Synthesis from Incompatible Specifications

Pavol Černý[†], Sivakanth Gopi[*], Thomas A. Henzinger[†], Arjun Radhakrishna[†], and Nishant Totla[*]

[†]IST Austria [*]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. 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 provide several new properties of the directed metric induced by simulation distances on the space of finite-state machines, and we generalize the framework to the case of quantitative alphabets. Finally, we illustrate our approach using a case study on scheduler synthesis.

1 Introduction

A major problem for the wider adoption of techniques for the formal verification and synthesis of systems is the quality of available specifications. Quantitative specifications have the potential to simplify the task of the designer, by enabling her to better, and more simply, capture her intent. 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, [**?**,**?**,**?**] 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 are more difficult to maintain than the original requirements. In contrast, quantitative frameworks allow the formalization of the intuitive the idea that what is desired is an implementation that comes "closest" to satisfying the mutually incompatible requirements.

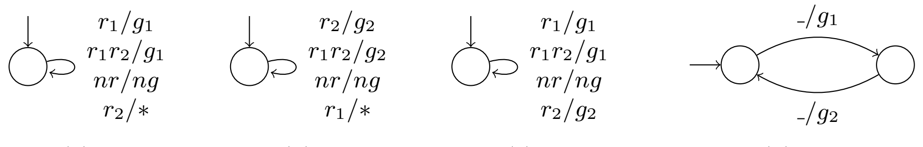
Synthesis has recently re-gained research interest [**?**,**?**,**?**] as a way to increase programmer productivity. The possible usefulness of synthesis is higher if the specifications remain at the high level. If the designer is forced to provide a detailed low-level specification to reconcile contradictory requirements, it decreases the usefulness of synthesis in two ways. First, it requires more effort from the designer, possibly 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 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. There are two processes periodically seeking access to the resource. The specification of the system consists of two parts: the first (resp. second) part states that a request from Process 1 (Process 2) for the resource 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 a 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. 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. In this paper, 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 extend the simulation distances framework of [**?**]. Simulation distances generalize the simulation relation (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 (the implementation) chooses moves (transitions) and Player 2 (the specification) tries to match each move. In order to generalize this definition to the quantitative setting, we allow the players to make errors, but they pay a certain prices for their choices of transitions. The cost of a trace is given by an objective function. We focus on the limit average objective (and thus long-term behavior of programs). The goal of Player 1 is trying to maximize the cost, and the goal of Player 2 is to minimize it. 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 [**?**] 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 given as finite-state machines, and allow specifying 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.

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) 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, in-



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

Fig. 1: Example 1

tuitively, (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 2, 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 $\mathcal{S}_1$ (and thus it has distance 0 from $\mathcal{S}_1$), but makes a mistake at every step w.r.t. $\mathcal{S}_2$ (and thus it is of distance 1 from $\mathcal{S}_2$). The implementation $\mathcal{I}_2$ handles the sequence $(r_1r_2)^*$ gracefully by alternation (note that _ in Figure 1d matches any input). However, on the input sequence $(nr)^*$, $\mathcal{I}_2$ 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 $(r_1r_2)^*$ and on this input sequence, it makes a mistake in every other step, w.r.t. $\mathcal{S}_1$ as well as $\mathcal{S}_2$.

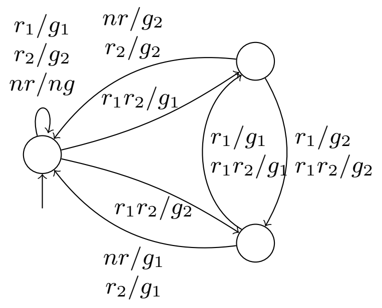


Fig. 2: Example 1: $\mathcal{I}_3$

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 [**?**]. 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 [**?**]), 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.

Furthermore, we extend the framework of simulation distances [**?**] 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 alphabets, where the alphabets themselves are metric spaces. We present a method for solving the problem of synthesis from incompatible specifications in this more flexible setting, and we prove a projection theorem that relates automata on quantitative alphabets to automata on finite alphabets. Finally, we demonstrate how our methods can enable simpler specifications, while allowing the synthesis of desirable solutions, using a case study 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 [**?**,**?**,**?**] and requirement debugging [**?**] were proposed. The problem of an inconsistent specification was approached in [**?**] 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 [**?**]. There have been several attempts to give a mathematical semantics to reactive processes based on quantitative metrics rather than boolean preorders [**?**,**?**]. In particular for probabilistic processes, it is natural to generalize bisimulation relations to bisimulation metrics [**?**], and similar generalizations can be pursued if quantities enter through continuous variables [**?**]. We consider distances between purely discrete (non-probabilistic, untied) systems. Synthesis from inconsistent specifications was considered in [**?**,**?**]. 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. Synthesis with respect to quantitative measures was considered in [**?**,**?**], but only in the context of consistent specifications, and not for simulation distances.

2 Behavioral Metrics on Automata

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 $\rho = \rho_0\sigma_0\rho_1 \dots$ 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:

- Alphabet Σ is the disjoint union of inputs $\mathcal{I}$ and outputs $\mathcal{O}$;
- Transition relation is strictly alternating on S_1 and S_2, and inputs and outputs, i.e., $E \subseteq (S_1 \times \mathcal{I} \times S_2) \cup (S_2 \times \mathcal{O} \times S_1)$;
- 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
- 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 a boolean or quantitative 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 $w \cdot s \in (S \times \Sigma)^* \cdot S_1$ and $\pi_i(w \cdot s) = (\sigma_1, s')$, 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 $\rho = \rho_0\sigma_0\rho_1\sigma_1 \dots$ *conforms* to a Player j strategy if $\forall i \geq 0 : \rho_i \in S_j \implies (\rho_{i+1}, \sigma_i) = \pi_1(\rho_0\sigma_0 \dots \rho_i)$. The *outcome* of a pair 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 define two restricted notions of strategies. A strategy π_i is:

- *Memoryless* if the output of the strategy function depends only upon the last state in the history.
- *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.

For formal definitions, refer to any work on graph games.

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 the goal of Player 1 in each case is choose to 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 [**?**], i.e., we use existential quantification over i rather than universal. However, all standard results from [**?**] 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 [**?**] 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: (a) $(s_0, s'_0) \in R$; (b) $(s, s') \in R \implies (s \in S_1 \leftrightarrow s' \in S'_1)$; (c) $\forall s, t \in S, s' \in S' : (s, s') \in R \wedge (s, \sigma, t) \in E \wedge s \in S_2 \implies (\exists t' : (s', \sigma, t') \in E' \wedge (t, t') \in R$; and (d) $\forall s \in S, s', t' \in S' : (s, s') \in R \wedge (s', \sigma, t') \in E' \wedge s' \in S'_1 \implies (\exists t : (s, \sigma, t) \in E \wedge (t, t') \in R$.

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: (a) $((s, \#, s'), \sigma, (t, \sigma, s'))$ whenever $(s, \sigma, t) \in E$ and $s \in S_2$; (b) $((s, \#, s'), \sigma, (s, \sigma, t'))$ whenever $(s', \sigma, t') \in E'$ and $s \in S_1$; (c) $((s, \sigma, s'), \sigma, (s, \#, t'))$ whenever $(s', \sigma, t') \in E'$ and $\sigma \in \mathcal{O}$; (d) $((s, \sigma, s'), \sigma, (t, \#, s'))$ whenever $(s, \sigma, t) \in E$ and $\sigma \in \mathcal{I}$; and (e) $(s, \#, s_{err})$ for all Player 2 states s. The objective for Player 1 is $Reach(\{s_{err}\})$.

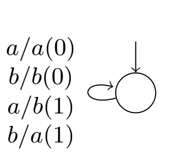
Intuitively, in each step of the simulation game, either Player 1 chooses an input transition of $\mathcal{A}'$ and Player 2 matches it with an input transition of $\mathcal{A}$, or Player 1 chooses an output transition of system $\mathcal{A}$ and Player 2 matches it with an output transition of $\mathcal{A}'$. 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 [**?**], we extended the simulation game by letting the simulation be inexact. In quantitative simulation games, Player 2 can simulate a $\mathcal{A}$ transition by a $\mathcal{A}'$ 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.

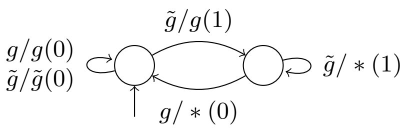
Here, we present a slight generalization of quantitative simulation games for reactive systems with richer error models.

Error Models. The modification schemes used in [**?**] 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 schemes from [**?**]. Hence, we define more general error models.

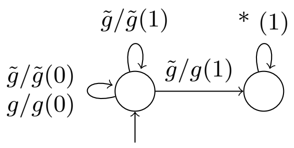
An *error model* over an alphabet $\Sigma = \mathcal{I} \cup \mathcal{O}$ is a deterministic weighted transition system over $\mathcal{O} \times \mathcal{O}$. Intuitively, a transition on label (σ_1, σ_2) (henceforth denoted as σ_1/σ_2) represents that a transition on label σ_1 in the simulated system can be simulated by a transition on label σ_2 in the simulating system with the accompanying cost. Furthermore, we require that each word over symbols of the form σ/σ is assigned cost 0 to ensure that correct simulations have a cost 0.



(a) Standard Model



(b) Delayed Response



(c) Spurious Response

Fig. 3: Sample error models

Given an error model $M = \langle S^e, \mathcal{O} \times \mathcal{O}, E^e, s_0^e \rangle$ 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 $\mathcal{A}_M = \langle S \times S^e, \Sigma, E^M, (s_0, s_0^e), (S_1^M, S_2^M) \rangle$ 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 3a) 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 [**?**].
- *Delayed Response Model.* (Figure 3b) 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.
- *Spurious Response Model.* (Figure 3c) In this model model, we measure the number of spurious grants produced by the implementation and fewer spurious grants the lower the cost of simulation.
- *Qualitative Model.* (Figure 6) This model recovers the boolean simulation games and the distance is 0 if and only if the simulation relation holds.

Quantitative Simulation Games. Given a modified specification system ${\mathcal{A}'}^M$ and an implementation system $\mathcal{A}$, the quantitative simulation game $\mathcal{Q}_M^{\mathcal{A},\mathcal{A}'}$ is a game with the game-graph of $\mathcal{G}^{\mathcal{A},\mathcal{A}'_\mathcal{M}}$, and a weight function that maps (a) Each Player 1 transition to 0, and (b) 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.

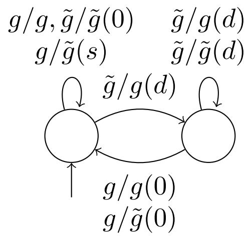


Fig. 4: Combined error model

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

Example 1. Consider the specifications and implementations from the motivating example in Section 1 (Figures 1 and 2). We formalize the informal preference conditions from Section 1 as a combined error model (Figure 4) of delayed- and spurious-response models. Delayed and spurious grants get penalties of d and s, respectively.

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 $\mathcal{S}_2$ (Figures 2 and 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 $\mathcal{I}_2$.

2.2 Properties of Simulation Distances

$d = 2, s = 01$	$\mathcal{S}_1$	$\mathcal{S}_2$
$\mathcal{I}_1$	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 1

In [?], we showed that simulation distances with restricted error models are directed metrics, i.e., reflexivity and the triangle inequality hold. 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 words 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 2.1 *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 [?] with slight modifications. All the error models from Figure 3 are transitive.

Theorem 2.2 *The simulation distance d_M is a directed metric if and only if the error model M is transitive.*

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

Proposition 2.3 *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.*

Proof. Consider the following transition system M^3 with state space $S^e \times S^e \times S^e$ with transitions of the form $((s_1, s_2, s_3), (\frac{\sigma_1}{\sigma_2}, \frac{\sigma_2}{\sigma_3}), (s_1', s_2', s_3'))$ with weight $w_1 + w_2 - w_3$, where $(s_1, \frac{\sigma_1}{\sigma_2}, s_1')$, $(s_2, \frac{\sigma_2}{\sigma_3}, s_2')$ and $(s_3, \frac{\sigma_1}{\sigma_3}, s_3')$ are in E^e having weight w_1, w_2 and w_3, respectively. Transitivity holds iff there is a reachable negative cycle in M^3, which can be checked using graph algorithms 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.

3.1 The Synthesis Problem

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 $\mathbf{v} = \langle v_1, v_2, \dots v_k \rangle \in \mathbb{Q}^k$, 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 $\forall \mathcal{I} : \max_{i \in \{1,2,\dots k\}} d_{M_i}(\mathcal{I}^*, \mathcal{S}_i) \leq \max_{i \in \{1,2,\dots k\}} d_{M_i}(\mathcal{I}, \mathcal{S}_i) + \epsilon$. We call such an implementation $\mathcal{I}^*$ an ϵ-optimal implementation.

We reduce the above problems to 2-player games with *MLimAvg* objectives.

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

Proof. 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 $S_1 = S_1^1 \times \dots \times S_1^k$ where S_1^i are the Player 1 states of $\mathcal{S}_i^{M_i}$;
- Player 2 states $S_2 = S_2^1 \times \dots \times S_2^k$ where S_2^i are the Player 2 states of $\mathcal{S}_i^{M_i}$;
- A transition from state $(s_1, s_2, \dots, s_k)$ to $(s_1', s_2', \dots, s_k')$ 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 $\nu^i((s_1, s_2, \dots, s_k), \sigma, (s_1', s_2', \dots, s_k')) = 2 * \nu((s_i, \sigma, s_i'))$ 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 5.

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 5). 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 $Val(\pi_2) \geq \max_{i \in \{1,2,\ldots k\}} d_{M_i}(\mathcal{I}, \mathcal{S}_i)$, 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 [**?**], 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. □

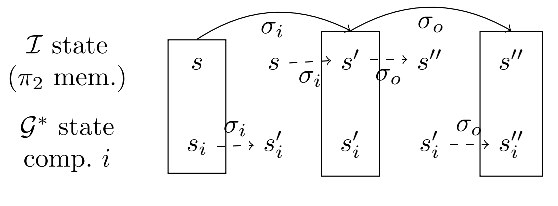


Fig. 5: Working of π_2: Solid edges are transitions in $\mathcal{G}^*$ and dashed edges are transitions in $d_{M_i}(\mathcal{I}, \mathcal{S}_i)$

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 [**?**], 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. □

For the specifications and error models from Example 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.

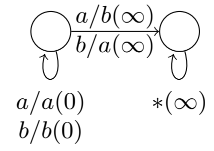


Fig. 6: Qualitative error model

For the qualitative error model (Figure 6) 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 [**?**], we show for the case of two specifications that it always is possible to do better.

Proposition 3.3 *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 a $\mathcal{I}^*$ 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 $d_M(\mathcal{I}^*, \mathcal{S}_1) \leq \delta$ to complete the proof. □

4 Quantitative Alphabets

We generalize the notion of simulation distances to alphabets that are infinite and quantitative, i.e., the alphabet itself is a metric space with the metric m_Σ. These alphabets can be used to model continuously controllable variables.

We consider a modified standard error model from Section 2 (assigning a constant penalty for each mismatch), with the cost of an edge labeled σ/σ' being $m_\Sigma(\sigma, \sigma')$. For a given Σ, the resultant simulation distance (denoted d_∞^Σ) is a directed metric. Σ is omitted when it is clear from the context.

Here, we restrict ourselves to alphabets which have a midpoint function $\mu : \Sigma \times \Sigma \to \Sigma$ such that $\forall \sigma, \sigma' : m_\Sigma(\sigma, \mu(\sigma, \sigma')) = m_\Sigma(\sigma', \mu(\sigma, \sigma')) = \frac{m_\Sigma(\sigma, \sigma')}{2}$.

For quantitative alphabets, the problem of synthesis from incompatible specifications can be solved with a simpler 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 quantitative inputs $\mathcal{I}$ and quantitative outputs $\mathcal{O}$. Let $\delta = \min(d_\infty(\mathcal{S}_1, \mathcal{S}_2), d_\infty(\mathcal{S}_2, \mathcal{S}_1))$. There exists an* *implementation $\mathcal{I}$ such that $d_\infty(\mathcal{I}, \mathcal{S}_1) \leq \frac{\delta}{2}$ and $d_\infty(\mathcal{I}, \mathcal{S}_2) \leq \frac{\delta}{2}$. Furthermore, the number of states of $\mathcal{I}$ is in $O(n_1 n_2)$ where n_i is the number of states of $\mathcal{S}_i$.*

Projection Theorem A classical approach for optimization when the discrete problem is difficult is finding the solution of the corresponding problem embedded in a continuous space and then projecting it to the discrete space to get a good approximation. In that spirit, it might be possible in some cases to synthesize from incompatible specifications for finite alphabets by solving it in the quantitative case and projecting the solution to the discrete alphabet. As a first step towards this goal, we define the projection operation and prove that a projection for automata exists under the natural condition on the alphabets.

Let (X, d) denote a set X equipped with (directed) metric d, and let $Y \subseteq X$. The projection $x|Y$ of $x \in X$ onto Y, is defined as the set of closest points to x in Y. In general, the projection of an element $x \in X$ may be empty. The set X is *projectible* on to Y if $\forall x \in X,\ x|Y \neq \emptyset$. Let $\Sigma = \mathcal{I} \cup \mathcal{O}$ and $\Sigma' = \mathcal{I} \cup \mathcal{O}'$ where $\mathcal{O}' \subset \mathcal{O}$ and let $\mathcal{O}$ be projectible onto $\mathcal{O}'$. We denote by $\mathcal{A}_\Sigma$ the set of open reactive systems over alphabet Σ. The proof of the following theorem relies on the fact that $\mathcal{O}$ is projectible onto $\mathcal{O}'$.

Theorem 4.2 *$(\mathcal{A}_\Sigma, d_\Sigma)$ is projectible onto $\mathcal{A}_{\Sigma'}$. Given a system A in $\mathcal{A}_\Sigma$, there exists a system A' in $\mathcal{A}_{\Sigma'}$ such that A' is in $A|\mathcal{A}_{\Sigma'}$ and the number of states of A' is $\mathcal{O}(n)$, where n is the number of states of A.*

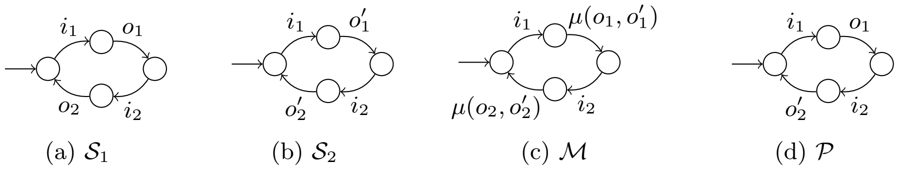


Fig. 7: Example for synthesis using Projection Theorem

Synthesis using projection Theorems 4.1 and 4.2 suggest a way to define a simpler (albeit approximate) construction for synthesis from incompatible specifications for the case of finite alphabets. Given two systems $\mathcal{S}_1$ and $\mathcal{S}_2$ with output alphabet $\mathcal{O}'$, we can consider the systems to be over some quantitative alphabet $\mathcal{O}$ such that $\mathcal{O}' \subset \mathcal{O}$. Using Theorem 4.1, one could synthesize an implementation with output symbols in $\mathcal{O}$, and then use Theorem 4.2 to get an implementation with outputs over the finite alphabet $\mathcal{O}'$. Figure 7 is an example where this method gives the optimal implementation. The machine (M) is the result of synthesis over the quantitative alphabet, thus its simulation distance to both $\mathcal{S}_1$ and $\mathcal{S}_2$ is less than or equal to $\frac{\delta}{2}$, where $\delta = \min(d_\infty(\mathcal{S}_1, \mathcal{S}_2), d_\infty(\mathcal{S}_2, \mathcal{S}_1))$. The machine $\mathcal{P}$ is then obtained as a projection of $\mathcal{M}$. 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 $\mathcal{O}'$).

5 Case study: Optimal Scheduling for Overloads

We present a case study to illustrate the use of simulation distances for conflicting requirements. Consider the task of scheduling on multiple processors, where processes have definite execution times and deadlines. In practice, deadlines are either "soft", where a small delay beyond the deadline is acceptable, but undesirable; or "hard", where any delay is catastrophic. In case of overload, processes are either delayed or dropped completely; and usually these processes are chosen based on priorities. Synthesis from incompatible specifications can be used to schedule based on exact penalties for missing deadlines or dropping processes.

Each process repeatedly requests for execution time and the scheduling is based on time-slices with each processor executing a single process in a time-slice. A process $\mathcal{P}(t, d, c)$ has three parameters: (a) the number of time-slices t needed for completion of the computation; (b) the deadline d from invocation time; and (c) 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 and the output g represents a single time-slice of execution, and the output c indicates that the invocation has been completed. A reactive system representing $\mathcal{P}(2, 3, 1)$ is shown in Figure 8.

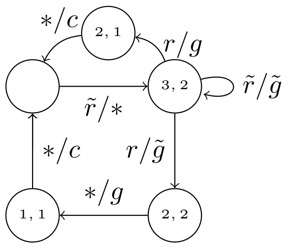


Fig. 8: Modelling processes: $\mathcal{P}(2, 3, 1)$

We define both hard and soft deadline error models. In the hard deadline model, a missed deadline leads to an immediate one-time large penalty p_l, whereas, in the soft deadline model, it leads to a small recurring penalty p_s in each step until the process is finished. Furthermore, we have a one state specification that no more than n processes can be scheduled in each step. For this specification, we use the qualitative failure model. We describe some of the optimal implementations and costs obtained for various combinations:

- For $\mathcal{P}(3, 6, 3)$ and $\mathcal{P}(3, 6, 3)$ and a single processor, we obtain a 0 cost schedule where each process is alternately scheduled till completion.
- 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, P_2 and P_3 on a soft, hard and hard deadlines respectively, 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 set of processes above, but with $p_s = 5$ and $p_l = 10$, we get a scheduler where P_1 is preferred over P_2 and P_3.

6 Conclusion

The key contributions of this paper are: First, we extend the simulation distances framework with stateful error models and inputs and present their basic properties. We show how this framework can be used for quantitative synthesis. Second, we give a method for constructing an optimal solution to the problem of synthesis from incompatible specifications. Third, we demonstrate the usefulness of the approach on a case study of scheduler synthesis.

Future work. There are several possible directions for future research. From the theoretical side, we will attempt to extend the framework of simulation distances to probabilistic systems (to model a probability distribution on inputs). Another possible extension is using bisimulation, as opposed to simulation, as the basis of the distance between systems. From the practical side, we plan to do a larger case study to test whether quantitative metrics enable us to express specifications which are either significantly simpler, more robust, or easier to maintain.

A Appendix to Section 4

Given a function $\mu : \mathcal{O} \times \mathcal{O} \to \mathcal{O}$, we construct the function extension $\tilde{\mu} : \mathcal{A}_\Sigma \times \mathcal{A}_\Sigma \to \mathcal{A}_\Sigma$. Let $\mathcal{A} = \langle S, \Sigma, E, s_0, (S_1, S_2)\rangle$, $\mathcal{A}' = \langle S', \Sigma, E', s'_0, (S'_1, S'_2)\rangle$. Let $\pi_1 : S_1 \times \mathcal{I} \times S'_2 \to S_2$ and $\pi_2 : S_1 \times \mathcal{O} \times S'_2 \to \mathcal{O} \times S'_1$ constitute the optimal positional strategy of player 2 in the $\mathcal{Q}_M^{\mathcal{A},\mathcal{A}'}$ game. π_1 gives the player 2 move in response to an input move by player 1 on $\mathcal{A}'$ and π_2 gives the response of player 2 to an output move on $\mathcal{A}$ by player 1. Let π_2^Σ, π_2^S be the projections of π_2 onto $\mathcal{O}$ and S'_1.

Define $\tilde{\mu}(\mathcal{A}, \mathcal{A}') = \tilde{\mathcal{A}} = \langle \tilde{S}, \Sigma, \tilde{E}, \tilde{s_0}, (\tilde{S_1}, \tilde{S_2})\rangle$ where $\tilde{S_1} = S_1 \times S'_1$, $\tilde{S_2} = S_2 \times S'_2$, $\tilde{s_0} = (s_0, s'_0)$. $\tilde{E}$ is defined as follows:

– $\forall (s, s') \in S_1 \times S'_1, (s', \sigma_\mathcal{I}, t') \in E' \Rightarrow ((s, s'), \sigma_\mathcal{I}, (\pi_1(s, \sigma_\mathcal{I}, t'), t')) \in \tilde{E}$
– $\forall (t, t') \in S_2 \times S'_2, (t, \sigma_\mathcal{O}, s) \in E \Rightarrow$
$((t, t'), \mu(\sigma_\mathcal{O}, \pi_2^\Sigma(s, \sigma_\mathcal{O}, t'), (s, \pi_2^S(s, \sigma_\mathcal{O}, t')))) \in \tilde{E}$

Intuitively, the function extension encodes the optimal moves that player 2 will make against any possible player 1 move, and the transition labels are decided based on the function μ.

Theorem 4.1 *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be specifications over a quantitative inputs $\mathcal{I}$ and quantitative outputs $\mathcal{O}$. Let $\delta = \min(d_\infty(\mathcal{S}_1, \mathcal{S}_2), d_\infty(\mathcal{S}_2, \mathcal{S}_1))$. There exists an implementation $\mathcal{I}$ such that $d_\infty(\mathcal{I}, \mathcal{S}_1) \le \frac{\delta}{2}$ and $d_\infty(\mathcal{I}, \mathcal{S}_2) \le \frac{\delta}{2}$. Furthermore, 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. We show that the construction $\mathcal{I} = \tilde{\mu}(\mathcal{S}_1, \mathcal{S}_2)$ (given $\delta = d_\infty(\mathcal{S}_1, \mathcal{S}_2) \le d_\infty(\mathcal{S}_2, \mathcal{S}_1)$)) satisfies the theorem. Here $\mu : \mathcal{O} \times \mathcal{O} \to \mathcal{O}$ is the midpoint map for $\mathcal{O}$ (this is assumed to exist).

Also suppose that π_0 is the optimal strategy of the minimizing player in the game $\mathcal{Q}_M^{\mathcal{S}_1,\mathcal{S}_2}$ (It has two parts π_1 and π_2 as we mentioned earlier).

We will first show $d_\infty(\mathcal{I}, \mathcal{S}_1) \le \frac{\delta}{2}$ and then $d_\infty(\mathcal{I}, \mathcal{S}_2) \le \frac{\delta}{2}$.

Consider the game $\mathcal{Q}_M^{\mathcal{I},\mathcal{S}_1}$. Suppose player 1 follows strategy σ and player 2 follows π, where the strategy π is to simply ensure that the state on $\mathcal{S}_1$ matches with the first co-ordinate of the state on I. π is sufficient to attain a value $\le \frac{\delta}{2}$. Looking at the sequence of moved followed in the game when player 2 follows π, the sequence of moves is what would have occurred if in the game $\mathcal{Q}_M^{\mathcal{S}_1,\mathcal{S}_2}$, the minimizing player would have followed strategy π_0. By the construction of $\mathcal{I}$, the value of the play $f(\sigma, \pi) \le \frac{\delta}{2}$. So we have

$$d_\infty(\mathcal{I}, \mathcal{S}_1) \le f(\sigma, \pi) \le \frac{\delta}{2}$$

Now consider the $\mathcal{Q}_M^{\mathcal{I},\mathcal{S}_2}$ game. Let σ' be any strategy of player 1. Let π' be the strategy of player 2 which is to ensure that the second coordinate of the current state of $\mathcal{I}$ matches with the current state of $\mathcal{S}_2$. Again

$$d_\infty(\mathcal{I}, \mathcal{S}_2) \le f(\sigma', \pi') \le \frac{\delta}{2}$$

Theorem 4.2 $(\mathcal{A}_\Sigma, d_\infty)$ *is projectible onto* $\mathcal{A}_{\Sigma'}$. *Given a system* A *in* $\mathcal{A}_\Sigma$, *there exists a system* A' *in* $\mathcal{A}_{\Sigma'}$ *such that* A' *is in* $A|\mathcal{A}_{\Sigma'}$ *and the number of states of* A' *is* $\mathcal{O}(n)$, *where* n *is the number of states of* A.

Proof. Let $\mathcal{A} = \langle S, \Sigma, E, s_0, (S_1, S_2)\rangle$. Define $A' \in \mathcal{A}_{\Sigma'}$ as $\mathcal{A}' = \langle S, \Sigma', E', s_0, (S_1, S_2)\rangle$ where E' is defined as :

- $\forall (s, \sigma_\mathcal{I}, t) \in E, s \in S_1, t \in S_2 \Rightarrow (s, \sigma_\mathcal{I}, t) \in E'$
- $\forall (t, \sigma_\mathcal{O}, s) \in E, s \in S_1, t \in S_2 \Rightarrow (t, \sigma_{\mathcal{O}'}, s) \in E'$ where $\sigma_{\mathcal{O}'} \in \sigma_\mathcal{O}|\Sigma'$.

We show that $\forall B \in \mathcal{A}_{\Sigma'},\ d_\infty(A, A') \leq d_\infty(A, B)$ and $d_\infty(A', A) \leq d_\infty(B, A)$ Let $B \in \mathcal{A}_{\Sigma'}$. Let $d_\infty(A, A') = f(\sigma^*, \pi^*)$ where σ^* and π^* are optimal positional strategies of player 1 and 2 respectively. σ^* and π^* together form a cycle in the game graph which corresponds to a walk W in A. Let $a(e)$ represent the label of edge e.

$$d_\infty(A, A') = \frac{1}{|W|} \sum_{e \in W} m_\Sigma(a(e), a(e)|\Sigma')$$

In the $\mathcal{Q}_M^{A,B}$ game, if player 1 follows the strategy σ_W, of following the walk W over and over, and if the player 2 follows his optimal strategy ν^* then

$$d_\infty(A, A') = \frac{1}{|W|} \sum_{e \in W} m_\Sigma(a(e), a(e)|\Sigma') \leq f(\sigma_W, \nu^*) \leq d_\infty(A, B)$$

This proves one part. The proof for $d_\infty(A', A) \leq d_\infty(B, A)$ is similar.