Quantitative Simulation Games[⋆]

Pavol Černý, Thomas A. Henzinger, and Arjun Radhakrishna

IST Austria (Institute of Science and Technology, Austria)

Abstract. 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, that is, how much the system can 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, that is, how much the property has to be modified so that it is satisfied by the system. These distances can be computed in polynomial time, since the computation reduces to the value problem in limit average games with constant weights. Finally, we demonstrate how the robustness distance can be used to measure how many transmission errors are tolerated by error correcting codes.

1 Introduction

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).

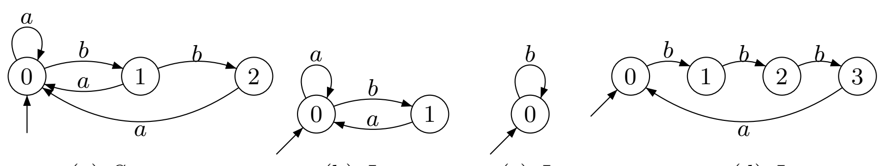
There is thus a natural question whether it is possible to extend the standard specification frameworks and verification algorithms to capture a finer and more

[⋆] This work was partially supported by the European Union project COMBEST and the European Network of Excellence ArtistDesign.

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 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. In this article, the limit average objective function is considered. For example, for incorrect implementations, that is 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.*

Consider now the examples in Figure 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 will thus 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 therefore be $1/4$.



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

Fig. 1. Example Systems

Considering the implementation I_2 from the Figure 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 the specification 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.

The correctness, coverage, and robustness distances can be obtained by solving the value problem in the corresponding games. The distances we considered lead to limit average games with constant weights. The value of such games can be computed in polynomial time [21].

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 [5, 6, 2, 11] 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.

There have been several attempts to give a mathematical semantics to reactive processes which is based on quantitative metrics rather than boolean preorders [19, 8]. In particular for probabilistic processes, it is natural to generalize bisimulation relations to bisimulation metrics [10, 20], 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 [7, 13, 15]). These metrics measure syntactic properties of the source code, and are fundamentally different from our distance functions that capture the difference in the behavior (semantics) of programs.

2 Quantitative Simulation Games

2.1 Transition Systems and Games

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 labeled transitions between the states, 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 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 $\mathcal{S}_Q$.

A run in a transition system T is an infinite path $\rho = \rho_0\rho_1\rho_2 \ldots \in S^\omega$ where for all $i \geq 0$, $(\rho_i, \sigma, \rho_{i+1}) \in E$ for some $\sigma \in \Sigma$.

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 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 $\mathcal{A}_Q$.

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 $\forall i \geq 0 : (\rho_i \in S_p \implies \rho_{i+1} = \pi(\rho_0\rho_1 \dots \rho_i))$. 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 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 $s \in S$.

Games and Objectives. A *game* consists of a game graph and a boolean or quantitative objective. A *boolean objective* is a function $\Phi : \Omega \to \{0, 1\}$. The goal of Player 1 in a game with boolean objective Φ is to choose a strategy so that, no matter what Player 2 does, the outcome maps to 1; and the goal of Player 2 is to ensure that the outcome maps to 0. A *quantitative objective* is a *value* function $f : \Omega \to \mathbb{R}$. 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 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., $\nu_1(\pi_1) = \inf_{\pi_2 \in \Pi_2} f(out(\pi_1, \pi_2))$. 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.

We consider only the *limit-average* quantitative objective. Given a game graph with the weight function v and a play $\rho = \rho_0\rho_1\rho_2 \dots$, for all $i \geq 0$, let $v_i = v(\langle \rho_i, \rho_{i+1} \rangle)$.

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

LimAvg is the long-run average of the weights occurring in a play.

Note that for *LimAvg* objectives, optimal memoryless strategies exist for both players [12].

2.2 Qualitative Simulation Games

The classical simulation preorder [17] 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. These relations can be cast in the form of 2-player games with qualitative 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

1. $(s_0, s'_0) \in H$, and
2. $\forall s, t \in S, s' \in S' : (s, s') \in H \wedge (s, \sigma, t) \in E \Rightarrow (\exists t' : (s', \sigma, t') \in E' \wedge (s', t') \in H)$.

If A' simulates A, we write $A \leq A'$.

We define a restricted form of alternating simulation defined in [1], which is sufficient for the 2-player turn-based alternating systems we consider. 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:

1. $(s_0, s'_0) \in H$, and
2. $\forall s \in S, s' \in S' : (s, s') \in H \Rightarrow (s \in S_1 \Leftrightarrow s' \in S'_1)$
3. $\forall s, t \in S, s' \in S' : ((s, s') \in H \wedge s \in S_1) \Rightarrow \forall (s, \sigma, t) \in E : (\exists (s', \sigma, t') \in E' : (t, t') \in H)$.
4. $\forall s \in S, s', t' \in S' : ((s, s') \in H \wedge s \in S_2) \Rightarrow \exists (s', \sigma, t') \in E' : (\forall (s, \sigma, t) \in E : (t, t') \in H)$.

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 $\mathcal{G}_{A,A'}$ ($\mathcal{H}_{A,A'}$). 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 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* $G_{A,A'} = \langle S^G, S^G_1, S^G_2, \Sigma, E^G, s^G_0 \rangle$ as follows:

1. The alphabet Σ is the same as the alphabet of A and A'.
2. The state space $S^G = S \times (\Sigma \cup \{\#\}) \times S' \cup \{s_{\text{err}}\}$.
3. The states are partitioned into Player 1 and Player 2 states as follows: $(s, \#, s') \in S^G_1$, and $(s, \sigma, s') \in S^G_2$ for all $\sigma \in \Sigma$. Also, state $s_{\text{err}} \in S^G_1$.
4. The initial state is $s_0 = (s_0, \#, s'_0)$.
5. Each transition of the game graph corresponds to a transition in either A or A' as follows:

   (a) $((s,\#,s'),\sigma,(t,\sigma,s')) \in E^G \Leftrightarrow (s,\sigma,t) \in E$
   (b) $((s,\sigma,s'),\sigma,(s,\#,t')) \in E^G \Leftrightarrow (s',\sigma,t') \in E'$

   For each of the above transitions, the weight is the same as the weight of the corresponding transition in A or A'.
6. If there is no outgoing transition from a particular state, transitions to s_{err} are added with all symbols. s_{err} is a sink with transitions to itself on all symbols. Each of these transitions has weight 1.

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 as follows. Given two quantitative 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, we define the corresponding *alternating simulation game graph* $H_{A,A'} = \langle S^H, S^H_1, S^H_2, \Sigma, E^H, s^H_0, v^H\rangle$ as:

1. The alphabet is the same as the alphabet of A and A'. The initial state is $(s_0, \#, s'_A, p)$ 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.
2. Player 1 states of the graph are $S^H_1 = \{(s,\#,s',1) \mid s \in S_1 \wedge s_2 \in S'_1\} \cup \{(s,\sigma,s',1) \mid s \in S_2 \wedge s' \in S'_1 \wedge \sigma \in \Sigma\} \cup \{s_{\text{err}}\}$. The first set of the union represents the states where Player 1 has to choose 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.
3. 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.
4. The transitions correspond to A or A' transitions as follows:
   (a) If (s,σ,t) is a transition in A and $(s,\#,s',1)$ is a Player 1 state, we have the corresponding transition $((s,\#,s',1),\sigma,(t,\sigma,s',2))$ in E^H, i.e., in states where Player 1 has to choose a transition of A, the A component of the state is changed to the destination of the A transition and the symbol is changed to the symbol of the A transition.
   (b) If (s',σ,t') is a transition in A' and $(s,\#,s',2)$ is a Player 2 state, we have the corresponding transition $((s,\#,s',2),\sigma,(s,\sigma,t',1))$ in E^H. These transitions are similar to the previous case, but Player 2 has to choose a A' transition for Player 1 to match.

(c) If (s, σ, t) is a transition in A and $(s, \sigma, s', 1)$ is a Player 1 state, we have the corresponding transition $((s, \sigma, s', 1), \sigma, (t, \#, s', 1))$ in E^H. Here, Player 1 chooses a transition to match the previous move of Player 2 which had the symbol σ. The A component of the state is changed accordingly and the symbol is reset to $\#$.

(d) If (s', σ, t') is a transition in A' and $(s, \sigma, s', 2)$ is a Player 2 state, we have the corresponding transition $((s, \sigma, s', 2), \sigma, (s, \#, t', 2))$ in E^H. This is the dual of the previous case.

The weight of each transition is equal to the weight of the corresponding A or A' transition.

5. If there is no outgoing transition from a particular state, a transition to s_{err} is added on all symbols. s_{err} is a sink with transitions to itself on all symbols. Each of these transitions has weight 1.

We consider the game graph without weights to define the alternating simulation game. As in the case of simulation games, the objective of Player 2 is to ensure that the play proceeds ad infinitum without reaching s_{err}, and the objective of Player 1 is to ensure that the play reaches s_{err}. We denote this qualitative alternating simulation game as $\mathcal{H}^{A,A'}$. Intuitively, as in the simulation game, whenever in a Player 1 state of A, Player 1 chooses a transition and Player 2 has to match it in A'. But, in a Player 2 state of A', Player 2 chooses a transition of A' and Player 1 matches it in A. Player 1 wins if a transition cannot be matched at some point, and Player 2 wins otherwise. Again, it can be seen that alternating simulation of A by A' holds iff there exists a winning strategy for Player 2 .

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 *LimAvg* objective to measure distances between systems.

Quantitative Simulation Games. Given two quantitative transition systems A and A', the *quantitative simulation game* is played on the quantitative simulation game graph $G_{A,A'}$ with the objective of Player 1 being to maximize the *LimAvg* value of the play, while the objective of Player 2 being to minimize it. We denote this game as $\mathcal{Q}_{A,A'}$.

Quantitative Alternating Simulation Games. For two alternating transition systems A and A', we similarly define the *quantitative alternating simulation game* played on $H_{A,A'}$ with the same objectives as a quantitative simulation game. We denote this game as $\mathcal{P}_{A_1,A_2}$.

2.4 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 with respect to a specification, we add to the specification some recovery behavior to be used in case of error. To ensure that the specification does not use this recovery behavior when it is not necessary, there is 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 basic structure system, we impose a strict set of rules on these schemes.

A *modification scheme* is a function $m : \mathcal{S} \to \mathcal{S}_Q \cup \mathcal{A}_Q$ from transition systems to quantitative (alternating) transition systems, which can be computed using the following steps:

1. Edges may be added to the transition system.
2. Each state may be replaced by a local subgraph. The graph is the same for all states of the system. All 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 $\mathbb{Q}$.

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

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 an weight of 2 to prohibit their free use. All other transitions have the weight zero. Given a transition 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 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 occurring, whereas Player 1 actually chooses the transition the system takes. We penalize Player 2 for the choice of not allowing errors to happen.

Given $T = \langle S, \Sigma, E, s_0 \rangle$ we define *ErrMod*(T) to be the quantitative alternating transition system obtained after the following steps.

1. All transitions that differ from existing transitions only in the symbol are added to the system as in *OutMod*.
2. The graph used to replace each state s is shown in Figure 2.
3. Only the transitions labeled $\neg c$ are given the weight 2. The rest are given the weight 0.

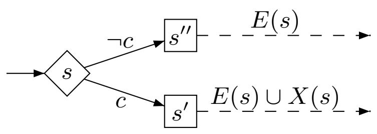


Fig. 2. Graph for *ErrMod*

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

3 Simulation Distances

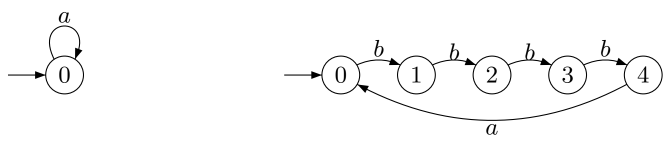
We present here examples of distances that can be defined using quantitative simulation games.

3.1 Correctness

Given a specification T_2 and an implementation T_1, such that T_2 is incorrect with respect to T_1, the correctness distance measures the degree of incorrectness. 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 symbol 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 3.1 (Correctness distance)**.** *The* correctness distance $d_{cor}(T_1, T_2)$ *from transition system T_1 to transition system T_2 is the Player 1 value of the quantitative simulation game $\mathcal{C}_{T_1,T_2} = \mathcal{Q}_{NoMod(T_1),OutMod(T_2)}$.*

The game $\mathcal{C}$ 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, but the specification T_2 is allowed to make errors. The implementation T_1 tries to make the specification commit as many errors as possible. Every move by Player 1 chooses a transition of T_1. Every matching move of Player 2 is a zero weight transition of T_2. If Player 2 cannot match a move, she must still choose an existing transition an incurs a weight of 2. (Other error models are possible where Player 2 can use a completely new transition.) 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. If the implementation is correct (T_2 simulates T_1), then the correctness distance is 0. Otherwise, the value of the game is the *LimAvg* of the number of errors, i.e., the long run average of the errors.



(a) I_1 (b) I_5

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 and Figure 1, 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 *LimAvg* correctness game are summarized in Table 1.

T_1	T_2	$d_{\mathrm{cor}}(T_1, T_2)$
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 the Correctness game

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.

3.2 Coverage

Now, we present the dual game of the one presented above. Here, we measure the behaviors that are present in one system but not in the other system. Given a specification T_2 and an implementation T_1, 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 3.2 (Coverage distance). *The* coverage distance $d_{cov}(T_1, T_2)$ *from transition system* T_1 *to transition system* T_2 *is the Player 1 value of the quantitative simulation game* $\mathcal{V}_{T_1,T_2} = \mathcal{Q}_{NoMod(T_2),OutMod(T_1)}$.

$\mathcal{V}$ measures the distance from T_1 to T_2 as the minimal number of errors that have to be committed by T_1 to cover all the behaviors of T_2.

Again, we present examples of systems and their distances according to $\mathcal{V}$. We use the systems from the examples in Figure 3 and Figure 1. The distances are summarized in Table 2.

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

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 whether implementation commits errors. Hence, in that case, the robustness distance will be 1.

Definition 3.3 (Robustness distance). *The* robustness distance $d_{rob}(T_1, T_2)$ *from transition system* T_1 *to transition system* T_2 *is the Player 1 value of the quantitative alternating simulation game* $\mathcal{R}_{T_1,T_2} = \mathcal{P}_{ErrMod(T_1),ErrMod_{\emptyset}(T_2)}$.

The game $\mathcal{R}_{ErrMod(T_1),ErrMod_{\emptyset}(T_2)}$ is simple and is played in the following steps:

1. The specification T_2 chooses whether the implementation T_1 is allowed to make an error.
2. The implementation chooses a transition on the implementation system. She is allowed to err based on the specification choice in the previous step.

3. Specification chooses a matching move to simulate the implementation.

The specification tries to minimize the number of moves where she prohibits the implementation to commit 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.

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 the Robustness game

Let us examine the examples from Figure 1 and Figure 3 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 outputting 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.

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.

The d_{cor}, d_{cov} and d_{rob} games are simple graph games with *LimAvg* objectives. The decision problem (deciding whether the value is greater than a given value) for these games is in NP $\cap$ co-NP [21], 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 [21] 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; and E and E' are the sets of transitions of the two systems.

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 $\mathcal{R}$ 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 [18]. 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 [14], and triple modular redundancy [16]. 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.

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 $(\#^n, \#^m)$.

```
$\boxed{\text{No error correction}}$
proc sender($\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$) ≡
    call send($\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$); .
proc receiver() ≡
    call receive($\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$);
    return($\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$).
```

```
$\boxed{\text{Hamming(7,4) error correction}}$
proc sender($\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$) ≡
    $\mathbf{P}_0 := \mathbf{B}_0 \oplus \mathbf{B}_1 \oplus \mathbf{B}_3$
    $\mathbf{P}_1 := \mathbf{B}_0 \oplus \mathbf{B}_2 \oplus \mathbf{B}_3$
    $\mathbf{P}_2 := \mathbf{B}_1 \oplus \mathbf{B}_2 \oplus \mathbf{B}_3$
    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() ≡
    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($\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$).
```

```
$\boxed{\text{Triple modular redundancy}}$
proc sender($\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$) ≡
    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() ≡
    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

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 *ErrMod*. 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.

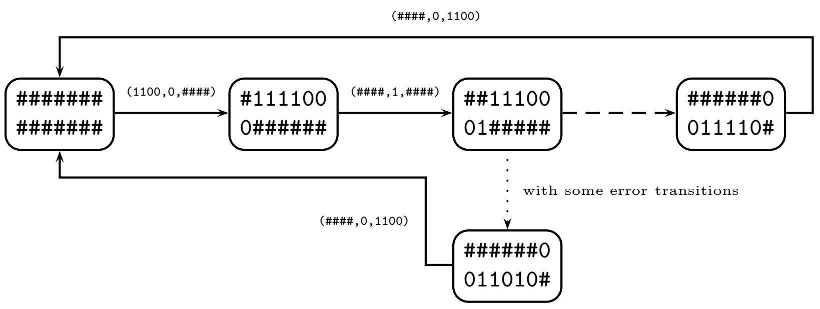


Fig. 5. Part of the transition graph for Hamming(7,4) system

T_1	T_2	$d_{\mathrm{rob}}(T_1, T_2)$
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 $d_{\mathrm{rob}}$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, the implementation I_1 will be given a higher rating than the implementation 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.

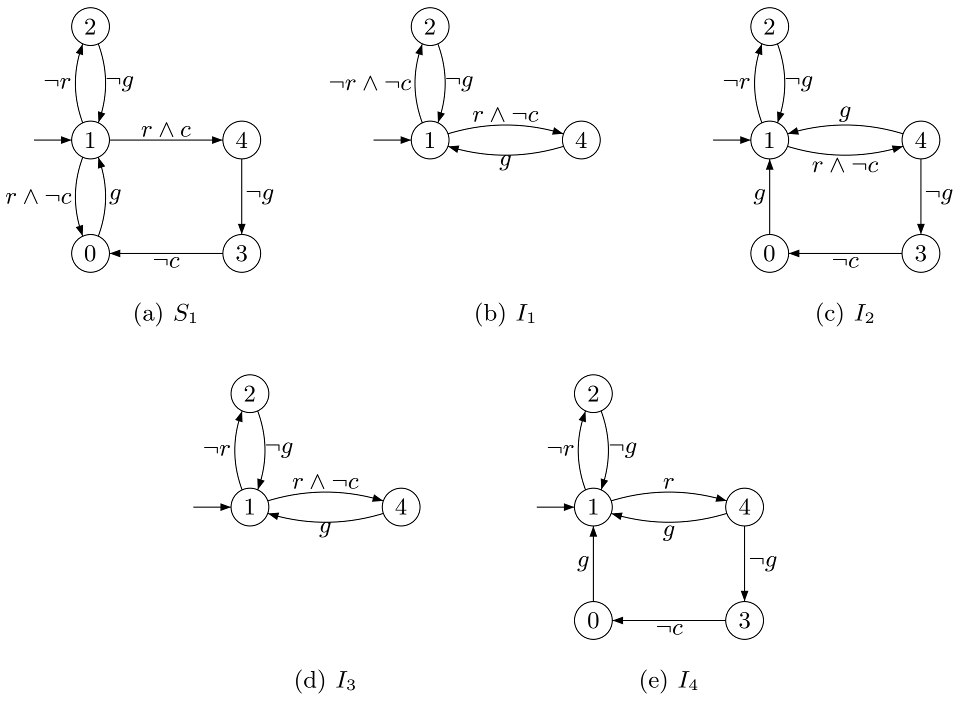


Fig. 6. Request-grant Systems

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 at all, allowing no cancels for the relevant two steps, allowing no cancel when there is a request, and allowing no cancel for the step following a request respectively. The restrictiveness as measured by the $\mathcal{V}$ is summarized in Table 4.2.

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

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. The theoretical aspects of the quantitative simulation game framework need to be developed. In the boolean setting, the simulation relation establishes a preorder on systems. A preorder is a reflexive and transitive relation; a generalization to the quantitative setting would be a directed metric, that is, a distance function such that the distance of an object to itself is zero, and such that it conforms to the triangle inequality property. We will investigate whether these properties hold for the distance function we defined. Furthermore, we plan to investigate how the distances between systems change under certain transformations, such as parallel composition or abstraction. Another 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 conforming to the specification. 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. P. Caspi and A. Benveniste. Toward an approximation theory for computerised control. In *EMSOFT*, pages 294–304, 2002.
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, M. Faella, and M. Stoelinga. Linear and branching system metrics. *IEEE Trans. Software Eng.*, 35(2):258–273, 2009.
9. L. de Alfaro, T. Henzinger, and R. Majumdar. Discounting the future in systems theory. In *ICALP*, pages 1022–1037, 2003.
10. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled markov processes. *Theor. Comput. Sci.*, 318(3):323–354, 2004.
11. M. Droste and P. Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007.
12. A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. In *International Journal of Game Theory*, pages 163–178, 1979.
13. N. Fenton. *Software Metrics: A Rigorous and Practical Approach, Revised (Paperback).* Course Technology, 1998.
14. R. W. Hamming. Error detecting and error correcting codes. *Bell System Tech. J.*, 29:147–160, 1950.
15. R. Lincke, J. Lundberg, and W. Löwe. Comparing software metrics tools. In *ISSTA*, pages 131–142, 2008.
16. 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.
17. R. Milner. An algebraic definition of simulation between programs. In *IJCAI*, pages 481–489, 1971.
18. C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948.
19. F. van Breugel. An introduction to metric semantics: operational and denotational models for programming and specification languages. *Theor. Comput. Sci.*, 258(1-2):1–98, 2001.
20. F. van Breugel and J. Worrell. Approximating and computing behavioural distances in probabilistic transition systems. *Theor. Comput. Sci.*, 360(1-3):373–385, 2006.
21. U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158(1&2):343–359, 1996.