Synthesis from Incompatible Specifications*

Pavol Černý
IST Austria

Sivakanth Gopi
IIT Bombay

Thomas A. Henzinger
IST Austria

Arjun Radhakrishna
IST Austria

Nishant Totla
IIT Bombay

ABSTRACT

Systems are often specified using multiple requirements on their behavior. In practice, these requirements can be contradictory. The classical approach to specification, verification, and synthesis demands more detailed specifications that resolve any contradictions in the requirements. These detailed specifications are usually large, cumbersome, and hard to maintain or modify. In contrast, quantitative frameworks allow the formalization of the intuitive idea that what is desired is an implementation that comes "closest" to satisfying the mutually incompatible requirements, according to a measure of fit that can be defined by the requirements engineer. One flexible framework for quantifying how "well" an implementation satisfies a specification is offered by simulation distances that are parameterized by an error model. We introduce this framework, study its properties, and provide an algorithmic solution for the following quantitative synthesis question: given two (or more) behavioral requirements specified by possibly incompatible finite-state machines, and an error model, find the finite-state implementation that minimizes the maximal simulation distance to the given requirements. Furthermore, we generalize the framework to handle infinite alphabets (for example, real-valued domains). We also demonstrate how quantitative specifications based on simulation distances might lead to smaller and easier to modify specifications. Finally, we illustrate our approach using case studies on error correcting codes and scheduler synthesis.

Categories and Subject Descriptors

D.1.2 [**Programming Techniques**]: Automatic Programming

*This research was supported in part by the European Research Council (ERC) Advanced Investigator Grant QUAREM and by the Austrian Science Fund (FWF) project S11402-N23.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EMSOFT'12, October 7-12, 2012, Tampere, Finland.
Copyright 2012 ACM 978-1-4503-1425-1/12/09 ...$15.00.

General Terms

Theory, Verification

Keywords

synthesis, incompatible specifications

1. INTRODUCTION

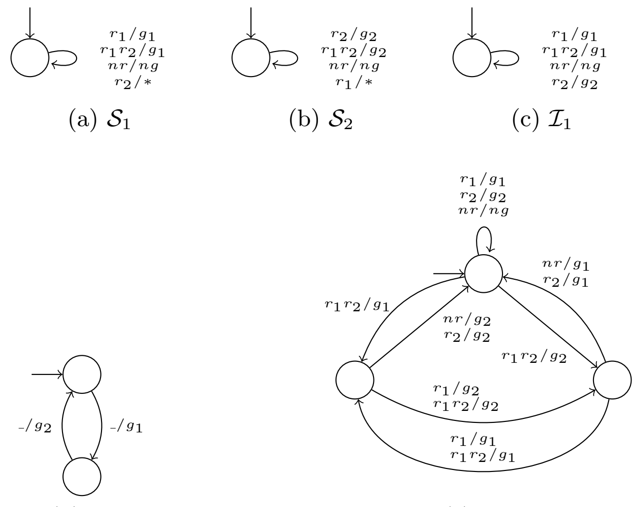
A major problem for the wider adoption of techniques for the formal verification and synthesis of systems is the difficulty of writing quality specifications. Quantitative specifications have the potential to simplify the task of the designer, by enabling her to capture her intent better, and more simply. In this paper, we focus on how quantitative specification and reasoning can be useful in cases when specifications are mutually incompatible. In practice, specifications of systems are often not monolithic. They are composed of parts that express different design requirements, possibly coming from different sources. Such high-level requirements can be therefore often contradictory (see, for instance, [16, 2, 14] which provide methods for requirements analysis). Using the classic boolean approach, the solution would be to resolve conflicts by writing more detailed specifications that cover all possible cases of contradictions, and say how to resolve them. However, such specifications may become too large, and more importantly, the different requirements become entangled in the specification. The specifications are then much more difficult to maintain than the original requirements, as it is hard to modify one requirement without rewriting the rest of the specification. In contrast, quantitative frameworks allow the formalization of the intuitive idea that what is desired is an implementation that comes "closest" to satisfying the requirements. More technically, we consider two questions: first, the (rigorously defined) distances from the implementation to (boolean) requirements are within given bounds, and second, the maximal distance to a requirement is minimized.

Furthermore, quantitative reasoning about systems is gaining importance with the spread of embedded systems with strict requirements on resource consumption and timeliness of response. The quantitative approach in this paper is fully compatible with quantitative resource (e.g. memory, energy) consumption requirements: the framework allows us to consider multiple specifications that model resource consumption, and it allows us to express the relative importance of resources. We can then ask the same two questions as above: first, we would like an implementation such that it consumes resources within given bounds; or, second, an implementation such that its maximal total consumption of a resource is minimized.

Synthesis from specifications [18, 17, 9] has been studied extensively as a technique to improve designer and programmer productivity. If designers are forced to write detailed low-level specifications to reconcile contradictory requirements, it decreases the usefulness of synthesis. First, it requires more effort from designers, requiring consideration of *how* a certain task will be performed, as opposed to *what* task should be performed. Second, the space of solutions to the synthesis problem is reduced, with possibly good implementations being ruled out. We therefore propose quantitative synthesis as a solution to the problem of synthesis from incompatible specifications.

Motivating example. Consider a system that grants exclusive access to a resource to two processes which periodically seek access to it. The specification of the system consists of two parts: the first (resp. second) part R_1 (resp. R_2) states that a request for the resource from Process 1 (Process 2) should be satisfied with a grant in the same step. The input alphabet $\mathcal{I}$ consists of symbols r_1, r_2, r_1r_2 and nr representing that requests from either Process 1, Process 2, both, or neither, respectively, coming in the current step. The output alphabet $\mathcal{O}$ consists of symbols g_1 (g_2) representing granting the resource to Process 1 (Process 2), and a special "don't care" symbol $*$. The two parts of the specification are shown in Figures 1a and 1b. The specifications are incompatible, because on input r_1r_2 the specification $\mathcal{S}_1$ allows only g_1, whereas specification $\mathcal{S}_2$ allows only g_2. Classically, the designer would have to manually resolve the conflict by, for example, constructing a specification that grants to Process 1 whenever both processes request in the same step (requirement R_3). However, the designer might not want to resolve the conflict at the specification level, but instead might want to state that she wants an implementation that comes close to satisfying the two mutually incompatible specifications, according to a measure of correctness. We provide a rigorous way for defining such measures.

Measuring correctness of an implementation. We model both systems and specifications as finite-state machines. For defining distances between systems (or between systems and specifications), we build on the simulation distances framework of [6]. Simulation distances generalize the simulation relation, which is a standard correctness condition, to the quantitative setting by measuring how "close" an implementation comes to satisfying a specification. In the classic boolean case, simulation can be seen as a 2-player game between an implementation $\mathcal{I}$ and a specification $\mathcal{S}$, where Player 1 chooses moves (transitions) from the implementation and Player 2 tries to match each move in the specification. In order to generalize this definition to the quantitative setting, we allow the players to make errors (intuitively, choose non-existent transitions), but they pay a certain price for each such choice. The cost of a trace is given by an objective function. We focus on the limit average objective (and thus long-term behavior of systems). The goal of Player 1 (resp., Player 2) is to maximize (resp. minimize) the cost. The best value Player 1 can achieve is then taken as the cost of an implementation with respect to the specification. In this paper, we extend the simulation distances of [6] in two ways: first we consider finite-state machines with both inputs and outputs, and second, we allow specifying simulation distances using error models. The error models are finite-state machines that specify what other additional options the simulating player (Player 2) has. Intuitively, the error models allow specifying how the simulating player can "cheat" in the simulation game.



(a) $\mathcal{S}_1$ (b) $\mathcal{S}_2$ (c) $\mathcal{I}_1$

(d) $\mathcal{I}_2$ (e) $\mathcal{I}_3$

Figure 1: Example 1

Synthesis from incompatible specifications. The main technical problem we concentrate on in this paper is the problem of synthesis from incompatible specifications. The input to the synthesis problem consists of a set of (two or more) mutually incompatible specifications given by finite-state open reactive systems, and a simulation distance (given by an error model). The output should be an implementation, given by a deterministic open reactive system, that minimizes the maximal simulation distance to the given specifications.

Motivating example (continued). Let us consider an error model that, intuitively, (i) assigns a cost 1 if the implementation does not grant a request that has arrived in the current step, and assigns the same cost for every step before the implementation grants the request, and (ii) assigns a cost 1 if the implementation grants a request that is not required in the current step. Let us now consider the three different implementations in Figures 1c, 1d, and 1e, and their distances to the specifications $\mathcal{S}_1$ and $\mathcal{S}_2$. The implementation $\mathcal{I}_1$ always prefers the request r_1 when the two requests arrive at the same time. The implementation $\mathcal{I}_1$ satisfies the specification $\mathcal{S}_1$, but on the input $(r_1r_2)^\omega$, $\mathcal{I}_1$ makes a mistake at every step w.r.t. $\mathcal{S}_2$. The implementation $\mathcal{I}_1$ thus has distance 0 from $\mathcal{S}_1$, and distance 1 from $\mathcal{S}_2$. The implementation $\mathcal{I}_2$ handles the sequence $(r_1r_2)^\omega$ gracefully by alternation (note that _ in Figure 1d matches any input). However, on the input sequence $(nr)^\omega$, $\mathcal{I}_2$ grants at every step, even though it should not grant at all. It thus has a distance of 1 to both $\mathcal{S}_1$ and $\mathcal{S}_2$. The implementation $\mathcal{I}_3$ also alternates grants in cases when the requests arrive at the same step, but does not grant unnecessarily. Its distance to both specifications would be $\frac{1}{2}$. This is because the worst-case input for this implementation is the sequence $(r_1r_2)^\omega$ and on this input sequence, it makes a mistake in every other step, w.r.t. $\mathcal{S}_1$ as well as $\mathcal{S}_2$.

The quantitative approach can be compared to the classical boolean approach to illustrate how it leads to specifications that are easier to modify:

- Consider an alternate requirement R_1' which says that every request by Process 1 should be granted in the next step (instead of the same step). In the boolean case, replacing requirement R_1 by R_1' also involves changing the requirement R_3 which resolves the conflict between R_1 and R_2. Requirement R_3 needs to be changed to R_3' which says that given that request r_1 happened in the previous step and r_2 happened in the current step, the output must be g_1 in the current step. However, in the quantitative case, no changes need to be done other than replacing R_1 with R_1'.
- Similarly, we can consider other ways of resolving the conflict between requirements R_1 and R_2, instead of using R_3 which prioritizes Process 1 over Process 2. We could have the requirement that we are equally tolerant to missed grants in each process (say requirement R_3') or that we tolerate twice as many missed grants in Process 1 than in Process 2, just by modifying the penalties in the error models. In the boolean case, the requirement R_3 is easily expressible, but the requirement R_3' is very hard to state without adding additional constraints to the specification. In the quantitative case, we can simply switch between R_3 and and R_3' just by changing the relative penalties for not granting r_1 or r_2.
- To illustrate how our framework can model resource consumption, we consider a system that sends messages over a network, as governed by a correcteness specification. It costs a certain amount (in dollars) to send a kB of data, so it might be useful to compress data first. However, compression uses energy (in Joules). In our framework, we could add two boolean requirements saying that (a) data should not be sent on the network, and (b) compression should not be used. Then we can relax the requirement, by giving error models that have costs for sending data and using compression. In this way, the framework allows to synthesize a system where e.g. both total energy costs and total network costs are within certain bounds. For further illustration of resource consumption modeling, we refer the reader to our case study on forward error correction codes, where the number of bits sent is the resource tracked.

Overview of results The main result of this paper is an ϵ-optimal construction for the synthesis from incompatible specifications problem. We first consider the decision version of the problem: given k possibly mutually incompatible specifications, and a maximum distance to each specification, the problem is to decide whether there exists an implementation that satisfies these constraints. We show that the decision problem is coNP-complete (for a fixed k). The result is obtained by reduction to 2-player games with multiple limit average objectives [7]. We then present a construction of an ϵ-optimal strategy for the problem of synthesis for incompatible specifications. Furthermore, for the case of two specifications, and for a specific error model (already considered in [6]), we show that the result of our optimal synthesis procedure is always better (in a precise sense) than the result of classical synthesis from just one of the specifications.

Moreover, we extend the framework of simulation distances [6] to open reactive systems (with both inputs and outputs), and introduce parametric stateful error models. We prove that simulation distances define a directed metric (i.e., the distance is reflexive and the triangle inequality holds) in this generalized setting.

We also study an extension of the framework to distances for automata on infinite metric alphabets, to model for example controlling a real-valued output. We present an algorithm (simpler than in the finite-alphabet setting) for solving the problem of synthesis from incompatible specifications in this more flexible setting. We then study the case when the controller can set the output only to a finite number of values from the infinite alphabet, and in this case we give an algorithm as well as a more efficient heuristic based on a projection theorem.

Finally, we demonstrate how our methods can enable simpler specifications, while allowing the synthesis of desirable solutions, using two case studies: on synthesis of custom forward error correction codes and on scheduler synthesis.

Related work. The fact that in practice requirements on systems might be inconsistent was recognized in the literature, and several approaches for requirement analysis [16, 2, 14] and requirement debugging [15] were proposed. The problem of an inconsistent specification was approached in [8] by synthesizing additional requirements on the environment so that unrealizability in the specification is avoided. It was also observed that quantitative measures can lead to simpler specifications [3]. There have been several attempts to give a mathematical semantics to reactive processes based on quantitative metrics rather than boolean preorders [19, 10]. In particular for probabilistic processes, it is natural to generalize bisimulation relations to bisimulation metrics [11], and similar generalizations can be pursued if quantities enter through continuous variables [4]. In contrast, we consider distances between purely discrete (non-probabilistic, untimed) systems.

Synthesis from inconsistent specifications was considered in [13, 12]. Here the conflicts between various components of the specification are resolved by considering priorities for different components, in contrast to our approach of using quantitative measures of correctness. However, it is not possible to express requirement such as R_3' from the motivating example using priorities. Synthesis with respect to quantitative measures was considered in [3, 5], but only for consistent specifications, and not for simulation distances.

2. DISTANCES ON SYSTEMS

Alternating Transition Systems. An *alternating transition system* (ATS) $\langle S, \Sigma, E, s_0, (S_1, S_2)\rangle$ consists of a finite set of states S, a finite alphabet Σ, an initial state s_0, a transition relation $E \subseteq S \times \Sigma \times S$ and a partition (S_1, S_2) of S into Player 1 and Player 2 states. We require that for every state $s \in S$, there exists a transition from s, i.e., $\forall s \in S : \exists \sigma \in \Sigma, s' \in S : (s, \sigma, s') \in E$. A *run* in a transition system is an infinite path $\rho = \rho_0\sigma_0\rho_1\ldots$ where $\rho_0 = s_0$ and $\forall i \geq 0 : (\rho_i, \sigma_i, \rho_{i+1}) \in E$. If the set S_1 is empty in an ATS, we call it a *transition system* and denote it by $\langle S, \Sigma, E, s_0\rangle$.
Reactive Systems. An *open reactive system* is a restriction of an ATS where: (i) Alphabet Σ is the disjoint union of inputs $\mathcal{I}$ and outputs $\mathcal{O}$; (ii) Transition relation is strictly alternating on S_1 and S_2, and inputs and outputs, i.e., $E \subseteq (S_1 \times \mathcal{I} \times S_2) \cup (S_2 \times \mathcal{O} \times S_1)$; (iii) Transition relation is deterministic in inputs, i.e., $(s, \sigma, s') \in E \wedge (s, \sigma, s'') \in E \wedge$

$\sigma \in \mathcal{I} \implies s' = s''$; and (iv) Transition relation is input enabled, i.e., $\forall s \in S_1, \sigma \in \mathcal{I} : \exists s' : (s, \sigma, s') \in E$.

In a reactive system, the computation proceeds with the environment (Player 1) choosing an input symbol from every state in S_1 and the system (Player 2) choosing an output symbol from every state in S_2. Each run is called a *behavior* of the system. We say a reactive system is an *implementation* if for all $s \in S_2$, there is exactly one transition leading out of s.

2-player Games. A 2-player game is an ATS (called game graph) along with an objective (defined below). In a game, a token is placed on the initial state; and Player i chooses the successor whenever the token is on a state in S_i. The set of all runs is denoted by Ω.

Strategies. A *strategy* for Player i in a game is a recipe that tells the player how to choose the successors in a game. Formally, a Player i strategy $\pi_i : (S \times \Sigma)^* \cdot S_i \to \Sigma \times S$ is a function such that for each $w \cdot s \in (S \times \Sigma)^* \cdot S_i$ and $\pi_i(w \cdot s) = (\sigma, s')$, we have $(s, \sigma, s') \in E$. Each w is called a history. The sets of all Player 1 and Player 2 strategies are denoted by Π_1 and Π_2, respectively. A play $\rho = \rho_0\sigma_0\rho_1\sigma_1 \ldots$ *conforms* to a Player j strategy if $\forall i \geq 0 : \rho_i \in S_j \implies (\rho_{i+1}, \sigma_i) = \pi_1(\rho_0\sigma_0 \ldots \rho_i)$. The *outcome* of strategies π_1 and π_2 is the unique path $out(\pi_1, \pi_2)$ that conforms to both π_1 and π_2.

We use two restrictions of strategies. Strategy π_i is:

- *Memoryless* if the output of the strategy function depends only upon the last state in the history. More formally, a strategy is memoryless if $\forall w_1, w_2 \in (S \times \Sigma)^*$ and all $s \in S_i$, we have $\pi_i(w_1 \cdot s) = \pi_i(w_2 \cdot s)$;
- *Finite-memory* if the output depends only upon the current state of a finite memory and the last state in the history. The memory is updated using a memory update function every time a player makes a move. More formally, a strategy is finite-memory, if there exists a finite memory set M, an initial memory state m_0, a memory update function $\pi_i^M : M \times (S \times \Sigma)^* \cdot S \to M$ and a output function $\pi_i^O : S_i \times M \to \Sigma \times S$ such that:
  - $\pi_i(w \cdot s) = \pi_i^O((s, \pi_i^M(m_0, w)))$ for all $w \in (S \times \Sigma)^*$ and $s \in S_i$; and
  - $\pi_i^M(m, \rho_0\sigma_0 \ldots \rho_i\sigma_i\rho_{i+1}) = \pi_i^M(\pi_i^M(m, \rho_0\sigma_0 \ldots \rho_i), \rho_i\sigma_i\rho_{i+1})$.

The set of Player i finite-memory and memoryless strategies is denoted by Π_i^{FM} and Π_i^{ML} respectively.

Objectives. A *boolean objective* is a function $\Phi : \Omega \to \{0, 1\}$ and *quantitative objective* is a function $\Psi : \Omega \to \mathbb{R}$ and the goal of Player 1 in each case is to choose a strategy such that no matter what strategy Player 2 chooses, the value of the outcome is maximized.

Optimal strategies and Values. For a boolean objective Φ, a strategy π_1 (π_2) is *winning* for Player 1 (2) if all plays conforming to it map to 1 (0). For a quantitative objective Ψ, the *value* of a Player 1 strategy π_1 is $Val(\pi_1) = \inf_{\pi_2 \in \Pi_2} \Psi(out(\pi_1, \pi_2))$, and of a Player 2 strategy π_2 is $Val(\pi_2) = \sup_{\pi_1 \in \Pi_1} \Psi(out(\pi_1, \pi_2))$. The *value of the game* is $Val(\Psi) = \sup_{\pi_1 \in \Pi_1} Val(\pi_1)$. A strategy π_i is optimal if $Val(\pi_i)$ is equal to value of the game.

If the above definitions are restricted to finite-memory strategies instead of arbitrary strategies, we get the notions of finite-memory values of strategies and games. In this paper, by default, value is taken to mean finite-memory value.

Reachability and LimAvg objectives. A boolean *reachability objective* $Reach(T)$ for $T \subseteq S$ has $Reach(T)(\rho_0\sigma_0\rho_1\sigma_1 \ldots) = 1$ if and only if $\exists i : \rho_i \in T$, i.e., Player 1 tries to reach the target states T while Player 2 tries to avoid them.

For any ATS, a *weight function* ν maps the transition set E to $\mathbb{N}^k$. Given ν:

- *Limit-average objective* $LimAvg^\nu : \Omega \to \mathbb{R}$ for $k = 1$ has $LimAvg(\rho_0\sigma_0\rho_1 \ldots) = \liminf_{n\to\infty} \frac{1}{n} \cdot \sum_{i=0}^{n} \nu((\rho_i, \sigma_i, \rho_{i+1}))$.
- *Multi-limit-average objective* $MLimAvg^\nu : \Omega \to \mathbb{R}$ maps plays to the maximum $LimAvg$ value of projections of ν to each component of the tuple.
- *Multi-limit-average threshold* objective $MLimAvg_{\mathbf{v}}$ ($\mathbf{v} \in \mathbb{R}^k$) is a boolean objective where a path is mapped to 1 if and only if there is an i, the $LimAvg$ of the i^{th} component of ν is more than the i^{th} component of $\mathbf{v}$.

Note that we use the dual of the standard definition of $MLimAvg_{\mathbf{v}}$ used in [7], i.e., we use existential quantification over i rather than universal. However, all results from [7] can be transferred by switching Player 1 and Player 2.

2.1 Quantitative Simulation Games

The simulation preorder is a widely used relation to compare two transition systems and was extended in [1] to alternating transition systems. The simulation preorder can be computed by solving a 2-player game.

Simulation and Simulation Games. Let $\mathcal{A} = \langle S, \Sigma, E, s_0, (S_1, S_2) \rangle$ and $\mathcal{A}' = \langle S', \Sigma, E', s'_0, (S'_1, S'_2) \rangle$ be two reactive systems. The system $\mathcal{A}'$ *simulates* the system $\mathcal{A}$ if there exists a relation $R \subseteq S \times S'$ such that:

- $(s_0, s'_0) \in R$;
- $(s, s') \in R \implies (s \in S_1 \leftrightarrow s' \in S'_1)$;
- $\forall s, t \in S, s' \in S' : (s, s') \in R \wedge (s, \sigma, t) \in E \wedge s \in S_2 \implies (\exists t' : (s', \sigma, t') \in E' \wedge (t, t') \in R$; and
- $\forall s \in S, s', t' \in S' : (s, s') \in R \wedge (s', \sigma, t') \in E' \wedge s' \in S'_1 \implies (\exists t : (s, \sigma, t) \in E \wedge (t, t') \in R$.

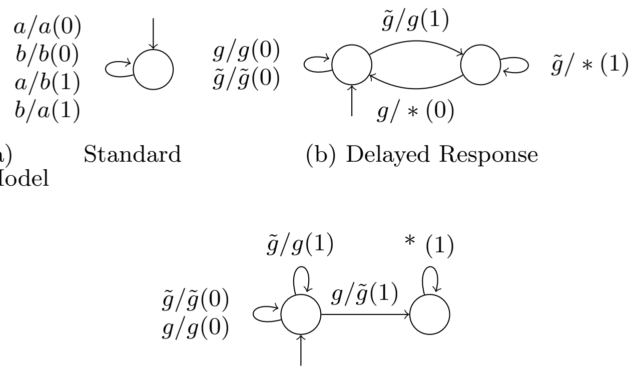
We can construct a game $\mathcal{G}^{\mathcal{A},\mathcal{A}'}$ with a reachability objective such that $\mathcal{A}'$ simulates $\mathcal{A}$ if and only if Player 2 has a winning strategy. The game graph $\mathcal{G}^{\mathcal{A},\mathcal{A}'}$ consists of the Player 1 states $S_1 \times \{\#\} \times S'_1 \bigcup S_2 \times \{\#\} \times S'_2$ and Player 2 states $S_1 \times \mathcal{I} \times S'_2 \bigcup S_1 \times \mathcal{O} \times S'_2 \bigcup \{s_{err}\}$, the alphabet Σ, and the initial state $(s_0, \#, s'_0)$. The transition set consists of the following transitions:

- $((s, \#, s'), \sigma, (t, \sigma, s'))$ where $(s, \sigma, t) \in E \wedge s \in S_2$;
- $((s, \#, s'), \sigma, (s, \sigma, t'))$ where $(s', \sigma, t') \in E' \wedge s \in S_1$;
- $((s, \sigma, s'), \sigma, (s, \#, t'))$ where $(s', \sigma, t') \in E' \wedge \sigma \in \mathcal{O}$;
- $((s, \sigma, s'), \sigma, (t, \#, s'))$ where $(s, \sigma, t) \in E \wedge \sigma \in \mathcal{I}$;
- $(s, \#, s_{err})$ for all Player 2 states s.

The objective for Player 1 is $Reach(\{s_{err}\})$.

Intuitively, in each step of the simulation game, either Player 1 chooses an input transition of $\mathcal{A}'$ and Player 2 matches it with an input transition of $\mathcal{A}$, or Player 1 chooses an output transition of system $\mathcal{A}$ and Player 2 matches it with an output transition of $\mathcal{A}'$. If Player 2 is not able to match a transition, s_{err} is visited and Player 2 loses the game. It is straightforward to show that $\mathcal{A}'$ simulates $\mathcal{A}$ if and only if Player 1 has a winning strategy.

Simulation Distances. In quantitative simulation games [6], Player 2 can simulate an $\mathcal{A}$ transition by an $\mathcal{A}'$ transition with a mismatching label. However, such mismatches have a cost and Player 2 tries to minimize the $LimAvg$ of costs. When simulation holds, there are no mismatches and the value of the game is 0. Here, we present a



(a) Standard Model (b) Delayed Response

(c) No Spurious Response

Figure 2: Sample error models

generalization of quantitative simulation games for reactive systems with richer error models.

Error Models. The modification schemes used in [6] to model the permitted errors during the simulation game do not cover some natural error schemes. For example, the criteria used for request-grant systems in Section 1 is not expressible as a modification scheme from [6]. Hence, we define more general error models. An *error model* over an alphabet $\Sigma = \mathcal{I} \cup \mathcal{O}$ is a deterministic weighted transition system over $\mathcal{O} \times \mathcal{O}$. Intuitively, a transition on label (σ_1, σ_2) (henceforth denoted as σ_1/σ_2) represents that a transition on label σ_1 in the simulated system can be simulated by a transition on label σ_2 in the simulating system with the accompanying cost. We also require that each word over symbols of the form σ/σ is assigned cost 0 to ensure that correct simulations have a cost 0.

Given an error model $M = \langle S^e, \mathcal{O} \times \mathcal{O}, E^e, s^e_0 \rangle$ with weight function ν and an ATS $\mathcal{A} = \langle S, \Sigma, E, s_0, (S_1, S_2) \rangle$, the *modified system* is the weighted transition system $\mathcal{A}_M = \langle S \times S^e, \Sigma, E^M, (s_0, s^e_0), (S^M_1, S^M_2) \rangle$ with weight function ν^e where:

- $((s, s^e), \sigma_1, (t, t^e)) \in E^M \Leftrightarrow s \in S_2,\ (s, \sigma_2, t) \in E \wedge (s^e, (\sigma_1, \sigma_2), t^e) \in E^e$,
- $((s, s^e), \sigma_1, (t, s^e)) \in E^M \Leftrightarrow s \in S_1 \wedge (s, \sigma_1, t) \in E$; and
- $\nu^e(((s, s^e), \sigma_1, (t, t^e))) = \nu((s^e, (\sigma_1, \sigma_2), t^e))$.

The modified transition system includes erroneous behaviors along with their costs. Note that we do not consider errors in the inputs as all our reactive systems are input enabled. We present a few natural error models here.

- *Standard Model.* (Figure 2a) Every replacement can occur during simulation with a constant cost and can be used to model errors like bit-flips. This model was defined and used in [6].
- *Delayed Response Model.* (Figure 2b) This model measures the timeliness of responses (g). Here, when the implementation outputs $\tilde{g}$ when a grant g is expected, all transitions have a penalty until the missing grant g is seen.
- *No Spurious Response Model.* (Figure 2c) This model is meant to ensure that no spurious grants are produced. If an implementation produces a grant not required by the specification, all subsequent transitions get a penalty.
- *Qualitative Model.* (Figure 5) This model recovers the boolean simulation games. The distance is 0 if and only if the simulation relation holds.
- *Delayed-vs-Spurious Responses Model.* (Figure 3) We formalize the informal preference conditions from the discussion of the motivating example in Section 1 in the this error model. Delayed and spurious grants get penalties of d and s, respectively.

Of the above models, the standard model and the qualitative model can be expressed as modification schemes from [6], whereas the delayed response, the spurious response model, and the Delayed-vs-Spurious response model cannot be cast as modification schemes.

Quantitative Simulation Games. Given a modified specification system $\mathcal{A}'^M$ and an implementation system $\mathcal{A}$, the quantitative simulation game $\mathcal{Q}^{\mathcal{A},\mathcal{A}'}_M$ is a game with the game-graph of $\mathcal{G}^{\mathcal{A},\mathcal{A}'_\mathcal{M}}$, and a weight function that maps:

- Each Player 1 transition to 0, and
- Each Player 2 transition to 4 times weight of the corresponding transition in $\mathcal{A}'_M$ (The constant 4 is for normalization). The objective of the game for Player 1 is to maximize the *LimAvg* of weights.

In a quantitative simulation game, the implementation system is simulated by the modified specification system and for each simulation error, penalty is decided by the error model. The value of the quantitative simulation game $\mathcal{Q}^{\mathcal{A},\mathcal{A}'}_M$ is the *simulation distance* (denote as $d_M(\mathcal{A}, \mathcal{A}')$).

$d = 2, s = 1$	$\mathcal{S}_1$	$\mathcal{S}_2$
$\mathcal{I}_1$	0	2
$\mathcal{I}_2$	1	1
$\mathcal{I}_3$	1	1

$d = 1, s = 10$	$\mathcal{S}_1$	$\mathcal{S}_2$
$\mathcal{I}_1$	0	1
$\mathcal{I}_2$	10	10
$\mathcal{I}_3$	$\frac{1}{2}$	$\frac{1}{2}$

Table 1: Simulation distances for Example 2.1

Example 2.1. *Consider the specifications and implementations from the motivating example in Section 1 (Figure 1), and the Delayed-vs-Spurious Response Model in Figure 3.*

By varying the penalties, we obtain different distances. The simulation distances of implementations $\mathcal{I}_1$, $\mathcal{I}_2$ and $\mathcal{I}_3$ to specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ (Figure 1) are summarized in Table 1 for two valuations of d and s. For example, when $d = 2$ and $s = 1$, $\mathcal{I}_2$ and $\mathcal{I}_3$ have equal distances to $\mathcal{S}_1$ and $\mathcal{S}_2$. However, when $d = 1$ and $s = 10$, the distances of $\mathcal{I}_3$ are lower than that of $\mathcal{I}_2$.

Note that in the example of Figure 1, the specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ specify what happens if the two requests r_1 and r_2 come at the same time. If we wanted to change this example and make the specifications completely independent (i.e. $\mathcal{S}_1$ only mentions r_1 and $\mathcal{S}_2$ only mentions r_2), we would need to modify our notion of product construction to allow two input symbols at the same time. Our results would still hold for this modified product construction.

2.2 Properties of Simulation Distances

Directed metric requires reflexivity and triangle inequality to hold. These are perhaps the minimal requirements that allow us to match the intuitive notion of distance. In [6], it is shown that simulation distances with restricted error models are directed metrics.

For our general error models, reflexivity follows from definition. However, the triangle inequality does not hold for all models. We provide a necessary and sufficient condition for the triangle inequality to hold.

An error model M is *transitive* if the following holds: For every triple of infinite lasso (i.e., ultimately periodic) words generated by the error model M of the form $\alpha = \frac{a_0}{b_0}\frac{a_1}{b_1}\ldots$, $\beta = \frac{b_0}{c_0}\frac{b_1}{c_1}\ldots$ and $\gamma = \frac{a_0}{c_0}\frac{a_1}{c_1}\ldots$, the $LimAvg(\alpha) + LimAvg(\beta) \geq LimAvg(\gamma)$.

Lemma 2.2. *An error model M is transitive if and only if $\forall \mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3 : d_M(\mathcal{S}_1, \mathcal{S}_3) \leq d_M(\mathcal{S}_1, \mathcal{S}_2) + d_M(\mathcal{S}_2, \mathcal{S}_3)$.*

The proof of the triangle inequality for transitive error models can be derived from the proof of Theorem 1 in [6] with slight modifications. For a non-transitive error model, the three systems whose only behavior outputs the lasso-words which witness the non-transitivity violate the triangle inequality. All the error models from Figures 2 and 3 are transitive.

Theorem 2.3. *The simulation distance d_M is a directed metric if and only if the error model M is transitive.*

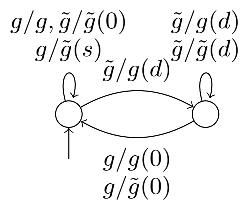


Figure 3: Delayed-vs-spurious response error model

The transitiveness of an error model can be checked in polynomial time.

Proposition 2.4. *It is decidable in polynomial time whether an error model $M = \langle S^e, \mathcal{O} \times \mathcal{O}, E^e, s_0^e \rangle$ is transitive.*

The result follows by constructing the product $M \times M \times M$ where the transitions with labels $\frac{\sigma_1}{\sigma_2}$, $\frac{\sigma_2}{\sigma_3}$ and $\frac{\sigma_1}{\sigma_3}$ and weights w_1, w_2 and w_3 are replaced with a transition of weight $w_1 + w_2 - w_3$. The model M is transitive iff there is no negative cycle in this graph (checkable in polynomial time).

3. THE INCOMPATIBLE SPECIFICATIONS PROBLEM

Specifications $\mathcal{S}_i$ and error models M_i for $1 \leq i \leq k$ are said to be *incompatible* if $\neg\exists\mathcal{I} : \bigwedge_i d_{M_i}(\mathcal{I}, \mathcal{S}_i) = 0$. Note that our definition may judge specifications compatible, even if there is no common implementation which is simulated classically by each specification. This happens if there exists an implementation with the distance 0 to each of the specifications, which is possible if the specifications share long-term behavior, but differ in the short-term initial behavior.

We formalize the synthesis from incompatible specifications problem as follows.

Incompatible Specifications Decision Problem. Given $\mathcal{S}_i$ and M_i for $1 \leq i \leq k$ as above, and a threshold vector $\mathbf{v} = \langle v_1, v_2, \ldots v_k \rangle \in \mathbb{Q}^k$, incompatible specifications decision problem asks if $\exists\mathcal{I} : \forall 1 \leq i \leq k : d_{M_i}(\mathcal{I}, \mathcal{S}_i) \leq v_i$.

Incompatible Specifications Optimization Problem. Given specifications $\mathcal{S}_i$ and error models M_i for $1 \leq i \leq k$ and a bound $\epsilon > 0$, the incompatible specifications optimization problem is to find an implementation $\mathcal{I}^*$ such that $\forall\mathcal{I}$: $\max_{i \in \{1,2,\ldots k\}} d_{M_i}(\mathcal{I}^*, \mathcal{S}_i) \leq \max_{i \in \{1,2,\ldots k\}} d_{M_i}(\mathcal{I}, \mathcal{S}_i) + \epsilon$. We call such an implementation $\mathcal{I}^*$ an ϵ-optimal implementation.

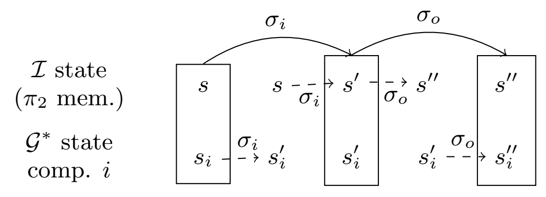


Figure 4: Working of π_2: Solid edges are transitions in $\mathcal{G}^*$ and dashed edges are transitions in $d_{M_i}(\mathcal{I}, \mathcal{S}_i)$

Theorem 3.1. *The incompatible specifications decision problem is* coNP*-complete for a fixed k.*

Proof. First, we prove that the incompatible specifications decision problem is in coNP. We reduce the problem to the decision problem in 2-player games with *MLimAvg* objectives.

Given a specification $\mathcal{S}_i$ and an error model M_i for $1 \leq i \leq k$, consider the following game graph $\mathcal{G}^*$ with:

- Player 1 states $S_1 = S_1^1 \times \ldots \times S_1^k$ where S_1^i are the Player 1 states of $\mathcal{S}_i^{M_i}$;
- Player 2 states $S_2 = S_2^1 \times \ldots \times S_2^k$ where S_2^i are the Player 2 states of $\mathcal{S}_i^{M_i}$;
- A transition from state $(s_1, s_2, \ldots, s_k)$ to $(s'_1, s'_2, \ldots, s'_k)$ on symbol σ if and only if each of $(s_i, \sigma, s'_i) \in E^i$ where E^i is the transition set of $\mathcal{S}_i^{M_i}$; and
- Weight function ν with the i^{th} component being $\nu^i((s_1, s_2, \ldots, s_k), \sigma, (s'_1, s'_2, \ldots, s'_k)) = 2 * \nu((s_i, \sigma, s'_i))$ for $1 \leq i \leq k$.

Intuitively, Player 1 chooses the inputs and Player 2 chooses output transitions from $\mathcal{S}_i^{M_i}$. We prove that a witness implementation exists if and only if there exists a finite memory Player 2 strategy in $\mathcal{G}^*$ for the *MLimAvg* objective.

(a) For any implementation $\mathcal{I}$, consider the games $\mathcal{Q}_{M_i}^{\mathcal{I},\mathcal{S}_i}$ and the optimal Player 2 strategy π_2^i in each. By standard results on *LimAvg* games, we have that each π_2^i is memoryless. From these strategies, we construct a finite-memory Player 2 strategy π_2 in $\mathcal{G}^*$ with the state space of $\mathcal{I}$ as the memory. The memory update function of π_2 mimics the transition relation of $\mathcal{I}$. Let s be the current state of π_2 memory and let $(s_1, s_2, \ldots, s_k)$ be the current state in $\mathcal{G}^*$. By construction, s is Player 1 state in $\mathcal{I}$ iff $(s_1, \ldots, s_k)$ is Player 1 state in $\mathcal{G}^*$.

- If $(s_1, s_2, \ldots, s_k)$ is a Player 1 state, Player 1 chooses an input symbol $\sigma_i \in \mathcal{I}$ and updates the $\mathcal{G}^*$ state. The memory of π_2 is updated to s' which is the unique successor of s on σ_i.
- Next, if the current state $(s'_1, s'_2, \ldots, s'_k)$ is a Player 2 state, the memory of π_2 is updated to the unique successor s'' of s' in $\mathcal{I}$ (Player 2 states have unique successors in implementations). If (s', σ, s'') is the corresponding $\mathcal{I}$ transition, the chosen $\mathcal{G}^*$ state is $(s''_1, s''_2, \ldots, s''_k)$ where each $s''_i = \pi_2^i((s'', \sigma, s'_i))$.

The construction of π_2 is explained in Figure 4.

For every path ρ conforming to π_2, we can construct a path ρ^i in $\mathcal{Q}_{M_i}^{\mathcal{I},\mathcal{S}_i}$ conforming to π_2^i from the memory of π_2 and the projection of ρ to i^{th} component (See Figure 4). Furthermore, the weights of the i^{th} component of ρ have the same *LimAvg* as the weights of ρ^i. Therefore, the *LimAvg* value of the i^{th} component of ρ is bound by $d_{M_i}(\mathcal{I}, \mathcal{S}_i)$. This

shows that the *MLimAvg* value of π_2, $Val(\pi_2)$ is at most the maximum of $d_{M_i}(\mathcal{I}, \mathcal{S}_i)$.
(b) For every finite-memory strategy π_2 of Player 2 in $\mathcal{G}^*$, we can construct an implementation $\mathcal{I}$ such that $Val(\pi_2) \geq \max_{i\in\{1,2,\dots k\}} d_{M_i}(\mathcal{I}, \mathcal{S}_i)$, by considering the product of $\mathcal{G}^*$ and the memory of π_2 and by removing all transitions originating from Player 2 states which are not chosen by π_2.

From the results of [7], we have that solving *MLimAvg* games for the threshold $\{0\}^k$ for finite memory strategies is coNP-complete. However, we can reduce the problem of solving *MLimAvg* games for a threshold $\mathbf{v} \in \mathbb{Q}^k$ to a problem with threshold $\{0\}^k$ by subtracting $\mathbf{v}$ from each of the edge weights. This reduction is obviously polynomial. Therefore, the inconsistent specifications decision problem can be solved in coNP time in the size of $\mathcal{G}^*$, which in turn is polynomial in the size of the input for fixed k. To show the coNP hardness, we can use a modification of the proof of coNP hardness of *MLimAvg* games by reduction from the complement of 3-SAT. □

Now, we can find an ϵ-optimal implementation for the optimization problem by doing a binary search on the space of thresholds.

Corollary 3.2. *The incompatible specifications optimization problem can be solved in* NEXP *time for a fixed k, ϵ and W, where W is the absolute value of the maximum cost in the error models.*

Proof. Without loss of generality, let $\epsilon = \frac{1}{q}$ for $q \in \mathbb{N}$. As the simulation distances are between 0 and W, we do a binary search on vectors of form $\{t\}^k$ to find $\{N/Wq\}^k$, the highest threshold for which an implementation exists. Since, the accuracy required is ϵ, the number of search steps is $O(\log(W/\epsilon)) = O(\log(Wq))$. We find an implementation (equivalently, a Player 2 finite memory strategy) with a value of at least this threshold. We reduce the problem to an equivalent threshold problem with integer weights and threshold $\{0\}^k$ by multiplying weights by Wq and subtracting $\{N\}^k$. From [7], we have that memory of size $O(|\mathcal{G}^*|^2 \cdot (|\mathcal{G}^*|qW)^k)$ is sufficient. For one such finite-memory strategy, checking whether it is sufficient can be done in polynomial time. Therefore, by guessing a strategy and checking for sufficiency, we have an NEXP time algorithm. □

Example 3.3. *For the specifications and error models from Example 2.1 (Section 2.1), we compute ϵ-optimal implementations for different values of d and s. In this case, we obtain optimal implementations by taking a small enough ϵ. We have that $\mathcal{I}_2$ is one of the optimal implementations for the case of $d = 2$ and $s = 1$. The implementation $\mathcal{I}_3$ and its dual (i.e., r_1, g_1 interchanged with r_2, g_2) are optimal for the case when $d = 1$ and $s = 10$. As expected, $\mathcal{I}_2$, which outputs many spurious grants, is worse when the cost of a spurious grant is higher.*

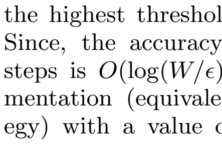


Figure 5: Qualitative error model

For the qualitative error model (Figure 5) and any set of incompatible specifications, for all implementations the distance to at least one of the specifications is ∞. However, for the standard error model of [6], we show for the case of two specifications that it always is possible to do better.

Proposition 3.4. *For specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ with the standard error model M, let $\delta = \min(d_M(\mathcal{S}_1, \mathcal{S}_2), d_M(\mathcal{S}_2, \mathcal{S}_1))$. There exists an implementation $\mathcal{I}^*$ with $d_M(\mathcal{I}^*, \mathcal{S}_1) < \delta$ and $d_M(\mathcal{I}^*, \mathcal{S}_2) < \delta$.*

Proof. Without loss of generality, let $d_M(\mathcal{S}_1, \mathcal{S}_2) \leq d_M(\mathcal{S}_2, \mathcal{S}_1)$. Consider the game graph of $\mathcal{Q}_M^{\mathcal{S}_1,\mathcal{S}_2}$ and modify it by letting Player 2 choose the $\mathcal{S}_1$ output transitions, i.e., Player 1 chooses the inputs and Player 2 chooses both $\mathcal{S}_1$ and $\mathcal{S}_2$ outputs. Let π_2^* be the optimal Player 2 strategy in this game. From π_2^*, we construct two different implementations $\mathcal{I}_1$ and $\mathcal{I}_2$ having as the state space the product of the state spaces of $\mathcal{S}_1$ and $\mathcal{S}_2$. In the transition set,

- There exists an input transition from (s_1, s_2) to (s_1', s_2') on the input symbol σ_i if and only if (s_1, σ_i, s_1') and (s_2, σ_i, s_2') are input transitions of $\mathcal{S}_1$ and $\mathcal{S}_2$;
- There exists an output transition $((s_1, s_2), \sigma_o, (s_1', s_2'))$ in $\mathcal{I}_1$ iff π_2^* chooses the $\mathcal{S}_1$ transition (s_1, σ_o, s_1') from state $(s_1, \#, (s_2, e))$ and the $\mathcal{S}_2^M$ transition (s_2, σ_o, s_2') in state $(s_1', \sigma_o, (s_2, e))$; and
- There exists an output transition $((s_1, s_2), \sigma_o, (s_1', s_2'))$ on σ_o in $\mathcal{I}_2$ iff π_2^* chooses the $\mathcal{S}_1$ transition (s_1, σ_o', s_1') from state $(s_1, \#, (s_2, e))$ and the $\mathcal{S}_2^M$ transition (s_2, σ_o', s_2') in state $(s_1', \sigma_o', (s_2, e))$ and $\mathcal{S}_2^M$ transition corresponds to the $\mathcal{S}_2$ transition (s_2, σ_o, s_2') and the error model transition $(e, \sigma_o'/\sigma_o, e)$.

Intuitively, π_2^* chooses the most benevolent $\mathcal{S}_1$ behavior and $\mathcal{I}_1$ implements this $\mathcal{S}_1$ behavior, while $\mathcal{I}_2$ is the $\mathcal{S}_2$ behavior used to simulate this game.

Now, we construct $\mathcal{I}^*$ by alternating between $\mathcal{I}_1$ and $\mathcal{I}_2$. For each Player 1 state (s_1, s_2) in $\mathcal{I}_i$, let $TU((s_1, s_2))$ be the tree unrolling of $\mathcal{I}_i$ from (s_1, s_2) to a depth $N \in \mathbb{N}$ and let $\mathcal{T}(\mathcal{I}_i)$ be the disjoint union of such trees. Let $\mathcal{I}^*$ be the union of $\mathcal{T}(\mathcal{I}_1)$ and $\mathcal{T}(\mathcal{I}_2)$ where each transition to a leaf state (s_1, s_2) in $\mathcal{T}(\mathcal{I}_1)$ is redirected to the root of $TU((s_1, s_2))$ in $\mathcal{T}(\mathcal{I}_2)$, and vice versa.

We now show that $d_M(\mathcal{I}^*, \mathcal{S}_i) < \delta$. Consider the Player 2 strategy π_2 in $\mathcal{Q}_M^{\mathcal{I}^*,\mathcal{S}_2}$: to simulate an $\mathcal{I}^*$ transition from (s_1, s_2) to (s_1', s_2') on σ_o, π_2 chooses the $\mathcal{S}_2^M$ transition $((s_2, e), \sigma_o, (s_2', e))$. If $((s_1, s_2), \sigma_o, (s_1', s_2'))$ was from $\mathcal{T}(\mathcal{I}_2)$, the cost of the simulation step is 0, and otherwise it is equal to the corresponding transition from $\mathcal{Q}_M^{\mathcal{S}_1,\mathcal{S}_2}$. Now, fix π_2 in $\mathcal{Q}_M^{\mathcal{I}^*,\mathcal{S}_2}$ and let C be the cycle of the path obtained by fixing the optimal Player 1 strategy. Cycle C is composed of paths through $\mathcal{I}_1$ and $\mathcal{I}_2$ each of length N. The cost of the path through $\mathcal{I}_2$ is 0. The cost of the path through $\mathcal{I}_1$ is equal to the cost of the corresponding cycle in $\mathcal{Q}_M^{\mathcal{S}_1,\mathcal{S}_2}$. If N is large enough, the path through $\mathcal{I}_1$ is composed of an acyclic part of length at most $n = 2 \cdot |\mathcal{Q}_M^{\mathcal{S}_1,\mathcal{S}_2}|$ and of cyclic paths of average cost less than $d_M(\mathcal{S}_1, \mathcal{S}_2) = \delta$. Therefore, for all $\epsilon > 0$ and $N > \frac{nW}{\epsilon}$ we have

$$d_M(\mathcal{I}^*, \mathcal{S}_2) \leq Val(\pi_2) \leq \frac{(N-n)\cdot\delta + n\cdot W}{2N} \leq \frac{\delta}{2} + \epsilon < \delta$$

Similarly, we can show $d_M(\mathcal{I}^*, \mathcal{S}_1) < \delta$ to complete the proof. □

4. QUANTITATIVE ALPHABETS

We have handled the case of controller synthesis from incompatible situations where all controlled variables are from a finite domain. However, often in practice, the controlled variable can take values from an infinite continuous domain. For example, an output representing voltage might take any value between 0 volts to 5 volts. Furthermore, in many cases, although the underlying controlled variable ranges over a continuous domain, due to practical reasons, the only viable controllers are the ones which have discretized outputs, i.e., allow the variable to assume values from a finite subset of the domain. In the case of voltages, it might be the case that the controller can only choose the values from one of 0 volts, 1 volt, 3 volts and 5 volts. Here we present results on synthesis for incompatible specifications with the controlled output variables ranging either over a continuous domain, or a finite discretization of a continuous domain.

Firstly, the notion of simulation distances can be generalized to systems with output alphabets that are infinite and quantitative. The assumptions about the output alphabet are that: (a) it is a metric space with the metric $m_{\mathcal{O}}$; and (b) it has a *midpoint function*, i.e., there exists a computable function $\mu : \mathcal{O} \times \mathcal{O} \to \mathcal{O}$ such that $\forall o_1, o_2 \in \mathcal{O} : m_{\mathcal{O}}(o_1, \mu(o_1, o_2)) = m_{\mathcal{O}}(o_2, \mu(o_1, o_2)) = \frac{m_{\mathcal{O}}(o_1, o_2)}{2}$. For example, any interval $I \subseteq \mathbb{R}$ with $m_I(a, b) = |a - b|$ and the midpoint function $\mu_I(a, b) = \frac{a+b}{2}$ satisfies our assumptions.

For simplicity, we consider only the standard error model from Section 2 (constant penalty for each mismatch), with the cost of an edge labeled σ/σ' being $m_{\mathcal{O}}(\sigma, \sigma')$. The resultant simulation distance is a directed metric.

4.1 Continuous Control Problem

For quantitative outputs with continuous control, i.e., when the controller can choose the output values from the whole continuous domain, the problem of synthesis from incompatible specifications can be solved with a simple construction. Intuitively, the reason is that when the two specifications differ, we can always find a "middle ground", due to our assumption on the existence of the midpoint function μ. The following theorem states that we can obtain an implementation which is no farther from either of the specification than half the original distance between the specifications.

Theorem 4.1. *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be specifications over a finite input alphabet $\mathcal{I}$ and a quantitative output alphabet $\mathcal{O}$. Let $\delta = \min(d(\mathcal{S}_1, \mathcal{S}_2), d(\mathcal{S}_2, \mathcal{S}_1))$. There exists an implementation $\mathcal{I}^*$ such that:*

- $\forall \mathcal{I} : \max_{i \in \{1,2\}} d(\mathcal{I}^*, \mathcal{S}_i), \leq \max_{i \in \{1,2\}} d(\mathcal{I}, \mathcal{S}_i)$;
- $d(\mathcal{I}^*, \mathcal{S}_1) \leq \frac{\delta}{2}$ *and* $d(\mathcal{I}^*, \mathcal{S}_2) \leq \frac{\delta}{2}$; *and*
- *The number of states of $\mathcal{I}^*$ is in $O(n_1 n_2)$ where n_i is the number of states of $\mathcal{S}_i$.*

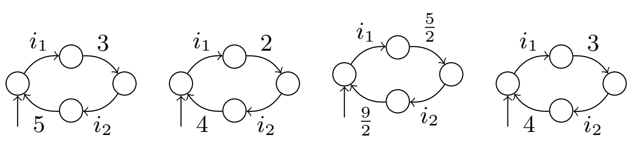
Proof Idea. Consider the game graph $\mathcal{G}^*$ whose states are the product of the state space of $\mathcal{S}_1$ and $\mathcal{S}_2$. For transitions (s_1, σ_1, s_1') and (s_2, σ_2, s_2') in $\mathcal{S}_1$ and $\mathcal{S}_2$, $\mathcal{G}^*$ contains the transition $((s_1, s_2), \mu(\sigma_1, \sigma_2), (s_1', s_2'))$ having the weight $m(\sigma_1, \sigma_2)$. We can show that the optimal Player 2 strategy to minimize the *LimAvg* of the weights in this game corresponds to the implementation $\mathcal{I}^*$.

4.2 Discretized Control Problem

In the case where a continuous output variable is forced to take discrete values, the problem of synthesis from incompatible specifications is considerably more complex.

We call a finite set $\mathcal{O}' \subseteq \mathcal{O}$ a *discretization* of $\mathcal{O}$. For any $o \in \mathcal{O}$, we let $o|\mathcal{O}' = \{o' \in \mathcal{O}' | \forall o'' \in \mathcal{O}' m(o, o') \leq m(o, o'')\}$.

Now, we make a further assumption about the metric-space of outputs $\mathcal{O}$ and the encoding of transitions in the specifications. For a pair of states, s and s' in the specification, let $\Delta(s, s') \subseteq \mathcal{O}$ be the set of outputs o such that



(a) $\mathcal{S}_1$ (b) $\mathcal{S}_2$ (c) $\mathcal{M}$ (d) $\mathcal{P}$

Figure 6: Example for synthesis using Projection Theorem

a transition (s, o, s') exists in the specification. We assume that for every $o' \in \mathcal{O}'$ and for every pair of states s, s', the value $\min_{o \in \Delta(s,s')} m(o', o)$ is computable. This is the case for example if the output alphabet represents a real-valued variable and the transitions in the specifications are given in terms of intervals.

We remark that due to the assumption on computability, we can use the synthesis algorithms for finite alphabets from Section 3 for the discretized control problem. However, we present a more efficient polynomial heuristic. It is based on the following theorem for solving the discretized synthesis problem for a single specification, i.e., finding the optimal discretized implementation for a single quantitative specification.

Theorem 4.2 (Projection). *Given a specification $\mathcal{S}$ over the output alphabet $\mathcal{O}$, and a discretization $\mathcal{O}' \subseteq \mathcal{O}$, there exists an implementation $\mathcal{I}^*$ over the output alphabet $\mathcal{O}'$ such that for all implementations $\mathcal{I}$ over the $\mathcal{O}'$, $d(\mathcal{I}^*, \mathcal{S}) \leq d(\mathcal{I}, \mathcal{S})$. Furthermore, the number of states of $\mathcal{I}^*$ is $\mathcal{O}(n)$, where n is the number of states of $\mathcal{S}$.*

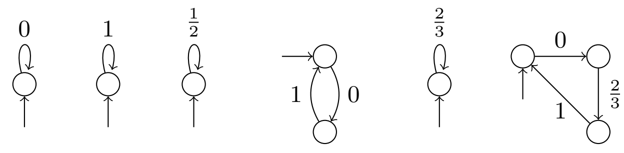
Proof Idea. Consider the game $\mathcal{G}^*$ with the same state space as $\mathcal{S}$ and each transition of $\mathcal{S}$ with label o has been replaced with a transition with label o' such that $o' \in o|\mathcal{O}'$. The weight of such a transition is $m(o, o')$. It is easy to show that $\mathcal{I}^*$ corresponds to an optimal Player 2 strategy to minimize the *LimAvg* of the weights in $\mathcal{G}^*$.

Heuristic Synthesis using Projection For incompatible specifications, the algorithms for the finite alphabet case have a high complexity. However, Theorems 4.1 and 4.2 suggest a simple (albeit non-optimal) construction for the case of discretized alphabets. Given specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ with output alphabet $\mathcal{O}$ or $\mathcal{O}'$, using Theorem 4.1, one could synthesize an optimal implementation over $\mathcal{O}$, and then use Theorem 4.2 to get the best approximation over the finite alphabet $\mathcal{O}'$. Though there is no guarantee that we get the optimal implementation, it can serve as a good heuristic.

In Figure 6, implementation $\mathcal{P}$ (on the discretized $\mathcal{O}' = \{2, 3, 4, 5\} \subseteq \mathbb{R} = \mathcal{O}$ is synthesized from $\mathcal{S}_1$ and $\mathcal{S}_2$ using this method. The system $\mathcal{M}$ is the optimal implementation for outputs in $\mathbb{R}$ produced by Theorem 4.1 and $\mathcal{P}$ is a projection of $\mathcal{M}$ to the discretization $\mathcal{O}'$. In this case, the projection happens to be the optimal system for discretized outputs.

We leave two questions open: (i) characterizing the implementation obtained in this way w.r.t. an implementation obtained by synthesis for the finite alphabet directly, and (ii) deriving heuristics for choosing a projection in case there are several projections available (e.g. $\mathcal{P}$ is only one of the possible closest projections of $\mathcal{M}$ to $\mathcal{O}'$).

To illustrate the second question, consider specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ from Figure 7. The implementation $\mathcal{I}_1$ is given by the construction of Theorem 4.1. Any of $\mathcal{S}_1$, $\mathcal{S}_2$ or $\mathcal{P}_1$ can serve as a projection of $\mathcal{I}_1$ on the alphabet $\{0, 1\}$. However, for $\mathcal{S}_1$ and $\mathcal{S}_2$, the distance to the other specification



(a) $\mathcal{S}_1$ (b) $\mathcal{S}_2$ (c) $\mathcal{I}_1$ (d) $\mathcal{P}_1$ (e) $\mathcal{P}_2$ (f) $\mathcal{I}_2$

Figure 7: Incompatible Specifications and Projection

is 1, whereas for $\mathcal{P}_1$, the distance to both specifications is $\frac{1}{2}$. Hence, $\mathcal{P}_1$ is the better (in fact, optimal) projection.

Furthermore, the optimal solution might not be a projection of the one obtained from the construction in Theorem 4.1. For $\mathcal{S}_1$ and $\mathcal{S}_2$, instead of the discretization $\{0, 1\}$, consider the discretization $\{0, \frac{2}{3}, 1\}$. Implementation $\mathcal{P}_2$ (a projection of $\mathcal{I}_1$) has distances $\frac{2}{3}$ and $\frac{1}{3}$ to $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively. However, $\mathcal{I}_2$ (which is not a projection of $\mathcal{I}_1$) has distances $\frac{4}{9}$ and $\frac{5}{9}$ to $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively.

5. CASE STUDIES

We present two case studies to demonstrate the use of simulation distances for modeling conflicting requirements. These case studies do not consider large-scale examples, but rather serve to demonstrate that simulation distances and the synthesis from incompatible specifications framework are in principle suitable for specifying real-world problems.

5.1 Case study: Synthesis of Forward Error Correcting Codes

Consider the task of sending messages over an unreliable network. Forward Error Correcting codes (FECs) are encoding-decoding schemes that are tolerant to bit-flips during transmission, i.e., the decoded message is correct in-spite of errors during transmission. For example, the well-known Hamming (7,4) code can correct any one bit-flip that occurs during the transmission of a bit-block. The Hamming (7,4) code transmits 7 bits for every 4 data bits to be transfered, and the 3 additional bits are parity bits.

Suppose bit-blocks of length 3 are to be transfered over a network where at most 1 bit-flip can occur during transmission. We want to minimize the number of transmitted bits. Furthermore, we also allow some errors in the decoded block. Therefore, we have two incompatible specifications:

- *Efficiency.* To minimize the number of bits transmitted, we add a requirement that only 3 bits are transmitted and an error model that has a constant penalty of e for each additional bit transmitted.
- *Robustness.* We want the decoded block to be as correct as possible. In a standard FEC scheme, all bits are given equal importance. However, to demonstrate the flexibility of our techniques, we consider the first bit to be the most significant one, and the third to be the least significant one. We add a requirement that the decoded bit block is the same as the original, with the following error model: An error in the first, second, and third bit have a cost of $4d$, $2d$, and d, respectively.

Formal modeling. The output and input alphabets are $\{T_0, T_1, R_0, R_1, O_0, O_1, \bot\}$ and $\{I_0, I_1, F, \neg F, \bot\}$ where T_i, R_i, I_i and O_i stand for transmission, receiving, input and output of bit i respectively. Symbols F and $\neg F$ denote if a bit-flip occurs or not during the current transmission. Symbol $\bot$ is used whenever the input/output does not matter.

EXAMPLE 5.1. *For example, the diagram below represents the transmission of bit-block* 010 *through a system without any error correction.*

In	I_0	I_1	I_0	$\bot$	F	$\bot$	$\neg F$	$\bot$	$\neg F$	$\bot$	$\bot$	$\bot$
Out	$\bot$	$\bot$	$\bot$	T_0	R_1	T_1	R_1	T_0	R_0	O_1	O_1	O_0

First, three bits are input. Next, each of the three bits is transmitted and received. The environment decides that the first bit is flipped and the value received is 1 *even though* 0 *is transmitted. Finally, the bit block* 110 *is output.*

In addition to *Efficiency* and *Robustness* requirements above, we need the following. For these, we use the qualitative error model where even a single error is not allowed.

- *Encoding and Decoding.* For any input (resp., received) bit-block, the same sequence of bits should be transmitted (resp. output). The specification remembers the transmitted (resp., output) bits for each input (resp., transmitted) bit-block and ensures that the same bits are transmitted (resp., output) in the future.
- *Reception.* The received bit should be correctly flipped or not based on whether the input is F or $\neg F$.

Results. For different relative values of efficiency penalty e and robustness penalty d, different optimal FEC schemes are obtained. Suppose $b_1 b_2 b_3$ is the input bit-block.

- $e = 1 \wedge d = 100$. The implementation is fully robust, i.e., always outputs the right bit-block. For example, one of the optimal strategies transmits b_1, b_2, b_3, $b_2 \oplus b_3$, $b_1 \oplus b_3$ and $b_1 \oplus b_2$. The bit-block can always be recovered from the received bits. This has a total error of 3 for efficiency and 0 for robustness per round.
- $e = 100 \wedge d = 1$. The implementation transmits only the three input bits and in the worst case outputs the most significant bit wrong. The worst-case errors are 0 for efficiency and 4 for robustness per round.
- $e = 10 \wedge d = 10$. The implementation ensures the correctness of the most significant bit by transmitting it thrice (triple modular redundancy), i.e., transmits b_1, b_1, b_1, b_2 and b_3. In the worst case, the second bit is output wrong and the error for efficiency is 20 and for robustness is 20 per round.

These results show how we can obtain completely different FECs just by varying the costs in the error models.

5.2 Case study: Optimal Scheduling for Overloads

Consider the task of scheduling on multiple processors, where processes have definite execution times and deadlines. Deadlines are either "soft", where a small delay is acceptable, but undesirable; or "hard", where any delay is catastrophic. During overload, processes are either delayed or dropped completely; and usually these processes are chosen based on priorities. Our techniques can be used to schedule based on exact penalties for missing deadlines or dropping processes.

Each process repeatedly requests execution and scheduling is based on time-slices with each processor executing a single process in a time-slice. A process $\mathcal{P}(t, d, c)$ represents:

- the time-slices t needed for the computation;
- the deadline d from invocation time; and

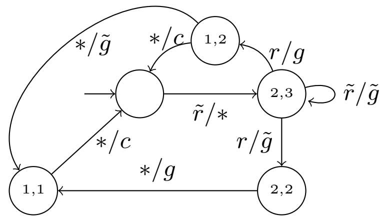


Figure 8: Modelling processes: $\mathcal{P}(2,3,1)$

- the minimum time c between the completion of one invocation and the next request.

We model a process as a reactive system with inputs $\{r, \tilde{r}\}$ and outputs $\{g, \tilde{g}, c\}$. The input r represents an invocation, the output g represents a single time-slice of execution, and the output c indicates completion of the invocation.

In Figure 8, all states (except the initial) are labeled by two numbers (t, d) representing, respectively, remaining execution steps, and time to deadline. Once request r is issued, execution starts at the state labeled $(2, 3)$ (input and output transitions are drawn together for readability). If the first time slice is granted, the execution goes to state $(2, 1)$ (i.e., deadline in two steps, and one step of work remaining). If the time slice is not granted, the execution transitions to a state labeled by $(2, 2)$. The model (specification) ensures that the task is completed before the deadline. After it is completed, the control is in the initial state, where a request cannot be issued for at least one time step.

We define both hard and soft deadline error models. In the hard deadline error model, a missed deadline leads to a one-time large penalty p_l, whereas in the soft deadline error model, a small recurring penalty p_s occurs every step until the process is finished. Furthermore, we have a specification that no more than n processes can be scheduled in each step, with the qualitative failure model (Figure 5). We describe some optimal implementations obtained for various inputs.

- For two $\mathcal{P}(3, 6, 3)$ processes and one processor, we obtain a 0 cost schedule where each process is alternately scheduled till completion. This schedule is obtained independently of whether the deadlines are hard or soft.
- For $P_1 = \mathcal{P}(5, 5, 5)$, $P_2 = \mathcal{P}(3, 5, 5)$, and $P_3 = \mathcal{P}(2, 5, 5)$ with P_1 on a soft deadline (i.e. with the soft deadline error model described above), P_2 on a hard deadline, and P_3 on a hard deadline. With $p_s = 1$ and $p_l = 10$, we get a scheduler where P_2 and P_3 are treated as having a higher priority. Whenever P_2 or P_3 requests arrive, P_1 is preempted till P_2 and P_3 finish.
- For the same processes, but with $p_s = 5 \land p_l = 10$, we get a scheduler where P_1 is preferred over P_2 and P_3.

6. CONCLUSION

There are several possible directions for future research. From the theoretical side, we will investigate extending the simulation distances framework to probabilistic systems (to model probabilistically distributed inputs). The second possible extension is using bisimulation (as opposed to simulation) as the basis of distances between systems. From the practical side, we plan to do a larger case study to test whether quantitative metrics lead to simpler, more robust, or easier to maintain real specifications.

7. REFERENCES

[1] R. Alur, T. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In *CONCUR*, pages 163–178, 1998.

[2] R. Bloem, R. Cavada, I. Pill, M. Roveri, and A. Tchaltsev. Rat: A tool for the formal analysis of requirements. In *CAV*, pages 263–267, 2007.

[3] R. Bloem, K. Chatterjee, T. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In *CAV*, pages 140–156, 2009.

[4] P. Caspi and A. Benveniste. Toward an approximation theory for computerised control. In *EMSOFT*, pages 294–304, 2002.

[5] P. Černý, K. Chatterjee, T. Henzinger, A. Radhakrishna, and R. Singh. Quantitative synthesis for concurrent programs. In *CAV*, pages 243–259, 2011.

[6] P. Černý, T. Henzinger, and A. Radhakrishna. Simulation distances. In *CONCUR*, pages 253–268, 2010.

[7] K. Chatterjee, L. Doyen, T. Henzinger, and J.-F. Raskin. Generalized mean-payoff and energy games. In *FSTTCS*, pages 505–516, 2010.

[8] K. Chatterjee, T. Henzinger, and B. Jobstmann. Environment assumptions for synthesis. In *CONCUR*, pages 147–161, 2008.

[9] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, 1982.

[10] L. de Alfaro, M. Faella, and M. Stoelinga. Linear and branching system metrics. *IEEE Trans. Software Eng.*, 35(2):258–273, 2009.

[11] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled Markov processes. *TCS*, 318(3):323–354, 2004.

[12] S. Divakaran, D. D'Souza, and R. Matteplackel. Conflict-tolerant specifications in temporal logic. In *ICSE*, pages 103–110, 2010.

[13] D. D'Souza and M. Gopinathan. Conflict-tolerant features. In *CAV*, pages 227–239, 2008.

[14] C. Heitmeyer, M. Archer, R. Bharadwaj, and R. Jeffords. Tools for constructing requirements specifications: the SCR toolset at the age of nine. *Comput. Syst. Sci. Eng.*, 20(1), 2005.

[15] R. Könighofer, G. Hofferek, and R. Bloem. Debugging formal specifications using simple counterstrategies. In *FMCAD*, pages 152–159, 2009.

[16] I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. Formal analysis of hardware requirements. In *DAC*, pages 821–826, 2006.

[17] N. Piterman and A. Pnueli. Synthesis of reactive(1) designs. In *VMCAI*, pages 364–380, 2006.

[18] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989.

[19] F. van Breugel. An introduction to metric semantics: operational and denotational models for programming and specification languages. *TCS*, 258(1-2):1–98, 2001.