Quantitative Simulation Games

Pavol Černý, Thomas A. Henzinger, and Arjun Radhakrishna

IST Austria (Institute of Science and Technology, Austria)

Abstract. Classical formalizations of systems and properties are boolean: given a system and a property, the property is either true or false of the system. The classical view partitions the world into "correct" and "incorrect" systems, offering few nuances. In reality, of several systems that satisfy a property in the boolean sense, often some are more desirable than others, and of the many systems that violate a property, usually some are less objectionable than others. For instance, among the systems that satisfy the response property that every request be granted, we may prefer systems that grant requests quickly (the quicker, the better), or we may prefer systems that issue few unnecessary grants (the fewer, the better); and among the systems that violate the response property, we may prefer systems that serve many initial requests (the more, the better), or we may prefer systems that serve many requests in the long run (the greater the fraction of served to unserved requests, the better).
Formally, while a boolean notion of correctness is given by a preorder on systems and properties, a quantitative notion of correctness is defined by a distance function on systems and properties, where the distance between a system and a property provides a measure of "fit" or "desirability." In this article, we explore several ways how the simulation preorder can be generalized to a distance function. This is done by equipping the classical simulation game between a system and a property with quantitative objectives. In particular, for systems that satisfy a property, a quantitative simulation game can measure the "robustness" of the satisfaction: how much can the system deviate from its nominal behavior while still satisfying the property? For systems that violate a property, a quantitative simulation game can measure the "seriousness" of the violation: how much does the property have to be modified so that it is satisfied by the system?

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 is a function that given two systems returns the distance between them.

Let us consider the definition of simulation by a two-player game, where the goal of Player 1 is to prove that simulation does not hold, whereas the goal of Player 2 is to prove that there exists a simulation relation. In order to extend this definition to capture how "good" (or how "bad") the simulation is, we make each of the players pay a certain price for his or her choices. More precisely, if we consider weighted transition systems, that is transition systems with rational weights associated with edges, then each play of the game where first player plays on a weighted transition system S_1 and the second player plays on a weighted transition system S_2 gives rise to a sequence of values. An objective function returns a value for each such sequence. We consider the limit average as the objective function. The definition of the game includes a parameter indicating which player will try to maximize the value of the play and which player will try to minimize it. The distance between the two systems is then defined as the value of the game for the maximizing player. We generalize this framework further by considering quantitative alternating simulation games, which are necessary for defining one of the distance functions we consider. We consider purely finite state transition systems. A weighted transition system S is obtained from a finite state transition system T using a *modification function* that adds weights to the edges of the system, and possibly adds new edges with weights. A distance function is defined by a pair of modification functions that are applied to finite state transition systems between which the distance is measured.

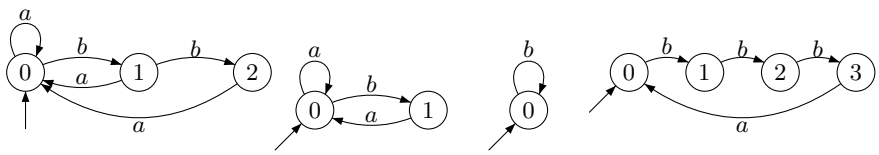
The framework is illustrated using three distance functions measuring the degree of correctness, the degree to which an implementation covers all behaviors allowed by its specification, and the degree of robustness. For incorrect implementations of reactive systems, that is those for which the specification S (represented as a finite-state machine) does not simulate the implementation I (represented as a finite state machine as well), we might be interested in how often the error occurs on average. The system I is left unmodified, while the specification S is allowed to "cheat". This is achieved using a modification scheme that gives the edges from the original system a weight of 0, and adds new "cheating" edges with a non-zero positive weight. 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 needs to cheat on average, that is, how often on average the implementation commits an error. This distance function is called *correctness*.

Let us consider an example in Figure 1. We take the system S_1 in part (a) 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 will thus

not be able to simulate either I_3 or I_4, but it is intuitively clear that 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 transition, but outputing a different symbol. When simulating the system I_3, the specification S_1 will have to output a b when taking a transition 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 therefore be $1/4$.

Considering the implementation I_2 from the Figure 1, it is easy to see that it is correct with resepct to the specifcation S_1. The correctness distance would thus be 0. However, it is also easy to see that I_2 is not perfect in oter respects: it does not include all behaviors allowed by S_1. The second distance function, *coverage*, is the dual of the correctness distance. For correct systems, we are interested in how much of the behavior allowed by the specification is actually implemented by the implementation. This distance is obtained as the value for the simulator of a game in which implementation I is required to simulate the specification S, with the implementation being allowed to cheat. The third distance function is called *robustness*. It measures how robust a particular implementation I is with respect to its specification S in the following sense: we require that even if the implementation makes an unexpected error, the resulting behavior is still captured by the specification. The unexpected error is for example an error caused by a hardware problem. It is modeled by the fact that a transition not specified by the transition relation is executed. The robustness distance is captured by a game where the implementation can cheat only if it is allowed to do so by the specification. It is in this case where quantitative alternating simulation games are needed.

Finally, we present two case studies. In the first, we consider robustness of error correction systems for transmitting data over noisy channels. Three implementations are considered: one based on the Hamming code, one based on triple modular redundancy, and an implementation without any error correction. In the second case study, 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.



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

Fig. 1. Example Systems

Related work Weighted automata [5, 6, 2, 9] provide a way to assign values to words, and to languages defined by finite-state systems. In contrast, we propose measuring distances between systems, and our approach provides distances on systems which cannot be obtained by calculating a measure for the two systems in question separately.

It has been noted in literature [8, 17] that boolean notions of correctness may not be suitable for systems that are inherently quantitative, such as real-time systems, or probabilistic systems. Metrics on such quantitative systems have been proposed [8, 4, 17, 18]. However, the systems we consider are purely finite-state, the quantitative aspect of the distance function arises only from the comparison of the behavior of the two systems between which we measure the distance.

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 [7, 11, 13]). These functions measure properties of the source code, and are fundamentally different from our distance function that capture the behavior of (reactive) programs as given by their operational semantics.

2 Quantitative Simulation Games

In Section 2.1 we define turn-based games with qualitative and quantitative objectives. In Section 2.2 we motivate and define quantitative simulation games and show their use in measuring properties and distances between systems.

2.1 Games, Objectives and Systems

Transition Systems. A *transition system* is a tuple $\langle S, \Sigma, E, s_0 \rangle$ where Σ is a finite alphabet, S is a finite set of states, $E \subseteq S \times \Sigma \times S$ is a set of transitions between the states, and s_0 is the initial state. The set of all transition systems is denoted by $\mathcal{S}$. A *weighted transition system* is a tuple $\langle S, \Sigma, E, s_0, v \rangle$ where S, Σ, E and s_0 are as before, and v is a function from E to $\mathbb{Q}$. The set of all weighted transition systems is denoted by $\mathcal{S}_Q$.

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) form a partition of S. Game graphs model systems where each choice is made by one of the two competing agents called players. The choice of the next state is made by Player 1 (Player 2) when the current state is in S_1 (S_2). A *weighted game graph* is a game graph along with a weight function v which assigns values from $\mathbb{Q}$ to transitions.

When the two competing agents represent the choices internal to a system, we call the game graph an *alternating transition system*. We only consider alternating transition systems where the states of Player 1 and Player 2 alternate. The set of all alternating transition systems is denoted by $\mathcal{A}$. The set of all weighted alternating transition systems is denoted by $\mathcal{A}_Q$.

A play in the game graph G is a path $\rho = \rho_0\rho_1\rho_2 \ldots \in S^\omega$, an infinite sequence where for all $i \geq 0$, we have $(\rho_i, \sigma, \rho_{i+1}) \in E$ for some $\sigma \in \Sigma$. The set of all plays is denoted by Ω.

Strategies. Given a game graph G, a *strategy* for Player 1 is a function $\pi : S^*S_1 \to S$ such that $\forall s_0 s_1 \ldots s_i \in S^*S_1$, we have that $(s_i, \pi(s_0 s_1 \ldots s_i)) \in E$. A strategy for Player 2 is defined in a similar way. The set of all strategies for Player i is denoted by Π_i. A play $\rho = \rho_0\rho_1\rho_2 \ldots$ *conforms* to a player p strategy π, if $\forall i \geq 0 : \rho_i \in S_p \implies \rho_{i+1} = \pi(\rho_0\rho_1 \ldots \rho_i)$. The *outcome* of a Player 1 strategy π_1 and a Player 2 strategy π_2 is the unique play which conforms to both π_1 and π_2. The outcome of the two strategies is represented as $out(\pi_1, \pi_2)$.

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 all the necessary information about the history can be summarized by a finite amount of information. Formally, a strategy π is called:

1. *Memoryless* if $\pi(w_1 s) = \pi(w_2 s)$ for all $w_1, w_2 \in S^*$ and $s \in S$.
2. *Finite memory* if there exists a finite set M, an initial memory state $m_0 \in M$, a memory update function $\mu : S^* \times M \to M$ and move function $\nu : S \times M \to S$ such that
   (a) $\mu(ws, m_0) = \mu(s, \mu(w, m_0))$, and
   (b) $\pi(ws) = \nu(s, \mu(ws, m_0))$ for all $w \in S^*$ and $s \in S$.

Objectives. A *boolean objective* is a function $\Phi : \Omega \to \{0, 1\}$. The goal of Player 1 in a game is to ensure that the play maps to 1; and the goal of Player 2 is to ensure that the play maps to 0. A *quantitative objective* is a *value* function $f : \Omega \to \mathbb{R}$. 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 if $\Phi(\rho) = 1$. Otherwise, it is winning for Player 2. A strategy π is called a *winning strategy* for Player p if every play starting at the initial state and 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. $\inf_{\pi_2 \in \Pi_2} f(out(\pi_1, \pi_2))$. 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 π is an *optimal strategy* for a player if the value of the strategy for the player is equal to the value of the game.

We will only be considering the following quantitative objective. Given a game graph with the weight function v and a play $\rho = \rho_0\rho_1\rho_2 \ldots$ in the game graph, let $v_i = v((\rho_0, \rho_1))$:

$$LimAvg(\rho) = \liminf_{n \to \infty} \frac{1}{n} . \sum_{i=0}^{n-1} v_i$$

We make a note here that for *LimAvg* objectives, optimal memoryless strategies exist for both players [10].

2.2 Qualitative Simulation Games

The classical simulation preorder [15] is a useful and polynomially computable relation to compare two transition systems. In [1] this relation was extended as alternating simulation to alternating transition systems. These relations can be cast in the form of 2-player games with qualitative objectives.

Simulation and Alternating simulation. Consider two transition systems $A_1 = \langle S_{A_1}, \Sigma, E_{A_1}, s_{0,A_1} \rangle$ and $A_2 = \langle S_{A_2}, \Sigma, E_{A_2}, s_{0,A_2} \rangle$. A_2 *simulates* A_1 if there exists a relation $H \subseteq S_{A_1} \times S_{A_2}$ such that:

1. $(s_{0,A_1}, s_{0,A_2}) \in H$, and
2. $(s, s') \in H \Rightarrow \forall \sigma, t : (s, \sigma, t) \in T_{A_1} \, [\exists t' : (s', \sigma, t') \in T_{A_2} \wedge (t, t') \in H]$.

This is written as $A_1 \leq A_2$.

For two alternating transition systems $A_1 = \langle S_{A_1}, S_{1,A_1}, S_{2,A_1}, \Sigma, E_{A_1}, s_{0,A_1} \rangle$ and $A_2 = \langle S_{A_2}, S_{1,A_2}, S_{2,A_2}, \Sigma, E_{A_2}, s_{0,A_2} \rangle$, *alternating simulation* of A_1 by A_2 holds if there exists a relation $H \subseteq S_{A_1} \times S_{A_2}$ such that:

1. $(s_{0,A_1}, s_{0,A_2}) \in H$, and
2. $(s, s') \in H \Rightarrow (s \in S_{1,A_1} \Leftrightarrow s' \in S_{1,A_2})$
3. $(s, s') \in H \wedge s \in S_{1,A_1} \Rightarrow \forall (s, \sigma, t) \in T_{A_1} : [\exists (s', \sigma, t') \in T_{A_2} : (t, t') \in H]$.
4. $(s, s') \in H \wedge s \in S_{2,A_1} \Rightarrow \exists (s', \sigma, t') \in T_{A_2} : [\forall (s, \sigma, t) \in T_{A_1} : (t, t') \in H]$.

This is written as $A_1 \leq_{\Box} A_2$. The definition above is a restricted form of alternating simulation introduced in [1]. The notion used here is sufficient for the kind of alternating transition systems we are interested in as they have only two agents and are turn-based.

Simulation and Alternating Simulation Games. Both the above notions of simulation can be cast in the form of two player turn-based games of the form introduced in Section 2.1. Given two (alternating) transition systems, A_1 and A_2, we can construct a game $\mathcal{G}_{A_1,A_2}$ such that, $A_1 \leq A_2$ ($A_1 \leq_{\Box} A_2$) if and only if Player 2 has a winning strategy in $\mathcal{G}_{A_1,A_2}$.

Quantitative Simulation Game Graphs. To construct simulation and alternating simulation games, 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 in a later section to define quantitative simulation games.

Given two quantitative transition systems, $A_1 = \langle S_{A_1}, \Sigma, E_{A_1}, s_{0,A_1}, v_{A_1} \rangle$ and $A_2 = \langle S_{A_2}, \Sigma, E_{A_2}, s_{0,A_2}, v_{A_2} \rangle$ with the same alphabet, we define the corresponding *quantitative simulation game graph* as follows:

1. The alphabet is the same as the alphabet of A_1 and A_2.

2. The state space of the game graph is the product of the state spaces of the systems and the alphabet. Formally, $S = S_{A_1} \times (\Sigma \cup \{\#\}) \times S_{A_2}$.
3. The states are partitioned into Player 1 and Player 2 states as follows: $(s_1, \#, s_2) \in S_1$ and $(s_1, \sigma, s_2) \in S_2$.
4. The initial state is $s_0 = (s_{0,A_1}, \#, s_{0,A_2})$.
5. Each transition of the game graph corresponds to a transition in either A_1 or A_2 as follows:
   (a) $((s_1, \#, s_2), \sigma, (s_1', \sigma, s_2)) \in E \Leftrightarrow (s_1, \sigma, s_1') \in E_{A_1}$
   (b) $((s_1, \sigma, s_2), \sigma, (s_1, \#, s_2')) \in E \Leftrightarrow (s_2, \sigma, s_2') \in E_{A_2}$
   For each of the above transitions, the weight is the same as the corresponding transition in A_1 or A_2.

For the 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, Player 2 wins. If the play arrives at a state where there is no possible transition, Player 1 wins. We denote this classical simulation game as $G_{\text{sim}}^{A_1,A_2}$.

Intuitively, in every state, Player 1 chooses a transition of A_1 and Player 2 has to simulate it by picking a transition of A_2. If Player 2 cannot simulate at some point, Player 1 wins that play. It is easy to see that if A_2 simulates A_1, there is a winning strategy for Player 2 and vice versa.

We can extend the simulation game to alternating simulation games as follows: Given two quantitative alternating transition systems $A_1 = \langle S_{A_1}, S_{1,A_1}, S_{2,A_1}, \Sigma, E_{A_1}, s_{0,A_1}, v_{A_1} \rangle$ and $A_2 = \langle S_{A_2}, S_{1,A_2}, S_{2,A_2}, \Sigma, E_{A_2}, s_{0,A_2}, v_{A_2} \rangle$, we define the corresponding *alternating simulation game graph* as:

1. Player 1 states of the graph are $S_1 = \{(s_1, \#, s_2, 1) \mid s_1 \in S_{1,A_1} \wedge s_2 \in S_{1,A_2}\} \cup \{(s_1, \sigma, s_2, 1) \mid s_1 \in S_{2,A_1} \wedge s_2 \in S_{1,A_2} \wedge \sigma \in \Sigma\}$. The first set of the union represents the states where Player 1 has to choose a transition for Player 2 to simulate and the second set represents the states where Player 2 has already chosen a transition with the symbol σ and Player 1 has to simulate it.
2. Player 2 states of the graph are $S_2 = \{(s_1, \#, s_2, 2) \mid s_1 \in S_{2,A_1} \wedge s_2 \in S_{2,A_2}\} \cup \{(s_1, \sigma, s_2, 2) \mid s_1 \in S_{2,A_1} \wedge s_2 \in S_{1,A_2} \wedge \sigma \in \Sigma\}$. The sets in this union are analogous to the ones in Player 1 states.
3. The alphabet is the same as the alphabet of A_1 and A_2. The initial state is $(s_{0,A_1}, \#, s_{0,A_2}, p)$ where p depends on whether s_{0,A_1} and s_{0,A_2} are Player 1 or Player 2 states. Note that if one of them is a Player 1 state and the other is a Player 2 state, alternating simulation of A_1 by A_2 cannot hold and hence, we do not define the game graph for such cases.
4. The transitions again correspond to A_1 or A_2 transitions as follows:
   (a) If (s_1, σ, s_1') is a transition in A_1 and $(s_1, \#, s_2, 1)$ is a Player 1 state, we have the corresponding transition $((s_1, \#, s_2, 1), \sigma, (s_1', \sigma, s_2, 2))$ in E, i.e. in states where Player 1 has to choose a transition of A_1, the A_1 component of the state is changed to the destination of the A_1 transition and the symbol is changed to the symbol of the A_1 transition.

(b) If (s_2, σ, s_2') is a transition in A_2 and $(s_1, \#, s_2, 2)$ is a Player 2 state, we have the corresponding transition $((s_1, \#, s_2, 2), \sigma, (s_1, \sigma, s_2', 1))$ in E. These transitions are similar to the previous case, but Player 2 has to choose a A_2 transition for Player 1 to simulate.

(c) If (s_1, σ, s_1') is a transition in A_1 and $(s_1, \sigma, s_2, 1)$ is a Player 1 state, we have the corresponding transition $((s_1, \sigma, s_2, 1), \sigma, (s_1', \#, s_2, 1))$ in E. Here, Player 1 chooses a transition to simulate the previous move of Player 2 which had the symbol σ. The A_1 component of the state is changed accordingly and the symbol is reset to #.

(d) If (s_2, σ, s_2') is a transition in A_2 and $(s_1, \sigma, s_2, 2)$ is a Player 2 state, we have the corresponding transition $((s_1, \sigma, s_2, 2), \sigma, (s_1, \#, s_2', 2))$ in E. This is the dual of the previous case.

The weight of each transition is equal to the corresponding A_1 or A_2 transition.

We consider the game graph without weights to define the alternating simulation game. As in the case of simulation games, the objective of the Player 2 is to ensure that the play proceeds ad infinitum and the objective of Player 1 is to ensure that the play stops in a state where there in no possible transition from it. We denote this qualitative alternating simulation game as $G_{\text{asim}}^{A_1,A_2}$.

Intuitively, as in the simulation game, whenever in a Player 1 state of A_1, Player 1 chooses a transition and Player 2 has to simulate it and in A_2. But, in a Player 2 state of A_2, Player 2 chooses a transition of A_2 and Player 1 chooses a transition to simulate it in A_1. Player 1 wins if a transition cannot be simulated at some point and Player 2 wins otherwise. Again, it can be seen that alternating simulation of A_1 by A_2 holds, there exists a winning strategy for Player 2 and vice versa.

2.3 Quantitative Simulation Games.

We now define a generalized notion of simulation games called quantitative simulation games. We replace the qualitative objectives of a simulation game by a quantitative objective to measure distances between systems.

Quantitative Simulation Games. Given two quantitative transition systems A_1 and A_2, the *quantitative simulation game* is played on the quantitative simulation game graph with the objective of Player 1 being to maximize the *LimAvg* of the play while the objective of Player 2 being to minimize it. If the play stops at a state with no possible outgoing transition, the value of the game is set to be 1. We denote this game as $G_{\text{qsim}}^{A_1,A_2}$.

Quantitative Alternating Simulation Games. Given two quantitative alternating transition systems, a *quantitative alternating simulation game* is played on the alternating simulation game graph of the two systems G^{A_1,A_2} where the objective of Player 1 is to maximize the *LimAvg* of the play whereas the objective of Player 2 is to minimize it. If the play stops in a state where no transitions are possible, the value of that play is defined to be 1. We denote this game as $G_{\text{qasim}}^{A_1,A_2}$

Intuitively, from a Player 1 state in A_1, Player 1 chooses a transition and Player 2 has to simulate in A_2 and in every Player 2 state in A_2, Player 2 chooses a transition and Player 1 has to simulate it. If the simulation cannot continue at some point, we have that Player 1 wins and get the maximal value of 1.

We calculate the distance between two systems according to a quantitative simulation game as the value of the game for the Player 1 . In Section 3, we use this notion of distance to compute various properties of systems.

2.4 Modification Schemes.

We will use quantitative simulation games to measure various properties of systems. For computing these properties, we would like to use small modifications of the original systems. For example, when trying to compute the distance as the number of errors an implementation commits with respect to a specification, we would like to add to the specification some recovery behaviour to be used in case of error. To ensure that the specification does not use this recovery behaviour when not necessary, there will be an extra cost for using it. We encode these kind of modifications using the notion of modification schemes. However, to ensure that modifications schemes do not change the system too much, we impose a strict set of rules on these functions.

A *modification scheme* is a function $m : \mathcal{S} \to \mathcal{S}_Q \cup \mathcal{A}_Q$ which can be computed using the following steps:

1. Edges may be added to the transition system without changing the state space.
2. Each state is replaced with a local subgraph. The graph is the same for all states of the system. All the edges of the graph, including those obtained from the previous step, have to be preserved.
3. Every edge of the system is associated with a weight from $\mathbb{Q}$.

Formally, a modification scheme is a triple $\langle e, G_S, v\rangle$ where $e : \mathcal{S} \to 2^{S\times\Sigma\times S}$ is a function that provides the extra edges to be added, G_S is a (game) graph which we will use to replace each state, and v is a weight function.

The above rules ensure that the modified system retains the structure of the original system. We present three examples of modification schemes.

Output Recovery. This scheme is used to add recovery behavior to a system that will allow 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 output symbols are added to the system, i.e. $\{(s, \alpha, s') \mid \alpha \in \Sigma\}$. However, these recovery transitions are given an extra cost to prohibit their free use. Formally, let $T = \langle S, \Sigma, E, s_0\rangle$. $\mathit{OutputRecovery} = \langle e, G_S, v\rangle$ where $\mathit{OutputRecovery}(T)$ is the quantitative transition system obtained by:

1. The transitions which differ from existing transitions only in the symbol added to the system, i.e. $e(\langle S, \Sigma, E, s_0\rangle) = \{(s, \alpha, t) \mid (s, \sigma, t) \in E\}$.
2. The states are replaced by the trivial graph with one state, so as to preserve the system obtained from the previous step, i.e. $G_S = \langle\{s\}, \Sigma, \emptyset, s\rangle$, and

3. Weights are assigned to transitions as follows: All transitions in the original system are given weight zero and the newly added transitions are given weight 2.

The weight used for the error edges is 2 instead of 1 so that the values computed for distances in Section 3 are normalized to 1.

In-place Recovery. This scheme used similarly as the *OutputRecovery*, i.e. to allow systems to compensate for an error. However, the kind of recovery behavior is different. In this case, a system is allowed to take a spurious transition without changing its state and output an arbitrary symbol. Formally, given $T = \langle S, \Sigma, E, s_0 \rangle$ and $\mathit{InPlaceRecovery} = \langle e, G_S, v \rangle$, we define $\mathit{InPlaceRecovery}(T)$ to be the quantitative transition system obtained by:

1. Self loops on each state are added for every symbol, i.e. $e(\langle S, \Sigma, E, s_0 \rangle) = \{(s, \alpha, s) \mid \alpha \in \Sigma\}$.
2. The states are replaced with a trivial graph as before, and
3. The edges present in the original system have a weight of 2 and the other edges have a weight of 0.

Controlled Error. In a perfectly functioning system, errors may occur due to unpredictable events like cosmic rays. We model this with an alternating transition system with one agent modeling the original system (Player 1) and one modeling the controlled error (Player 2). At every state, Player 2 chooses whether or not a 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. Therefore, Player 2 controls the possibility of an error occuring, whereas Player 1 actually chooses the transition the system takes. We penalize Player 2 for the choice of not allowing errors to happen.

The nature of errors that can occur can be taken as a parameter to the modification scheme. For a transition system T with transitions E, we will consider the following kind of errors transitions specified by $\mathit{Err}(T) = \{(s, \alpha, s') \mid (s, \sigma, s') \in E \wedge \alpha \in \Sigma\}$. The modification scheme with errors specified by a function $X : \mathcal{S} \to S \times \Sigma \times S$ is denoted by $\mathit{ContErr}_X$. Without a subscript, it is taken to be $\mathit{ContErr}_{\mathrm{Err}}$. We also use a variant of this scheme where there are no error transitions and denote it as $\mathit{ContErr}_\emptyset$.

Given $T = \langle S, \Sigma, E, s_0 \rangle$ we define $\mathit{ContErr}(T)$ to be the quantitative alternating transition system obtained by:

1. The transitions which differ from existing transitions only in the symbol are added to the system as in *OutputRecovery*.
2. The graph G_S used to replace each state s is shown in Figure 2.
3. Weights are assigned to transitions as follows:

$$v((s, \sigma, t)) = \begin{cases} 0 & \text{if } \sigma \neq \neg c \\ 2 & \text{if } \sigma = \neg c \end{cases}$$

In addition to these modification schemes, we define the trivial modification scheme where no changes are made to the transitions of the system and edge is given the weight 0. Let us call this scheme *NoMod*.

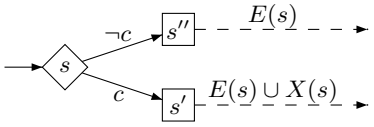


Fig. 2. Graph for *ContErr*

3 Distances

In this section we consider examples of distances which can be captured by the notion of quantitative simulation games defined in the Section 2. The first two presented here are modifications of the classical simulation game in which the simulation relation is not perfect. The games are set up in such a way that the amount of imperfection in the simulation measures a property of the relation between the systems. The third game presented measures the robustness of a system in a novel way.

The correctness and coverage distances presented here could also have been formulated as the maximum difference in the values assigned to a behavior by weighted automata as in [2]. However, the complexity of computation of such distances is not known [5, 6]. The problem is the quantitative analogue of boolean language inclusion. Our solution of using quantitative version of simulation games corresponds to the choice of using the simulation preorder instead of language inclusion as it has much better complexity.

3.1 Correctness

The boolean simulation relation between systems can determine whether the behavior of one system can be simulated by another. However, this 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 can measure the worst-case number of errors, i.e. the worst-case number of times the specification system has to recover by using transitions which are not allowed.

Definition 3.1 (Correctness distance)**.** *We define the* correctness distance *from transition system T_1 to transition system T_2 to be the Player 1 value of the quantitative simulation game $G_{qsim}^{NoMod(T_1),OutputRecovery(T_2)}$ between $NoMod(T_1)$ and $OutputRecovery(T_2)$. We denote this distance as $d_{cor}(T_1, T_2)$. We denote the corresponding game as $\mathcal{G}_{cor}^{T_1,T_2}$.*

The game $\mathcal{G}_{\text{cor}}$ can be intuitively understood as follows: Given two transition systems T_1 and T_2, we are trying to simulate the system T_1 by T_2 with possible errors. Let us consider T_1 as an implementation of the specification T_2. Now, every move by Player 1 is equivalent to choosing transition of T_1. Every zero weight move of Player 2 is a transition of a T_2. Whenever the implementation

commits an error, we have that Player 2 cannot simulate the Player 1 move with a zero weight transition and has to use a non-zero weight transition to recover. Player 2 tries to minimize the use of non-zero weight transitions. The value of the game is the *LimAvg* of the number of errors or recoveries committed. Player 2 tries to show that the number of errors of T_1 is as small as possible for all strategies of Player 1 , i.e. all behaviors of the implementation.

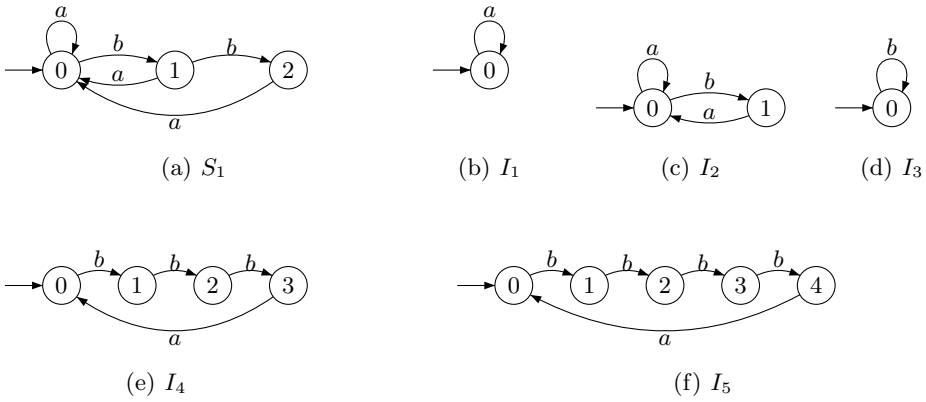


Fig. 3. Example Systems

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 Figure 3, S_1 is the specification system against which we want to measure the rest of the systems. In this case, the specification says that there cannot be more than two b's in a row. The distances between these systems according to the correctness game are summarized in Table 1.

T_1	T_2	$d_{\text{cor}}(T_1, T_2)$
S_1	S_1	0
S_1	I_1	0
S_1	I_2	0
S_1	I_3	1/3
S_1	I_4	1/4
S_1	I_5	1/5

Table 1. Distances according to Correctness game

From the table, we can see that the systems in which all behaviors satisfy the specification have a distance of zero to it. Among the systems which do not satisfy the specification, i.e. I_3 and I_4, we can intuitively see that I_3 is worse

than I_4 in the sense that I_3 violates the specification that there are no more than two b's in a row more often than I_4. This fact is reflected in the distances as I_3 is more distant from S_1 than I_4. However, surprisingly the distance to I_5 is less than the distance to I_4. In fact, the distances reflect on the long run the number of times the specification has to err to simulate the implementation.

We now examine the properties of d_{cor} and prove that it is a *directed metric* as defined in [8]. Directed metrics are a natural quantitative extension of preorders where the reflexivity is replaced by a zero property and the transitivity is replaced by the triangle inequality.

Proposition 3.2. *d_{cor} is a directed metric, i.e. :*

1. $\forall S \in \mathcal{S} : d_{cor}(S, S) = 0$
2. $\forall S_1, S_2, S_3 \in \mathcal{S} : d_{cor}(S_1, S_3) \leq d_{cor}(S_1, S_2) + d_{cor}(S_2, S_3)$

Proof: We will first prove the triangle inequality. Let us denote by $\mathcal{G}^{i,j}_{\text{cor}}$ the game for computing $d_{\text{cor}}(S_i, S_j)$. Let τ_2 and τ_3 be optimal memoryless strategies for Player 2 in $\mathcal{G}^{1,2}_{\text{cor}}$ and $\mathcal{G}^{2,3}_{\text{cor}}$ respectively. Now, we construct a finite memory strategy τ^* for Player 2 in $\mathcal{G}^{1,3}_{\text{cor}}$ which will give us the required inequality. We will use the state space of S_2 as the memory for τ^*.

The strategy τ^* works as follows:

1. Let Player 1 chooses to move according to the S_1 transition (s_1, σ_1, s'_1) at the game state $(s_1, \#, s_3)$.
2. The memory is updated from the state s_2 to s'_2 where $(s'_1, \#, s'_2)$ is the successor chosen by τ_2 when at $\mathcal{G}^{1,2}_{\text{cor}}$ position (s'_1, σ_1, s_2). Let the corresponding $\mathit{OutputRecovery}(S_2)$ transition be (s_2, σ_1, s'_2).
3. Now, τ^* chooses the successor $(s'_1, \#, s'_3)$ where $(s'_2, \#, s'_3)$ is the successor chosen by τ_3 at the $\mathcal{G}^{2,3}_{\text{cor}}$ state (s'_2, σ_1, s_3). Let the corresponding $\mathit{OutputRecovery}(S_3)$ transition be (s_3, σ_1, s'_3). (Note that we may not be able to replicate the transition (s_2, σ_2, s'_2) in $\mathcal{G}^{2,3}_{\text{cor}}$ as it may be a transition that exists only in $\mathit{OutputRecovery}(S_2)$. However, this is not a problem as S_2 will contain a transition (s_2, σ'_2, s'_2) which can be used instead in $\mathcal{G}^{2,3}_{\text{cor}}$.)

We make the following observation: 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 memoryless Player 1 strategy σ. We consider only memoryless strategies as optimal memoryless strategies exist for each player in limit average games. 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 $\mathit{OutputRecovery}(S_2)$ transition (s_2, σ_1, s'_2), an S_2 transition (s_2, σ'_2, s'_2) and an

OutputRecovery(S_3) transition (s_3, σ_1, s_3'). We depict the situation as follows:

$$\left.\begin{array}{l} \left.\begin{array}{l} s_{1,0} \xrightarrow{\sigma_1(v_{1,0})} s_{1,1} \xrightarrow{\sigma_1(v_{1,1})} s_{1,2}\cdots \\ s_{2,0} \xrightarrow{\sigma_1(v_{2,0})} s_{2,1} \xrightarrow{\sigma_1(v_{2,1})} s_{2,2}\cdots \end{array}\right\}\rho_1 \\ \left.\begin{array}{l} s_{2,0} \xrightarrow{\sigma_2'(v_{2,0})} s_{2,1} \xrightarrow{\sigma_2'(v_{2,1})} s_{2,2}\cdots \\ s_{3,0} \xrightarrow{\sigma_1(v_{3,0})} s_{3,1} \xrightarrow{\sigma_1(v_{3,1})} s_{3,2}\cdots \end{array}\right\}\rho_2 \end{array}\right\}\rho$$

The play ρ in $\mathcal{G}_{\text{cor}}^{1,3}$ corresponds to the transitions in the first and the last rows. This play can be decomposed into plays ρ_1 and ρ_2 in $\mathcal{G}_{\text{cor}}^{1,2}$ and $\mathcal{G}_{\text{cor}}^{2,3}$ 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 n^{th} move in a play η by η^n. Therefore, we have,

$$\begin{aligned}
\nu(\rho) &= \liminf_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho^i) \\
&\leq \liminf_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} \big(v(\rho_1^i) + v(\rho_2^i)\big) \\
&= \lim_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} \big(v(\rho_1^i) + v(\rho_2^i)\big) \\
&= \lim_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho_1^i) + \lim_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho_2^i) \\
&= \liminf_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho_1^i) + \liminf_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n} v(\rho_2^i) \\
&\leq d_{\text{cor}}(S_1, S_2) + d_{\text{cor}}(S_2, S_3)
\end{aligned}$$

In the above set of equations we use limit and limit infimum interchangeably. This can be done because of the following reason: as all the strategies we are considering are either memoryless or finite memory, each play and hence, each sequence of weights is ultimately repeating. Hence, the limit infimum of the average of such a sequence is the limit of the average of the sequence as the limit of the average converges to the average weight of the repeating sequence.

We have proved that for every memoryless Player 1 strategy, the value of the game is less than $d_{\text{cor}}(S_1, S_2) + d_{\text{cor}}(S_2, S_3)$. To prove that $d_{\text{cor}}(S, S) = 0$ is very simple. It can be shown by construction of a Player 2 strategy that copies every Player 1 move. Hence, we have the result. ■

3.2 Coverage

Now, we present the dual game of the one presented above. Here, we are trying to measure the behaviors that are there in one system that are not present

in the other system. Given a specification and implementation, the coverage distance corresponds to the behavior of the specification which is farthest from any behaviour of the implementation. Hence, we have that the coverage distance from a system T_1 to a system T_2 is the correctness distance from T_2 to T_1.

Definition 3.3 (Coverage distance)**.** *We define the* coverage distance *from transition system T_1 to transition system T_2 to be the Player 1 value of the quantitative simulation game $G_{qsim}^{NoMod(T_2),OutputRecovery(T_1)}$ between $NoMod(T_2)$ and $OutputRecovery(T_1)$. We denote this distance as $d_{cov}(T_1, T_2)$. We denote the corresponding game as $\mathcal{G}_{cov}^{T_1,T_2}$.*

$\mathcal{G}_{\text{cov}}$ measures the distance between T_1 and T_2 as the worst-case number of recoveries from errors that have to be committed by T_1 to cover all the behaviors of T_2. This can intuitively be understood as the farthest behavior of specification T_2 which is not implemented by T_1, i.e. a measure of the behaviors of the specification that are not covered by the system.

Again, we present examples of systems and their distances according to $\mathcal{G}_{\text{cov}}$ to demonstrate that the game actually measures the distance we require. We use the systems from the examples in Figure 3. The distances are summarized in Table 2.

T_1	T_2	$d_{\text{cov}}(T_1, T_2)$
S_1	S_1	0
S_1	I_1	2/3
S_1	I_2	1/3
S_1	I_3	1
S_1	I_4	3/4
S_1	I_5	4/5

Table 2. Distances according to Coverage game

In these tables, we can see that S_1 itself implements all the behavior of S_1. But, I_1 and I_2, although correct systems, do not implement all the behaviors of S_1. The worst case for these systems is the case where S_1 outputs as many b's as possible. In that case, I_1 has to err in two-thirds of the steps whereas I_2 has to err one-third of the steps. The distances also illustrate the fact that I_2 implements more of the specification S_1 than I_1.

For the systems I_3 and I_4, the worst case behavior of S_1 is the one where it only outputs a's. In this case, I_3 has to err all the time whereas, I_4 has to err three-fourths of the steps to emulate S_1.

As before we prove properties of d_{cov} to show that it is directed metric.

Proposition 3.4. *d_{cov} is a directed metric, i.e. :*

1. $\forall S \in \mathcal{S} : d_{cov}(S, S) = 0$

2. $\forall S_1, S_2, S_3 \in \mathcal{S} : d_{cov}(S_1, S_3) \leq d_{cov}(S_1, S_2) + d_{cov}(S_2, S_3)$

Proof: The proof of this proposition follows from the fact that for any two systems S_1 and S_2, we have that $d_{\text{cov}}(S_1, S_2) = d_{\text{cor}}(S_1, S_2)$. ■

3.3 Robustness

In this section, we present a game that captures a certain notion of robustness. Suppose we have a specification system and an implementation of the specification. The notion of robustness presented here is a measure of the number of places in the long run where an error by the implementation will make it nonconformant to the specification. In short, it is the of the number of critical points, or points where an error will lead to an unacceptable behavior. Hence, the lower the value of the robustness distance, the more robust a system is.

Definition 3.5 (Robustness distance)**.** *We define the* robustness distance *from transition system T_1 to transition system T_2 to be the Player 1 value of the quantitative alternating simulation game $G_{qasim}^{ContErr(T_1), ContErr_\emptyset(T_2)}$ played between $ContErr(T_1)$ and $ContErr_\emptyset(T_2)$. We denote this distance as $d_{rob}(T_1, T_2)$. We denote the corresponding game as $\mathcal{G}_{rob}^{T_1,T_2}$.*

The game $G_{\text{qasim}}^{ContErr(T_1), ContErr_\emptyset(T_2)}$ in intuitively simple and is played in the following steps:

1. Player 2 chooses whether Player 1 is allowed to make an error on the implementation system.
2. Player 1 chooses a transition on the implementation system. She is allowed to err based on Player 2 choice in the previous step.
3. Player 2 chooses a move on the specification to simulate the implementation.

Player 2 tries to minimize the number of moves where she disallows Player 1 to commit errors (without destroying the simulation) whereas Player 1 tries to maximize it. Intuitively, the positions where Player 2 cannot allow errors are the critical points for the implementation.

We again present examples to demonstrate that the game actually measures robustness in the intuitive sense. Also, in Section 4.1, we compute the robustness of forward error correction systems according to this game. We show that the values computed according to the robustness game correspond to the error tolerance of these different systems.

Let us examine a few of the examples in detail. 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 outputing a b. The next two moves can be b's irrespective 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 3.

T_1	T_2	$d_{\text{rob}}(T_1, T_2)$
S_1	S_1	1
S_1	I_1	1/3
S_1	I_2	2/3
S_1	I_3	1
S_1	I_4	1
S_1	I_5	1

Table 3. Distances according to Robustness game

Again, we prove some properties of d_{rob}. In this case, d_{rob} is not a directed metric because it violates the zero property. The first row of Table 3 is a witness counterexample. However, we can still prove the triangle inequality as before.

Proposition 3.6. *d_{rob} conforms to the triangle inequality, i.e. :*

$$\forall S_1, S_2, S_3 \in \mathcal{S} : d_{rob}(S_1, S_3) \leq d_{rob}(S_1, S_2) + d_{rob}(S_2, S_3)$$

Proof: We will proceed to prove this result along the same lines as the proof of Proposition 3.2 by constructing a strategy for Player 2 in $\mathcal{G}^{1,3}_{\text{rob}}$ from the optimal strategies of Player 2 in $\mathcal{G}^{1,2}_{\text{rob}}$ and $\mathcal{G}^{2,3}_{\text{rob}}$. Let the optimal strategies for Player 2 in $\mathcal{G}^{1,2}_{\text{rob}}$ and $\mathcal{G}^{2,3}_{\text{rob}}$ be τ_2 and τ_3 respectively. We construct a strategy τ^* for Player 2 in $\mathcal{G}^{1,3}_{\text{rob}}$ with the state space of S_2 as memory.

In the following description, we will denote a state of the alternating game (s_i, σ_k, s_j, p) by (s_i, s_j). Let (s_1, s_3) be the state of the game and s_2 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. In the step where Player 2 has to choose either a c or a $\neg c$ transition, τ^* chooses a c transition in state (s_1, s_3) if, either τ_2 chooses a c transition in state (s_1, s_2) or if τ_3 chooses a $\neg c$ transition in state (s_2, s_3).
2. Now, if $\neg c$ is chosen 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. Therefore, if Player 1 changes the state of S_1 from s_1 to s'_1:
   (a) Update the memory of τ^* to s'_2 where (s'_1, s'_2) is the successor chosen by τ_2 in $\mathcal{G}^{1,2}_{\text{rob}}$ at the state (s'_1, s_2).
   (b) The τ^* move in $\mathcal{G}^{1,3}_{\text{rob}}$ is to (s'_1, s'_3) where (s'_2, s'_3) is the successor chosen by τ_3 in $\mathcal{G}^{2,3}_{\text{rob}}$ at the state (s'_2, s_3).

   The symbol chosen by each strategy is the same as simulation is assumed.
3. Now, if c is chosen, 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 S_1 from s_1 (including the erroneous moves) can be simulated by 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:

(a) For every $ContErr(S_1)$ transition from s_1 to s_1' on σ, there is a non-erroneous S_1 transition between the same states (by definition). Now, we update the memory of τ^* to s_2' where (s_1', s_2') is the τ_2 move in reply to the non-erroneous transition.
(b) Now, the τ^* move is to (s_1', s_3') where (s_2', s_3') is the successor chosen by τ_3 in reply to the transition from s_2 to s_2' on σ in $\mathcal{G}_{\text{rob}}^{2,3}$.

As in the proof of Proposition 3.2, we can decompose any play of $\mathcal{G}_{\text{rob}}^{1,3}$ conforming to τ^* into two plays of $\mathcal{G}_{\text{rob}}^{1,2}$ and $\mathcal{G}_{\text{rob}}^{2,3}$ using the memory of τ^*. Also, we have the case that there is a non-zero weight move in $\mathcal{G}_{\text{rob}}^{1,3}$ if and only if there is corresponding non-zero weight move in either $\mathcal{G}_{\text{rob}}^{1,2}$ or $\mathcal{G}_{\text{rob}}^{2,3}$. 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 $\mathcal{G}_{\text{rob}}^{1,3}$ breaks. Due to the way τ^* is defined, the simulation between S_1 and S_3 breaks in $\mathcal{G}_{\text{rob}}^{1,3}$, if and only if the simulation breaks in $\mathcal{G}_{\text{rob}}^{1,2}$ or $\mathcal{G}_{\text{rob}}^{2,3}$. Hence, we have $d_{\text{rob}}^{1,3} = 1$ due to the failure of simulation if and only if $d_{\text{rob}}^{2,3} = 1$ of $d_{\text{rob}}^{1,2} = 1$, which will give us the required inequality. ■

4 Applications of Distances

In this section, we present two examples of application of the distances defined in Section 3 to measure interesting properties of larger systems. In Section 4.1, we show examine forward error correction systems for bit streams and show a relation between their robustness measured by $\mathcal{G}_{\text{rob}}$and the bit-error rate they can tolerate. In Section 4.2, we measure the coverage of a number of implementations of a request-grant system with respect to a specification and illustrate how d_{cov} measures the restriction placed on the environment by the implementations.

4.1 Forward Error Correction Systems

Forward Error Correction systems are a mechanism of error control for data transmission on noisy channels [16]. A very important characteristic of these error correction systems is the *maximum tolerable bit-error rate*, which 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} distance between a system and an ideal system (specification).

We will examine three forward error correction systems: one with no error correction facilities, the Hamming(7,4) code [12], and triple modular redundancy [14]. Intuitively, each of these systems is at a different point in the trade-off between efficiency of the 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 for transmission of messages 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 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.

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

The pseudo-code for the three systems we are examining is presented in Figure 4. The basic architecture of each system is the same. Each system has a four bit input and an encoder which adds the error correction bits to these. Then, the bits are multiplexed and transmitted over a noisy channel. The bits received on the other side of the channel are de-multiplexed and decoded and output. Each system is split into the sender and receiver. The only errors we allow are bit flips during transmission.

The transition systems for these systems are constructed according to the following rules:

1. The state space of the system is $\{0, 1, \#\}^n \times \{0, 1, \#\}^m$ where n and m are constants specific to the system. The first component is the list of bits to

be transmitted by the sender, and the second component is the list of bits already received by the receiver. The initial state is $(\#^n, \#^m)$.
2. The alphabet for the transition systems consist of $\Sigma = \{0, 1, \#\}^4 \times \{0, 1\} \times \{0, 1, \#\}^4$. Here, the first part of the symbol is the input received at the sender, the second part is the bit that is transmitted and the third part is the output at the receiver.
3. All the actions except the transmission are considered to happen instantaneously, as they are local and have negligible error rates.
4. Bit flips can occur during the transmission and the state is changed according to the bit received. These transitions which have a bit flip are considered as erroneous transmissions. To measure the robustness of the system, we will be using the modification scheme *ContErr*, where X is the collection of such erroneous transitions.

Example. Suppose we are working with the Hamming(7,4) system. Let us examine the transmission of the bit block 1100. The encoded bit string for this block is 0111100. Now, 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 $(\#111100, 0\#\#\#\#\#\#)$ (assuming no errors). From this state, we go on the symbol $(\#\#\#\#, 1, \#\#\#\#)$ to the state $(\#\#11100, 01\#\#\#\#\#)$ and so on. The outline of the set of states and transitions for this transmission is illustrated in Figure 5.

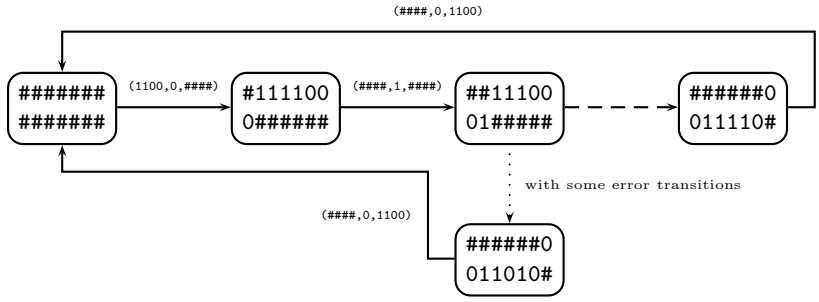


Fig. 5. Part of the transition graph for Hamming(7,4) system

T_1	T_2	$d_{\text{rob}}(T_1, T_2)$
No error correction	Ideal System	1
Hamming (7,4)	Ideal System	6/7
Triple modular redundancy	Ideal System	2/3

Table 4. Robustness of FEC systems

The values of $d_{\text{rob}}$of these systems measured against the ideal system are summarized in Table 4.1. As the table shows, the robustness measured are what

one would expect from the systems. The system which uses no error correction is the least robust as it cannot tolerate even a single bit error. The Hamming(7,4) system does better as it can tolerate one error in seven bits on the long run, whereas the system which uses Triple modular redundancy can tolerate one error in three bits. The robustness values clearly mirror the error tolerance values as each robustness value is equal to $1-e$ where e is the corresponding error tolerance value.

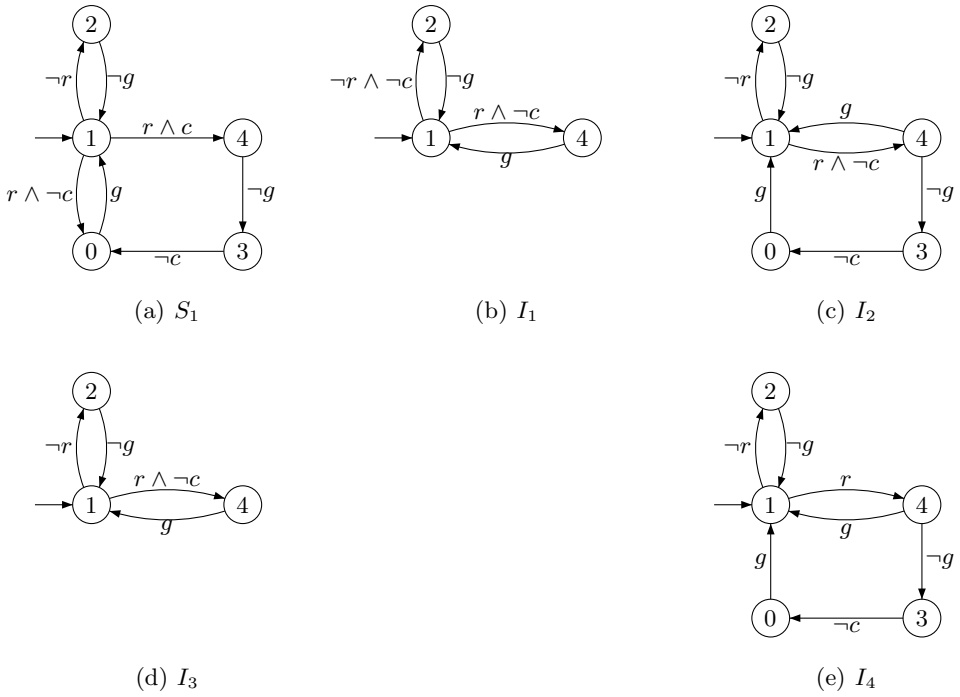
4.2 Environment Restriction for Reactive Systems

In reactive systems, the transitions of the system are controlled by two agents, the system itself and the environment. The system has no control over the actions of the environment. Hence, while considering the refinement of a specification for a reactive system, care has to be taken to ensure that apart from the fact that all behaviors of the implementation are simulated by the specification, but also that the behavior of the environment is not restricted more than in the specification. A number of refinements of the classical simulation relation have been suggested to include this requirement such as ready simulation [3].

We propose here 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 (restricted to the environment actions), 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 that do not allow it. However, say I_1 allows the behavior $(r_1 r_2)^\omega$ whereas I_2 allows only r_2^ω, I_1 will be given a higher rating than I_2.

The way we will measure the amount of environment restriction is using the coverage distance (d_{cov}) introduced in Section 3. We model a reactive system with inputs and outputs as a transition systems 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 Figure 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 requirement mandatorily forbids some environment behaviors. For example, the input behavior with both r and c high all the time does not have any valid output behavior. The specification S_1 restricts the environment so that for every request r, cancel c is low in the current or the following step. This is the most permissive restriction possible. Implementations I_1, I_2, I_3 and I_4 restrict the environment to various amounts by allowing no cancels, allowing no cancels for the relevant two steps, allowing no cancels for the current step,



(a) S_1 (b) I_1 (c) I_2

(d) I_3 (e) I_4

Fig. 6. Request-grant Systems

and allowing no cancel for the following step respectively. The restrictiveness as measured by the $\mathcal{G}_{\text{cov}}$is summarized in Table 4.2.

The values in Table 4.2 reflects the intuitive notion that I_1 is the most restrictive, followed by I_2 and I_3, which are equally restrictive and then by I_4 which allows all the input behaviors of the specification.

5 Conclusion

We have motivated the notion of distance between two systems or between a system and a specification, and introduced quantitative simulation games as a framework for measuring such distances. We presented three particular 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. We plan to investigate how the distances between systems change under certain transformations, such as parallel composition or abstraction. 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

T_1	T_2	$d_{\mathrm{cov}}(T_1, T_2)$
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

Table 5. Restrictiveness of request-grant systems

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, and M. Vardi. Alternating refinement relations. In *CONCUR*, pages 163–178, 1998.
2. R. Bloem, K. Chatterjee, T. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In *CAV*, pages 140–156, 2009.
3. B. Bloom. *Ready simulation, bisimulation, and the semantics of CCS-like languages.* PhD thesis, MIT, 1989.
4. K. Chatterjee, L. de Alfaro, R. Majumdar, and V. Raman. Algorithms for game metrics. In *FSTTCS*, pages 107–118, 2008.
5. K. Chatterjee, L. Doyen, and T. Henzinger. Quantitative languages. In *CSL*, pages 385–400, 2008.
6. K. Chatterjee, L. Doyen, and T. Henzinger. Expressiveness and closure properties for quantitative languages. In *LICS*, pages 199–208, 2009.
7. S. Chidamber and C. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Software Eng.*, 20(6):476–493, 1994.
8. L. de Alfaro, R. Majumdar, V. Raman, and M. Stoelinga. Game refinement relations and metrics. *Logical Methods in Computer Science*, 4(3), 2008.
9. M. Droste and P. Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007.
10. A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. In *International Journal of Game Theory*, pages 163–178, 1979.
11. N. Fenton. *Software Metrics: A Rigorous and Practical Approach, Revised (Paperback).* Course Technology, 1998.
12. R. W. Hamming. Error detecting and error correcting codes. *Bell System Tech. J.*, 29:147–160, 1950.
13. R. Lincke, J. Lundberg, and W. Löwe. Comparing software metrics tools. In *ISSTA*, pages 131–142, 2008.
14. R. E. Lyons and W. Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM J. Res. Dev.*, 6(2):200–209, 1962.
15. R. Milner. An algebraic definition of simulation between programs. In *IJCAI*, pages 481–489, 1971.
16. C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948.

17. F. van Breugel and J. Worrell. An algorithm for quantitative verification of probabilistic transition systems. In *CONCUR*, pages 336–350, 2001.
18. F. van Breugel and J. Worrell. A behavioural pseudometric for probabilistic transition systems. *Theor. Comput. Sci.*, 331(1):115–142, 2005.