplus (\mathbf{P}_0.\mathbf{P}_1.\neg\mathbf{P}_2)$;
  $\mathbf{B}_3 := \mathbf{B}_3 \oplus (\mathbf{P}_0.\mathbf{P}_1.\mathbf{P}_2)$;
  return$(\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3)$.

Triple modular redundancy

proc sender$(\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3) \equiv$
  call send$(\mathbf{B}_0, \mathbf{B}_0, \mathbf{B}_0)$;
  call send$(\mathbf{B}_1, \mathbf{B}_1, \mathbf{B}_1)$;
  call send$(\mathbf{B}_2, \mathbf{B}_2, \mathbf{B}_2)$;
  call send$(\mathbf{B}_3, \mathbf{B}_3, \mathbf{B}_3)$; .
proc receiver$() \equiv$
  call receive$(\mathbf{B}_{01}, \mathbf{B}_{02}, \mathbf{B}_{03})$;
  call receive$(\mathbf{B}_{11}, \mathbf{B}_{12}, \mathbf{B}_{13})$;
  call receive$(\mathbf{B}_{21}, \mathbf{B}_{22}, \mathbf{B}_{23})$;
  call receive$(\mathbf{B}_{31}, \mathbf{B}_{32}, \mathbf{B}_{33})$;
  $\mathbf{B}_0 := \mathbf{B}_{01}.\mathbf{B}_{02} \vee \mathbf{B}_{02}.\mathbf{B}_{03} \vee \mathbf{B}_{03}.\mathbf{B}_{01}$;
  $\mathbf{B}_1 := \mathbf{B}_{11}.\mathbf{B}_{12} \vee \mathbf{B}_{12}.\mathbf{B}_{13} \vee \mathbf{B}_{13}.\mathbf{B}_{11}$;
  $\mathbf{B}_2 := \mathbf{B}_{21}.\mathbf{B}_{22} \vee \mathbf{B}_{22}.\mathbf{B}_{23} \vee \mathbf{B}_{23}.\mathbf{B}_{21}$;
  $\mathbf{B}_3 := \mathbf{B}_{31}.\mathbf{B}_{32} \vee \mathbf{B}_{32}.\mathbf{B}_{33} \vee \mathbf{B}_{33}.\mathbf{B}_{31}$;
  return$(\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3)$.

Fig. 4. Forward error correction algorithms.

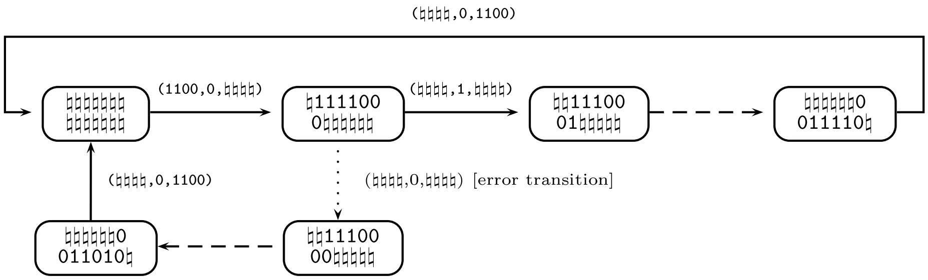


Fig. 5. Part of the transition graph for Hamming(7,4) system.

and 8 extra bits for transmitting a four bit message. We compute the values of the error tolerance by measuring robustness with respect to an ideal system which can tolerate an unbounded number of errors. The ideal system is modeled as a system which non-deterministically sends a number of bits and then outputs the correct message. The pseudo-code for the three systems is presented in Fig. 4. The only errors we allow are bit flips during transmission.

We explain the modeling with an example: we present the transmission of the bit block 1100 in the Hamming(7,4) system (Fig. 5). The encoded bit string for this block is 0111100. In each state, the two components represent the sequence of bits to be sent and the sequence of bits to be received. In each transition symbol, the first component represents the input message to the system; the second component represents the bit transmitted in the current step; and the third component represents the output message from the system. From the initial state $(♮^7, ♮^7)$, on the input 1100, the transmitted bit is 0 (the first bit of the encoded string) and the state changes to (♮111100, 0♮♮♮♮♮♮) on the transitions symbol (1100, 0, ♮♮♮♮) (assuming no errors). From this state, we go on the symbol (♮♮♮♮, 1, ♮♮♮♮) to the state (♮♮11100, 01♮♮♮♮♮) and so on. An error transition from (♮111100, 0♮♮♮♮♮♮) will lead to the state (♮♮11100, 00♮♮♮♮♮).

The values of d_{rob} for these systems are summarized in Table 2. The robustness values clearly mirror the error tolerance values. In fact, each robustness value is equal to 1–e where e is the corresponding error tolerance value.

5.2. Environment restriction for reactive systems

In reactive systems, the transitions of the system are controlled by two agents, the system and the environment. While refining a specification for a reactive system, care has to be taken to ensure (a) all behaviors of the implementation are simulated by the specification, and (b) the behavior of the environment is not restricted more than in the specification. A

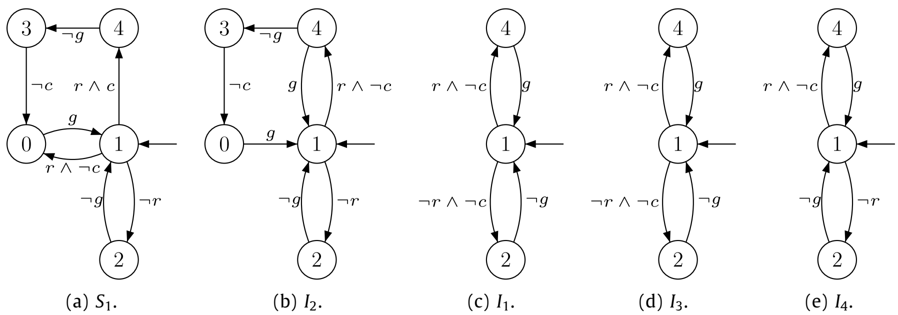


Fig. 6. Reactive systems.

Table 3
Restrictiveness of request-grant systems.

T_1	T_2	$d_{cov}(T_1, T_2)$
S_1	S_1	0
S_1	I_1	1/2
S_1	I_2	1/4
S_1	I_3	1/4
S_1	I_4	0

number of extensions of the classical simulation relation have been suggested to include this requirement such as ready simulation [3].

We propose a method to measure the amount of restriction the implementation system places on the environment over and above the restriction in the specification. The measure proposed here not only takes into consideration the languages of the two systems, but also the distance of the farthest unimplemented behavior in the implementation. For example, consider a specification that allows the environment behavior r_1^ω and two implementations I_1 and I_2 forbid it. However, say I_1 allows the behavior $(r_1r_2)^\omega$ whereas I_2 allows only r_2^ω, I_1 will be given a higher rating than I_2.

We measure the amount of environment restriction using the coverage distance (d_{cov}). We model a reactive system as a transition system with the alphabet $\Sigma^{\mathcal{I}} \cup \Sigma^{\mathcal{O}}$ (where $\mathcal{I}$ and $\mathcal{O}$ are the environment actions (inputs) and system actions (outputs), respectively), and the transitions labeled with $\mathcal{I}$ and $\mathcal{O}$ alternate. To measure the excessive restriction on the environment, we project out the $\mathcal{O}$ symbols (as we are not interested in correctness) and then compute the d_{cov} distance between the system and the implementation. We demonstrate that this method of measuring environment restriction by computing the distances for a *request-grant* system.

Consider the specification S_1 and the implementations I_n in the Fig. 6. All these systems are built so that every request r is granted by g in the same step or in the next step. However, if cancel c is high, there should be no grant in that step. These requirements mandatorily forbid some environment behaviors, like the behavior with both r and c high all the time. The specification S_1 restricts the environment most permissively: for every request r, cancel c, which is low in the current or the following step. Implementations I_1, I_2, I_3 and I_4 restrict the environment to various amounts by allowing no c's, allowing no c's for the relevant two steps, allowing no c's for the current step, and allowing no c's for the following step, respectively. The restrictiveness values (d_{cov}) are summarized in Table 3 and reflect the intuitive notion that I_1 is the most restrictive, followed by I_2 and I_3, and then by the unrestrictive I_4.

6. Conclusion

We have motivated the notion of distance between systems, and introduced quantitative simulation games as a framework for measuring such distances. We presented three distances—two for quantifying aspects of correct systems, namely coverage and robustness and one for measuring the degree of correctness of an incorrect system.

There are several possible directions for future work. An interesting question is how to synthesize a system that minimizes a distance from a given specification—for example, given a specification, one might be interested in synthesizing the most robust system. Further possibilities include building a tool for measuring the robustness distance for programs or protocols implementing various error recovery or error correction mechanisms.

References

[1] R. Alur, T. Henzinger, O. Kupferman, M. Vardi, Alternating refinement relations, in: CONCUR, 1998, pp. 163–178.
[2] R. Bloem, K. Chatterjee, T. Henzinger, B. Jobstmann, Better quality in synthesis through quantitative objectives, in: CAV, 2009, pp. 140–156.
[3] B. Bloom, Ready simulation, bisimulation, and the semantics of CCS-like languages, Ph.D. Thesis, MIT, 1989.

[4] P. Caspi, A. Benveniste, Toward an approximation theory for computerised control, in: EMSOFT, 2002, pp. 294–304.
[5] P. Černý, T. Henzinger, A. Radhakrishna, Simulation distances, in: CONCUR, 2010, pp. 253–268.
[6] K. Chatterjee, L. Doyen, T. Henzinger, Expressiveness and closure properties for quantitative languages, in: LICS, 2009, pp. 199–208.
[7] K. Chatterjee, T.A. Henzinger, M. Jurdzinski, Mean-payoff parity games, in: LICS, 2005, pp. 178–187.
[8] L. de Alfaro, M. Faella, M. Stoelinga, Linear and branching system metrics, IEEE Trans. Software Eng. 35 (2) (2009) 258–273.
[9] L. de Alfaro, T. Henzinger, R. Majumdar, Discounting the future in systems theory, in: ICALP, 2003, pp. 1022–1037.
[10] L. de Alfaro, R. Majumdar, V. Raman, M. Stoelinga, Game refinement relations and metrics, Log. Methods Comput. Sci. 4 (3) (2008).
[11] J. Desharnais, V. Gupta, R. Jagadeesan, P. Panangaden, Metrics for labelled Markov processes, Theoret. Comput. Sci. 318 (3) (2004) 323–354.
[12] M. Droste, P. Gastin, Weighted automata and weighted logics, Theoret. Comput. Sci. 380 (1–2) (2007) 69–86.
[13] N. Fenton, Software Metrics: A Rigorous and Practical Approach, Revised (Paperback), Course Technology, 1998.
[14] Y. Gurevich, L. Harrington, Trees, automata, and games, in: STOC, 1982, pp. 60–65.
[15] R.W. Hamming, Error detecting and error correcting codes, Bell System Tech. J. 29 (1950) 147–160.
[16] T.A. Henzinger, O. Kupferman, S.K. Rajamani, Fair simulation, in: Information and Computation, 1997, pp. 273–287.
[17] R. Lincke, J. Lundberg, W. Löwe, Comparing software metrics tools, in: ISSTA, 2008, pp. 131–142.
[18] R.E. Lyons, W. Vanderkulk, The use of triple-modular redundancy to improve computer reliability, IBM J. Res. Dev. 6 (2) (1962) 200–209.
[19] R. Milner, An algebraic definition of simulation between programs, in: IJCAI, 1971, pp. 481–489.
[20] F. van Breugel, An introduction to metric semantics: operational and denotational models for programming and specification languages, Theoret. Comput. Sci. 258 (1–2) (2001) 1–98.
[21] F. van Breugel, J. Worrell, Approximating and computing behavioural distances in probabilistic transition systems, Theoret. Comp. Sci. 360 (1–3) (2006) 373–385.
[22] U. Zwick, M. Paterson, The complexity of mean payoff games on graphs, Theoret. Comput. Sci. 158 (1–2) (1996) 343–359.