Interface Simulation Distances[1]

Pavol Černý[a], Martin Chmelík[b], Thomas A. Henzinger[b], Arjun Radhakrishna[b]

[a] *University of Colorado Boulder*
[b] *Institute of Science and Technology Austria*

Abstract

The classical (boolean) notion of refinement for behavioral interfaces of system components is the alternating refinement preorder. In this paper, we define a distance for interfaces, called *interface simulation distance*. It makes the alternating refinement preorder quantitative by, intuitively, tolerating errors (while counting them) in the alternating simulation game. We show that the interface simulation distance satisfies the triangle inequality, that the distance between two interfaces does not increase under parallel composition with a third interface, that the distance between two interfaces can be bounded from above and below by distances between abstractions of the two interfaces, and how to synthesize an interface from incompatible requirements. We illustrate the framework, and the properties of the distances under composition of interfaces, with two case studies.

Keywords: Alternating simulation, Quantitative analysis, Games.

1. Introduction

The component-based approach is an important design principle in software and systems engineering. In order to document, specify, validate, or verify components, various formalisms that capture behavioral aspects of component interfaces have been proposed [10, 16, 17, 22]. These formalisms capture assumptions on the inputs and their order, and guarantees on the outputs and their order. For closed systems (which do not interact with the environment via inputs or outputs), a natural notion of refinement is given by the simulation preorder. For open systems, which expect inputs and provide outputs, the corresponding notion is given by the alternating simulation preorder [1]. Under alternating simulation, an interface A is refined by an interface B if, after any given sequence of inputs and outputs, B accepts all inputs that A accepts, and B provides only outputs that A provides. The alternating simulation preorder

[1] This research was partially supported by the European Research Council (ERC) Advanced Investigator Grant QUAREM, the Austrian Science Fund (FWF) projects S11402-N23 and S11407-N23 (RiSE), the Austrian Science Fund (FWF) Grant No P 23499-N23, ERC Start grant (279307: Graph Games) and Microsoft faculty fellows award.

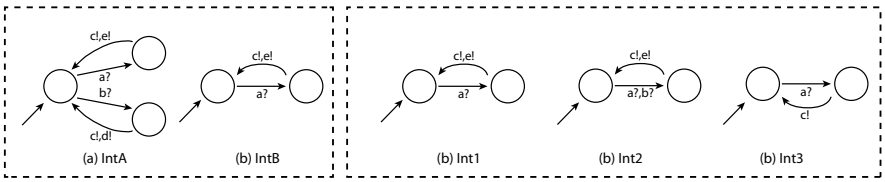

Figure 1: Interface examples

is a boolean notion. Interface A either is refined by interface B, or it is not. However, there are various reasons for which the alternating simulation can fail, and one can make quantitative distinctions between these reasons. For instance, if B does not accept an input that A accepts (or provides an output that A does not provide) at every step, then B is more different from A than an interface that makes a mistake once, or at least not as often as B.

We propose an extension of the alternating simulation to the quantitative setting. We build on the notion of simulation distances introduced in [5]. Consider the definition of alternating simulation of an interface A by an interface B as a two-player game. In this game, Player 1 chooses moves (transitions), and Player 2 tries to match them. Player 1 chooses input transitions from the interface A and output transitions from interface B, Player 2 responds by a transition from the other system. The goal of Player 1 is to prove that the alternating simulation does not hold, by driving the game into a state from which Player 2 cannot match the chosen move; the goal of Player 2 is to prove that there exists an alternating simulation, by playing the game forever. We extend this definition to the quantitative case. Informally, we will tolerate errors by Player 2. However, Player 2 will pay a certain price for such errors. More precisely, Player 2 is allowed to "cheat" by following a non-existing transition. The price for such transition is given by an *error model*. The error model assigns the transitions from the original system a weight 0, and assigns the new "cheating" transitions a positive weight. The goal of Player 1 is then to maximize the cost of the game, and the goal of Player 2 is to minimize it. The cost is given by an objective function, such as the limit average of transition prices. As Player 2 is trying to minimize the value of the game, she is motivated not to cheat. The value of the game measures how often Player 2 can be forced to cheat by Player 1.

Example 1. Consider the example in Figure 1. The two interfaces on the left side (IntA and IntB) represent requirements on a particular component by a designer. The three interfaces (Int1, Int2, and Int3) on the right side are interfaces for different off-the-shelf components provided by a vendor. We illustrate how interface simulation distances can be used by the designer to choose a component whose interface satisfies her requirements most closely. Interface Int1 is precisely the interface required by IntB, so the distance from IntB to Int1 will be 0. However, the distance from IntA to Int1 is much greater. Informally, this

is because Player 1, choosing a transition of IntA could choose the $b?$ input. Player 2, responding by a transition of Int1 has to cheat by playing the $a?$ input. After that, Player 1 could choose the $e!$ output (as a transition of Int1), and Player 2 (this time choosing a transition from IntA) has to cheat again. Player 2 thus has to cheat at every step. Interface Int2 (resp. Int3) improves on Int1 with respect to requirement IntA by adding inputs (resp. removing outputs). The distance from IntA to Int2 (or Int3) is exactly half of the distance from IntA to Int1. The interfaces Int2 and Int3 have distance 0 to IntB. Int2 and Int3 satisfy the requirements IntA and IntB better than the interface Int1.

The model of behavioral interfaces we consider is a variant of interface automata [10]. This choice was made for ease of presentation of the main ideas of the paper. However, the definition of interface simulation distance can be extended to richer models.

We establish basic properties of the interface simulation distance. First, we show that the triangle inequality holds for the interface simulation distance. This, together with the fact that reflexivity holds for this distance as well, shows that it is a *directed metric* [13]. Second, we give an algorithm for calculating the distance. The interface simulation distance can be calculated by solving the value problem in the corresponding game, that is, in limit-average games or discounted-sum games. The values of such games can be computed in pseudo-polynomial time [23]. (More precisely, the complexity depends on the magnitude of the largest weight used in the error model. Thus the running time is exponential in the size of the input, if the weights are given in binary.)

We present composition and abstraction techniques that are useful for computing and approximating simulation distances between large systems. These properties suggest that the interface simulation distance provides an appropriate basis for a quantitative analysis of interfaces. The composition of interface automata, which also composes the assumptions on their environments, was defined in [10]. In this paper, we prove that the distance between two interfaces does not increase under the composition with a third interface. The technical challenges in the proof appear precisely because of the involved definition of composition of interface automata, and are not present in the simpler setting closed systems of [5]. We also show that the distance between two interfaces can be over- or under- approximated by distances between abstractions of the two interfaces. For instance, for over-approximation, input transitions are abstracted universally, and output transitions are abstracted existentially. Finally, we show how to synthesize an interface given a set of incompatible requirements. This situation often occurs when requirements are provided by different sources. We provide an algorithm that given the requirements constructs an ϵ-optimal implementation interface.

We illustrate the interface simulation distance, and in particular its behavior under interface composition, on two case studies. The first concerns a simple message transmission protocol over an unreliable medium. The second case study models error correcting codes.

Related work. The alternating simulation preorder was defined in [1] in order

to generalize the simulation preorder to input/output systems. The alternating simulation can be checked in polynomial time, as is the case for the ordinary simulation relation. Interface automata have been defined in [10] to facilitate component-based design, and various techniques were proposed to handle real-time, resource-handling models, and modal systems [15, 12, 7, 19]. Interface automata and related techniques have enabled verification of "real-world" systems (see for example [18, 20]). The natural notion of refinement for interface automata corresponds to the alternating simulation preorder. Simulation distances have been proposed in [5] (the full version was published recently in [6]) as a step towards extending specification formalisms and verification algorithms to a quantitative setting. This paper extends the quantitative notion of simulation distances to the alternating simulation preorder for interface automata.

There have been several attempts to give mathematical semantics to reactive processes based on quantitative metrics rather than boolean preorders [21, 9]. In particular for probabilistic processes, it is natural to generalize bisimulation relations to bisimulation metrics [14], and similar generalizations can be pursued if quantities enter through continuous variables, such as time [2]. In contrast, we consider distances between purely discrete (non-probabilistic, untimed) systems.
Summary. This paper defines the interface simulation distance for automata with inputs and outputs, establishes basic properties of this distance, as well as abstraction and compositionality theorems. In Section 2 we introduce all the necessary terms. In Section 3 we present the properties of the measure, and finally in Section 4 we present two case studies. This work extends the work presented in [3], in particular we extend the results by presenting new proof for Theorem 4 in Section 3 and in Subsection 3.4 we present new work on how to synthesize an interface from incompatible requirements.

2. Interface Simulation Distances

2.1. Broadcast Interface Automata

Interface automata were introduced in [10] to model components of a system communicating through interfaces. We use a variant of interface automata which we call *broadcast interface automata* (BIA).

A *broadcast interface automaton* F is a tuple $(Q, q^0, A^I, A^O, \delta)$ consisting of a finite set of states Q, the initial state q^0, two disjoint sets A^I and A^O of input and output actions and a set $\delta \subseteq Q \times A \times Q$ of transitions. We let $A = A^I \cup A^O$. Additionally, we require that F is input deterministic, i.e., for all $q, q', q'' \in Q$ and all $\sigma_I \in A^I$ if there are transitions (q, σ_I, q') and $(q, \sigma_I, q'') \in \delta$, then $q' = q''$.

Given a state $q \in Q$ and an action $\sigma \in A$ let $\mathsf{post}(q, \sigma) = \{q' \mid (q, \sigma, q') \in \delta\}$. Similarly given a state $q \in Q$ let $A^I(q)$ be the input actions enabled at state q ($A^O(q)$ for output actions). Note that the *BIA* is not required to be input-enabled, hence there may be states q where $A^I(q) \neq A^I$.

An example of a *BIA* can be seen on Figure 1. The actions terminated by the ?(resp. !) symbol are input (resp. output) actions. The *BIA* IntA can input

$a?$ or $b?$. Depending on the input it can output $c!$ or $e!$ if the input is $a?$ ($c!$ or $d!$ if the input is $b?$), and this repeats forever.

There are two differences between standard interface automata and BIAs. First, the communication paradigm in interface automata is pairwise, i.e., an output from a component can serve as the input to only one other component. However, in BIAs the communication model is *broadcast*, i.e., an output from a component can serve as input for multiple different components. Second, standard interface automata have hidden (internal) actions, which are omitted from the definition of BIAs. These modifications were introduced in order to simplify the presentation of the interface simulation distance, and to enable us to clearly express the principal ideas. The distance can be defined for richer models of automata with inputs and outputs, including for standard interface automata.

Alternating Simulation. Given two BIAs F and F', a binary relation on states $\succeq \subseteq Q_F \times Q_{F'}$ is an alternating simulation by F of F' if $q \succeq q'$ implies:

1. for all $\sigma_I \in A^I(q)$ and $r \in \mathsf{post}(q, \sigma_I)$, there exists a state $r' \in \mathsf{post}(q', \sigma_I)$ such that $r \succeq r'$;
2. for all $\sigma_O \in A^O(q')$ and $r' \in \mathsf{post}(q', \sigma_O)$, there exists a state $r \in \mathsf{post}(q, \sigma_O)$ such that $r \succeq r'$.

A BIA F' *refines* a BIA F (written $F \succeq F'$) if

1. $A^I_F \subseteq A^I_{F'}$ and $A^O_F \supseteq A^O_{F'}$;
2. there exists an alternating simulation $\succeq$ by F of F' such that $q^0_F \succeq q^0_{F'}$.

The intuition behind the above definitions is that when $F \succeq F'$, the component F in a system can be replaced with component F' without leading to any erroneous behavior.

Consider the BIAs IntB and $\mathsf{Int2}$ in Figure 1. Note that $\mathsf{Int2}$ refines IntB, i.e., $\mathsf{IntB} \succeq \mathsf{Int2}$. One can easily observe that the converse is not true.

Composition of BIAs. When composing BIAs it is required for the inputs (outputs) of the two automata not to mix, i.e., two BIAs F and G are *composable* if $A^I_F \cap A^I_G = \emptyset$ and $A^O_F \cap A^O_G = \emptyset$. For two composable *BIAs* F and G we let $\mathit{shared}(F, G) = A_F \cap A_G$.

Whenever there is an action $\sigma \in \mathit{shared}(F, G)$ the composed system makes a joint transition and the output action remains visible. Finally, the *composition* of two composable *BIAs* $F = (Q_F, q^0_F, A^I_F, A^O_F, \delta_F)$ and $G = (Q_G, q^0_G, A^I_G, A^O_G, \delta_G)$ is a BIA $F \times G = (Q_{F\times G}, q^0_{F\times G}, A^I_{F\times G}, A^O_{F\times G}, \delta_{F\times G})$ where the states of the product $Q_{F\times G}$ are $Q_F \times Q_G$, with the initial state $q^0_{F\times G} = (q^0_F, q^0_G)$. The input(resp. output) alphabet of is $A^I_{F\times G} = A^I_F \cup A^I_G \setminus \mathit{shared}(F, G)$ (resp. $A^O_{F\times G} = A^O_F \cup A^O_G$). The transition relation $\delta_{F\times G}$ contains the transition $((q, r), \sigma, (q', r'))$ iff

- $\sigma \notin \mathit{shared}(F, G)$ and $(q, \sigma, q') \in \delta_F$ and $r = r'$, or
- $\sigma \notin \mathit{shared}(F, G)$ and $(r, \sigma, r') \in \delta_G$ and $q = q'$, or
- $\sigma \in \mathit{shared}(F, G)$ and $(q, \sigma, q') \in \delta_F$ and $(r, \sigma, r') \in \delta_G$.

Given two composable *BIAs* F and G, a product state (p, q) is an *error* state of the product automaton $F \times G$ if there exists a shared action $\sigma \in \mathit{shared}(F, G)$ such that $\sigma \in A^O_F(p)$ and $\sigma \notin A^I_G(q)$ or $\sigma \in A^O_G(q)$ and $\sigma \notin A^I_F(p)$. A state

(p, q) of the product automaton is *compatible* if no error state is reachable from the state (p, q) using only output actions.

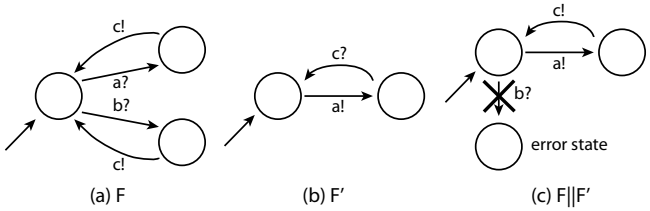

Figure 2: Composition of *BIAs*

A state that is not compatible is *incompatible*. Two *BIAs* F and G are *compatible* iff the initial state of their product automaton $F \times G$ is compatible (denoted by $F \sim G$). The product of two compatible automata F and G restricted to compatible states is denoted by $F \parallel G$ and is obtained from $F \times G$ by removing input action transitions that lead from compatible to incompatible states.

A composition of *BIAs* F and F' and the composed interface $F \parallel F'$ restricted to compatible states can be seen on Figure 2. Actions a and c become shared actions in the composition and the composed interface makes a joint transition on these actions. Note that when constructing the product $F \times F'$ an error state is reachable, therefore the input transition ($b?$) leading from a compatible to an incompatible state is removed from the product $F \parallel F'$.

2.2. Graph Games

In this section, we introduce concepts from the theory of 2-player graph games that are necessary for the exposition. A *game graph* is a tuple $H = (S, S_1, S_2, E, s_\iota)$, where S is a finite set of states, $E \subseteq S \times S$ is a set of edges, $s_\iota \in S$ is an initial state, and S_1 and S_2 partitions the state space S into Player 1 and Player 2 states, respectively. The game proceeds as follows: First, a token is placed on the initial state s_ι. Now, whenever the token is on a state $s \in S_i, i \in \{1, 2\}$ Player i picks a successor s' of s and the token is moved to the state s', and the process continues infinitely. The result $\rho = \rho_0\rho_1 \ldots$ of an infinite sequence of visited states is called a *play*. The set of all plays is denoted by Ω.

Strategies. A *strategy for Player i* is a recipe for Player i to choose the next transition. Formally, a Player i strategy $\pi^i : S^* \cdot S_i \to S$ is a function such that for all $w \in S^*$ and $s \in S_i$, we have $(s, \pi^1(w \cdot s)) \in E$. We denote by Π^i, the set of all Player i strategies. The string w is called the *history* of the play and s is called the *last state* of the play.

We define two restricted notions of strategies that are sufficient in many cases. A strategy is:

- *Positional* or *memoryless* if the chosen successor is independent of the history, i.e., for all $w \in S^*$, $\pi^i(w \cdot s) = \pi^i(s)$.
- *Finite-memory* if there exists a finite memory set M and an initial memory state $m_0 \in M$, a memory function $\mu : S^* \times M \to M$, and a move function $\nu : M \times S_i \to S$ such that: (a) $\mu(\epsilon, m_0) = m_0$ and $\mu(w \cdot s, m_0) = \mu(s, \mu(w, m_0))$; and (b) $\pi^i(w \cdot s) = \nu(\mu(w, m_0), s)$. Intuitively, (a) the state of the memory is updated based only upon the previous state of the memory and the last state of the play; and (b) the chosen successor depends only on the state of the memory and the last state of the play.

A play $\rho = \rho_0\rho_1 \ldots$ is *conformant* to a Player i strategy π^i if for every $\rho_j \in S_i$, we have $\pi^i(\rho_0 \ldots \rho_j) = \rho_{j+1}$. Given a game graph H and strategies π^1 and π^2 for Player 1 and Player 2, respectively, we get a unique path $Out_H(\pi^1, \pi^2)$ that is conformant to both of the strategies.

Objectives. A *boolean objective* $\Phi \subseteq \Omega$ denotes that Player 1 wins if the resultant play ρ is in Φ, and that Player 2 wins otherwise. A Player i *winning strategy* is one for which all plays conformant to it are winning for Player i. We deal with only the *reachability* boolean objective. Given a set of target states $T \subseteq S$ and a play $\rho = \rho_0\rho_1 \ldots$, $\rho \in Reach_T$ if and only if $\exists i : \rho_i \in T$.

A *quantitative objective* is a real–valued function $f : \Omega \to \mathbb{R}$ and the goal of Player 1 is to maximize the value of the play, whereas the goal of Player 2 is to minimize it. We consider the following quantitative objectives: Given a weight function $\omega : E \to \mathbb{R}$, we have

- $\text{LimAvg}(\rho) = \liminf_{n\to\infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} \omega((\rho_i, \rho_{i+1}))$
- $Disc_\lambda(\rho) = \lim_{n\to\infty} \sum_{i=0}^{n-1} \lambda^i \cdot \omega((\rho_i, \rho_{i+1}))$

Given a quantitative objective f and a Player 1 strategy π^1, the *value of strategy* π^1, denoted by $\nu_1(\pi^1 \in \Pi^1)$ is $\inf_{\pi^2\in\Pi^2} f(Out_H(\pi^1, \pi^2))$. Similarly, value $\nu_2(\pi^2)$ of a Player 2 strategy π^2 is $\sup_{\pi^1\in\Pi^1} f(Out_H(\pi^1, \pi^2))$. The *value of the game* is defined as $\sup_{\pi^1\in\Pi^1} \nu_1(\pi^1)$ or equivalently, $\sup_{\pi^1\in\Pi^1} \inf_{\pi^2\in\Pi^2} \nu(Out_H(\pi^1, \pi^2))$. A strategy is *optimal* if its value is equal to the value of the game. We conclude this section by stating the memoryless-determinacy theorems for *LimAvg* and *Disc* objectives (see e.g.[23]).

Theorem 1. *For any game graph H and a weight function ω, we have that* $\sup_{\pi^1\in\Pi^1} \inf_{\pi^2\in\Pi^2} f(Out_H(\pi^1, \pi^2)) = \inf_{\pi^2\in\Pi^2} \sup_{\pi^1\in\Pi^1} f(Out_H(\pi^1, \pi^2))$ *for $f \in \{LimAvg, Disc\}$. Furthermore, there exist memoryless optimal strategies for both players.*

2.3. Interface Simulation Games

Simulation like relations can be characterized as the existence of winning strategies in 2-player games known as simulation games. Here, we present the analogue of simulation games for alternating simulation of BIAs.

Alternating Simulation Games. Intuitively, given BIAs F and F', Player 1 picks either an input transition from F or an output transition from F', and

Player 2 has to match with a corresponding transition with the same action from F' or F. We have $F \succeq F'$ if and only if Player 2 can keep matching the transitions forever.

Given *BIAs* $F = (Q_F, q_F^0, A_F^I, A_F^O, \delta_F)$ and $F' = (Q_{F'}, q_{F'}^0, A_{F'}^I, A_{F'}^O, \delta_{F'})$, such that $A_F^I \subseteq A_{F'}^I$ and $A_F^O \supseteq A_{F'}^O$, the *alternating simulation game* $H_{F,F'} = (S, S_1, S_2, E, s^0)$ is defined as follows:

- The state-space $S = S_1 \cup S_2$, where $S_1 = \{(s, \#, s') \mid s \in Q_F, s' \in Q_{F'}\} \cup \{s_{\mathsf{err}}\}$ and $S_2 = \{(s, \sigma, s') \mid s \in Q_F, s' \in Q_{F'}, \sigma \in \Sigma\}$.
- The initial state is $s^0 = (q_F^0, \#, q_{F'}^0)$;
- The Player 1 edges correspond to:
  - Either input transitions from F: $(s, \#, s') \to (t, \sigma_I, s') \in E \Leftrightarrow s \xrightarrow{\sigma_I} t \in \delta_F$; or
  - Output transitions from F': $(s, \#, s') \to (s, \sigma_O, t') \in E \Leftrightarrow s' \xrightarrow{\sigma_O} t' \in \delta_{F'}$.
- The Player 2 edges correspond to
  - Either input transitions from F': $(t, \sigma_I, s') \to (t, \#, t') \in E \Leftrightarrow s' \xrightarrow{\sigma_I} t' \in \delta_{F'}$; or
  - Output transitions from F: $(s, \sigma_O, t') \to (t, \#, t') \in E \Leftrightarrow s \xrightarrow{\sigma_O} t \in \delta_F$.
- For all states $s \in S_2$ if there is no outgoing edge from s we make an edge $s \to s_{\mathsf{err}}$; and for all states $s \in S_1$ if there is no outgoing edge from s we make a selfloop on s.

The objective of Player 1 is to reach the state s_{err} and the objective of Player 2 is to avoid reaching s_{err}.

We have the following theorem.

Theorem 2. *Given BIAs F and F' and the corresponding alternating simulation game $H_{F,F'}$, we have that $F \succeq F'$ if and only if Player 2 has a winning strategy in $H_{F,F'}$.*

2.4. Quantitative Interface Simulation Games

We aim to establish a distance function between broadcast interface automata that expresses how "compatible" the automata are, even when the standard boolean notion of refinement is not true. In order to do that we give more power to Player 2, by allowing him to play actions that are not originally in the game. However, to avoid free use of such actions every time Player 2 plays the added action he receives a penalty. As we do not want Player 2 to play completely arbitrarily we formalize the allowed "cheating" by a notion of input (output) error models. We assume that the error model is provided to us by the user. The error models express the kind of errors or violations of specifications ("cheating") that is considered acceptable in the current scenario, and assign a cost or a penalty to each kind of violation..

An *input* (resp. *output*) *error model* is a function $M : A^I \times A^I \to \mathbb{N} \cup \{\bot\}$ (resp. $A^O \times A^O \to \mathbb{N} \cup \{\bot\}$). We require for all $a, b, c \in A^I (A^O)$ that $M(a, a) = 0$ and $M(a, b) + M(b, c) \geq M(a, c)$. Given a *BIA* $F = (Q, q^0, A^I, A^O, \delta)$ and an error model M, let the *modified system* be $F \otimes M = (Q, q^0, A^I, A^O, \delta^e)$ with a weight function $\omega_M : \delta^e \to N$, where the terms are defined as follows:

- $(s, \sigma_2, t) \in \delta^e \Leftrightarrow ((s, \sigma_1, t) \in \delta \wedge M(\sigma_1, \sigma_2) \neq \bot)$;
- $\omega_M((s, \sigma_2, t)) = \min_{(s,\sigma_1,t)\in\delta}\{M(\sigma_1, \sigma_2)\}$;

Note, that the automata enhanced with input error models are not *BIAs* as they may not be input deterministic. However, all the definitions for *BIAs* can be naturally interpreted on a *BIA* composed with an error model. Given two *BIAs* F with an output error model M_O and F' with an input error model M_I we construct a game $H_{F\otimes M_O, F'\otimes M_I}$ for systems $F \otimes M_O$ and $F' \otimes M_I$ similarly as is described for *BIAs* in the previous subsection. We measure the "cheating" performed by Player 2 as either the limit-average or the discounted sum of the weights on the transition. The transitions going out from Player 1 states get weight 0 with an exception of the selfloop on s_{err} state that gets the maximal weight assigned by the error model. The weight of an edge from a Player 2 state is either (a) twice the weight of the corresponding $F' \otimes M_I$ transition when matching inputs; or (b) twice the weight of the corresponding $F \otimes M_O$ transition when matching outputs. The factor 2 occurs due to normalization. Given two BIAs F and F', a quantitative objective $f \in \{\text{LimAvg}, Disc_\lambda\}$, an input error model M_I, and an output error model M_O, the *interface simulation distance* $d^f(F \otimes M_O, F' \otimes M_I)$ is defined to be the value of game $H_{F\otimes M_O, F'\otimes M_I}$. Consider again the example in Figure 1, when using error models that can play input (output) actions interchangeably by receiving penalty 1. The distances d among the systems for the quantitative objective LimAvg are presented in Table 1.

The result $d(\mathsf{IntA}, \mathsf{Int1}) = 1$ varies from the case of simulation distance where the distance would be 0. Intuitively, the reason for the maximal distance is that the specification requires the system to be able to handle input $b?$. This input is missing in implementation Int1 and therefore cannot be handled. Moreover, on the only input it handles it can produce an output $e!$ that is missing in the specification.

	Int1	Int2	Int3
IntA	1	1/2	1/2
IntB	0	0	0

Table 1: The distances in Ex. 1

This fact is reflected in the computation of the interface simulation distance as follows: According to the specification, the user expects to receive output $c!, d!$ on input $b?$. The implementation does satisfy any of these requirements and fails to match the user expectation in every step, i.e., the specification IntA prescribes behavior on input $b?$ that is completely missing in the implementation Int1. If Player 1 chooses to play input $b?$ in IntA, then Player 2 has no choice but to respond by $b?$ and receiving the first penalty. Again Player 1 can play the $e!$ output action forcing the second Player 2 to cheat again. By repeating these transitions Player 1 can force Player 2 to receive a penalty in every turn and therefore the distance is 1. The distance can be improved by adding a $b?$ input action as is shown in the case of Int2, where the distance has decreased to 1/2. Player 2 can now match every possible input, but fails to react on the $e!$ output action. Player 1 can ensure the value 1/2 by playing $b?$ and $e!$ repeatedly. The second option to improve the distance is to remove some of the output edges as is shown in Int3. Player 2 still cannot match the input $b?$ but can respond to $c!$ without receiving any penalty. As in the previous case playing a sequence $b?, c!$

ensures value 1/2 for Player 1.

2.5. Complexity

By the results presented in [23] the complexity of finding the value of the game for *LimAvg* objective is in $\mathcal{O}(|V|^3 \cdot |E| \cdot W)$, where $|V|$ is the number of game states, $|E|$ is the number of edges and W is the maximal non–infinite weight used in the game. In our case for *BIA* $F = (Q_F, q_F^0, A_F^I, A_F^O, \delta_F)$ and $G = (Q_G, q_G^0, A_G^I, A_G^O, \delta_G)$ and error model M_O, M_I, the number of states in the game $H_{F \otimes M_O, G \otimes M_I}$ is $|Q_F| \cdot |Q_G| \cdot (|A_F| + |A_G| + 1) + 1$ and the number of edges is bounded by $|V|^2$. The algorithm for $Disc_\lambda$ given a fixed λ is in PTIME by a variation of an algorithm presented in [23].

3. Properties of Interface Simulation Distances

In this section, we present properties of the interface simulation distance. The distance satisfies the triangle inequality and does not increase when *BIAs* are composed with a third interface. Moreover, the distance can be bounded from above and below by considering the abstractions of the systems. We start with the following statement, that relates refinement and interface simulation distances.

Theorem 3. *For $f \in \{LimAvg, Disc_\lambda\}$ and for all error models M_I, M_O and BIAs F, F' we have if $F \succeq F'$ then $d^f(F \otimes M_O, F' \otimes M_I) = 0$.*

Proof. From the assumption $F \succeq F'$ follows that Player 2 does not need to cheat in the quantitative interface simulation game. As all the error models satisfy the property that $M(a, a) = 0$, it follows that the distance is 0. □

Note, that in general the assumption that the distance is 0 does not guarantee that the boolean refinement is true. Consider an example where Player 2 can be forced to cheat only in the first turn. In that case the distance would be 0 for the *LimAvg* objective, however the boolean notion of refinement is not satisfied.

3.1. Triangle Inequality

The triangle inequality is the quantitative analogue of the boolean transitivity property. We show that the interface simulation distance is a directed metric, i.e., it satisfies the triangle inequality and the reflexivity property.

Theorem 4. *For $f \in \{LimAvg, Disc_\lambda\}$ the interface simulation distance d^f is a directed metric, i.e.:*

1. *For all error models M_I, M_O and BIAs F_1, F_2, F_3 we have:*

$$d^f(F_1 \otimes M_O, F_3 \otimes M_I) \leq d^f(F_1 \otimes M_O, F_2 \otimes M_I) + d^f(F_2 \otimes M_O, F_3 \otimes M_I)$$

2. *For all error models M_I, M_O and BIAs F we have*

$$d^f(F \otimes M_O, F \otimes M_I) = 0$$

Proof. We only prove the triangle inequality and omit the proof of the reflexivity property as it is simple. (a) The basic idea of the proof of the triangle inequality is as follows: from the best strategy for Player 2 in the games for computing $d(F_1 \otimes M_O, F_2 \otimes M_I)$ and $d(F_2 \otimes M_O, F_3 \otimes M_I)$, we construct a witness Player 2 strategy in the game for computing $d(F_1 \otimes M_O, F_3 \otimes M_I)$.

For brevity, we denote by H_{ij} the game $H_{F_i \otimes M_O, F_j \otimes M_I}$. Let π^2_{12} and π^2_{23} be the optimal Player 2 strategies in games H_{12} and H_{23}, respectively. Due to the memoryless determinacy of mean-payoff games, we can assume that π^2_{12} and π^2_{23} are positional. We now construct a finite-memory strategy π^2_{13} with the memory being Q_2, i.e., the state space of F_2.

Intuitively, π^2_{13} works as follows: Let $(s_1, \#, s_3)$ be the state of the game H_{13} and let the memory of π^2_{13} be s_2. We can consider the auxiliary H_{12} and H_{23} states to be $(s_1, \#, s_2)$ and $(s_2, \#, s_3)$.

We handle the case of the Player 1 choosing an input transition. The other case is symmetric. Say Player 1 chooses the input transition (s_1, σ_I, s'_1) in $F_1 \otimes M_O$ and the H_{13} state changes to (s'_1, σ_I, s_3).

- Let (s_2, σ_I, s'_2) be the $F_2 \otimes M_I$ transition chosen by π^2_{12} in the state $s'_1, \sigma_I, s_2)$.
- By definition, there exists an action σ'_I such that the transition (s_2, σ'_I, s'_2) is in F_2 such that the cost of the choice is $M_I(\sigma_I, \sigma'_I)$.
- Let (s_3, σ'_I, s'_3) be the $F_3 \otimes M_I$ transition chosen by π^2_{23} in the state (s'_2, σ'_I, s_3), i.e., the response to the Player 1 transition (s_2, σ'_I, s'_2).
- Again by definition, there exists a transition (s_3, σ''_I, s'_3) such that the cost of this choice is $M_I(\sigma'_I, \sigma''_I)$

Now, the responce of π^2_{13} in H_{13} is (s_3, σ_I, s''_3). The cost of this choice is at most $M_I(\sigma_I, \sigma''_I)$, which is at most $M_I(\sigma_I, \sigma'_I) + M_I(\sigma'_I, \sigma''_I)$.

Fix the optimal positional Player 1 strategy in H_{13}. Now, for play ρ_{13} in H_{13} conforming to π^2_{13}, for each step, we can extract the following parts:

- Transition chosen by Player 1 in $F_1 \otimes M_O$ (or $F_3 \otimes M_I$);
- Transition chosen by π^2_{12} in H_{12} from $F_2 \otimes M_I$ (or by π^2_{23} in H_{23} from $F_2 \otimes M_O$);
- The corresponding $F_2 \otimes M_O$ transition or (or $F_2 \otimes M_I$ transition); and
- Transition chosen by π^2_{13} in H_{13} from $F_3 \otimes M_I$ (or $F_1 \otimes M_O$).

From these transitions we can reconstruct a play ρ_{12} in game H_{12} and ρ_{23} in H_{23}. Furthermore, if η^n_{ij} is the cost of the n^{th} step in ρ_{ij}, we have

$$\eta^n_{13} \leq \eta^n_{12} + \eta^n_{23}$$

Therefore, we have

$$\nu(\rho_{13}) = \liminf_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \eta^i_{13} \leq \liminf_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \eta^i_{12} + \eta^i_{23}$$

As the strategies we are considering are only of finite memory, the sequence of penalties is ultimately repeating. This allows us to use lim instead of lim inf and vice versa.

$$\begin{aligned}
d(F_1 \otimes M_O, F_3 \otimes M_I) &= \liminf_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \eta_{13}^i \\
&\leq \liminf_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \eta_{12}^i + \eta_{23}^i = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \eta_{12}^i + \eta_{23}^i \\
&= \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \eta_{12}^i + \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \eta_{23}^i \\
&= \liminf_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \eta_{12}^i + \liminf_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \eta_{23}^i \\
&= \nu(\rho_{12}) + \nu(\rho_{23}) \\
&\leq d(F_1 \otimes M_O, F_2 \otimes M_I) + d(F_2 \otimes M_O, F_3 \otimes M_I)
\end{aligned}$$

We showed that the exists finite memory Player 2 strategy that ensures the needed value of the game. This concludes the proof. □

3.2. Composition

In this part we show that the distance between two *BIAs* F and F' does not increase when both are composed with a third *BIA* G, when using the same error models M_O, M_I.

As we want to use the same output error model M_O in F and $F \parallel G$ (similarly M_I in F' and $F' \parallel G$), we restrict the error models. Assume $\sigma_1 \neq \sigma_2$, then:

- If $M_O(\sigma_1, \sigma_2) \neq \perp$, then $\sigma_2 \in A^O_{F\parallel G} \setminus A^O_G$.
- If $M_I(\sigma_1, \sigma_2) \neq \perp$, then $\sigma_2 \in A^I_{F'\parallel G} \setminus A^I_G$.

Remark 5. *By the above restriction on the error models, we get* $A^O_F = A^O_{F\otimes M_O}$ *and* $A^I_{F'} = A^I_{F'\otimes M_I}$. *Therefore, we get that* $F \otimes M_O$ *and* $F' \otimes M_I$ *are composable with* G *if* F *and* F' *are composable with* G.

The following lemma establishes that extending F with the error model does not change compatibility with G. Note that this would not be the case if the assumption on the error models was violated.

Lemma 6. *For all* BIAs F, G *and error model* M_I, M_O, *if* F *is compatible with* G, *then* $F \otimes M_O(M_I)$ *is compatible with* G.

The proof for the output error model M_O follows easily from the fact that any sequence of output actions in $(F \otimes M_O) \otimes G$ can be replayed in $F \otimes G$ by replacing those actions that are added by the error model in $F \otimes M_O$ with the original transitions from F. The case of input error model follows directly from the definition.

First, we establish the following preliminary lemma in anticipation of the main theorem. We need to show that property of incompatibility and of being error states is preserved even when the BIAs are extended with error models.

Lemma 7. *Let F, F', and G be BIAs with $shared(F', G) \subseteq shared(F, G)$, M_I an input error model, and M_O an output error model. Suppose that (p', q) is a state in $(F' \otimes G) \otimes M_I$ and $p \succeq p'$ for some alternating simulation relation $\succeq \subseteq Q_F \times Q_{F'}$ between $F \otimes M_O$ and $F' \otimes M_I$. Then,*

1. *(p', q) is an error state, then (p, q) is an error state;*
2. *(p', q) is an incompatible state, then (p, q) is an incompatible state.*

Proof.

1. From the definition of an error state, it follows that there exists an action $a \in shared(F', G) \subseteq shared(F, G)$ such that either,
   - $a \in A^O_{F'}(p')$ and $a \notin A^I_G(q)$, or
   - $a \in A^O_G(q)$ and $a \notin A^I_{F'}(p')$.

   In the former case as $p \succeq p'$, we have $a \in A^O_F(p)$, hence (p, q) is an error state. In the latter case from $a \notin A^I_{F'}(p')$ and $p \succeq p'$ follows that $a \notin A^I_F(p)$ and again (p, q) is an error state.
2. If (p', q) is an incompatible state in $(F' \otimes G) \otimes M_I$, it follows that an error state is autonomously reachable from (p', q) using only output actions. As $p \succeq p'$ the same sequence of actions can be replayed from the state (p, q): (i) the actions that change only the G component of the state are the same, and (ii) the actions that change the F' component can be simulated in F as $p \succeq p'$. We have either (a) that the replayed sequence reaches an error state before the end; or (b) the last reached state is an error state. The claim (b) follows from the previous part 1. In both cases, we get the required result.

□

The following lemma states that the broadcast interface automata enhanced with error models have the same properties on composition as interface automata. Note that the restrictions on error models imply that the *BIA* composed with an error model remains input deterministic on shared actions. Due to this fact the proof is a variation of a similar result for interface automata presented in [11].

Lemma 8. *Consider three BIAs F, G, and F' with an input error model M_I and an output error model M_O, such that $F \otimes M_O$ and G are composable and $shared(F', G) \subseteq shared(F, G)$. If $F \otimes M_O \sim G$ and $F \otimes M_O \succeq F' \otimes M_I$, then $F' \sim G$.*

Finally, we can prove the main theorem, showing that composition with a third interface can only decrease the distance. In the game between the composed systems, we construct a Player 2 strategy that (a) for the first component, uses the Player 2 strategy from the game $H_{F \otimes M_O, F' \otimes M_I}$; and (b) for the second component, copies the Player 1 transition.

There are two obstacles to this scheme of using the Player 2 strategy in the first component: (a) some of the actions become shared actions; and (b) some

of the states of the composed system may become unreachable due to their incompatibility. Using Lemma 7 and Lemma 6, we will overcome the obstacles.

Theorem 9. *Consider three* BIAs F, G, *and* F', *a quantitative objective* $f \in \{LimAvg, Disc_\lambda\}$, *and input (output) error models* M_I, M_O *such that* F *and* G *are composable, compatible, and* $shared(F', G) \subseteq shared(F, G)$. *Then,*

$$d^f(F \otimes M_O, F' \otimes M_I) \geq d^f((F \parallel G) \otimes M_O, (F' \parallel G) \otimes M_I).$$

Proof. We split the proof into two cases.
(a) Player 2 cannot avoid reaching s_{err} state in the game $H_{F \otimes M_O, F' \otimes M_I}$. This is the easier case and we will not present the details here.
(b) Player 2 can avoid reaching the s_{err} state in the game $H_{F \otimes M_O, F' \otimes M_I}$. We get that $F' \otimes M_I$ refines $F \otimes M_O$. Let $\succeq'$ be the maximal alternating simulation relation and furthermore, let π^2_* be the optimal positional Player 2 strategy in the game. By Remark 5 we get that $F \otimes M_O$ is composable with G, from Lemma 6 it follows that $F \otimes M_O$ is compatible with G and finally by Lemma 8 we get that the composition $F' \parallel G$ is not empty.

Using the relation $\succeq'$, we define an alternating simulation relation $\succeq^*$ by $(F \parallel G) \otimes M_O$ of $(F' \parallel G) \otimes M_I$ as follows:

$$(p, q) \succeq^* (r, s) \Leftrightarrow p \succeq' r \ \wedge \ q = s$$

for p and r states of F and F' and q and s states of G. We construct a positional Player 2 strategy π^2 in the game $H_{(F \parallel G) \otimes M_O, (F' \parallel G) \otimes M_I}$ based on the strategy π^2_*, such that for all Player 1 strategies π^1 the strategy π^2 will ensure that $f(out(\pi^1, \pi^2)) \leq d^f(F \otimes M_O, F' \otimes M_I)$.

We will match actions in the first component using the strategy π^2_*. Actions from the G component are going to be copied directly. This will ensure that every reachable Player 1 state $((p, q), \#, (r, s))$ satisfies $(p, q) \succeq^* (r, s)$. We have the following cases based on the kind of transition chosen by Player 1:

- **Unshared actions from G:** If Player 1 chooses the state $((p, q'), \sigma_I, (r, s))$, we have $\pi^2(((p, q'), \sigma_I, (r, s))) = ((p, q'), \#, (r, q'))$. This is possible as $q = s$. Similarly for a state $((p, q), \sigma_O, (r, s'))$ we define $\pi^2(((p, q), \sigma_O, (r, s'))) = ((p, s'), \#, (r, s'))$
- **Unshared input action from F:** If Player 1 chooses the state $((p', q), \sigma_I, (r, s))$, we have $\pi^2(((p', q), \sigma_I, (r, s))) = ((p', q), \#, (r', s))$ if $\pi^2_*(p', \sigma_I, r) = (p', \#, r')$. We have to make sure that the transition $((r, s), \sigma_I, (r', s))$ is not removed to ensure compatibility. In that case, from Lemma 7 and the fact that $(p, q) \succeq *(r, s)$, we would have that (p, q) is an incompatible state. However, the transition from compatible to incompatible state in the $(F \parallel G) \otimes M_O$ component is possible only by a Player 1 transition. Therefore, we have that Player 2 will not play an incompatible transition if Player 1 does not play an incompatible transition.
- **Unshared output action from F':** This case is similar to the previous, but simpler as output transitions are not removed to ensure compatibility.

- **Shared output action (input from G):** If Player 1 chooses the state $((p,q),\sigma_O,(r',s'))$, we have $\pi^2(((p,q),\sigma_O,(r',s'))) = ((p',s'),\#,(r',s'))$ if $\pi^2_*(p,\sigma_O,r') = (p',\#,r')$.
- **Shared output action (output from G):** This case is the trickiest due to the need to simulate inputs in the first component the "wrong" way (from F' to F).
  If Player 1 chooses the state $((p,q),\sigma_O,(r',s'))$, we have $\pi^2(((p,q),\sigma_O,(r',s'))) = ((p',s'),\#,(r',s'))$ where p' is the unique state reachable from p on the action σ_O. The existence of this action is argued here.
  - Firstly, due to input determinacy on shared actions, at most one state is reachable from p on action σ_O. Furthermore, there can be no transitions with action σ_O added by M_I as σ_O is shared with G.
  - Second, assuming that (p,q) is compatible, we have that at least one state is reachable from p on action σ_O. As in the above cases, we can argue that (p,q) is compatible.

  In the game $H_{F\otimes M_O,F'\otimes M_I}$, we can translate this step as follows. From $(p,\#,r)$, Player 1 chooses the successor (p',σ_O,r) (note that σ_O is an input action for F and F'); and then, $\pi^2_*((p',\sigma_O,r)) = (p',\#,r')$. The justification is as follows: Since, s_{err} is not visited, π^2_* has to choose a successor with the transition symbol σ_O (which is uniquely p', as above).

Let π^1 be an arbitrary Player 1 strategy. If we consider the play $\rho = out(\pi^1,\pi^2)$, (a) If the first case from the 5 above occurs, the transition weight is 0; and (b) For any of the other cases, the transition weight is the same as weights from a play ρ' in $H_{F\otimes M_O,F'\otimes M_I}$ conformant to π^2_*.

Therefore, we have that weights in ρ are weights in ρ', interspersed with some 0 weights. Hence, we get

$$d^f((F \parallel G)\otimes M_O,(F' \parallel G)\otimes M_I) \leq f(\rho') \leq f(\rho) \leq \nu(\pi^2_*) = d^f(F\otimes M_O,F'\otimes M_I)$$

proving the required result. □

3.3. Abstraction

In the classical boolean case, systems can by analyzed with the help of sound over- and under-approximations. We present the quantitative analogue of the soundness theorems for over- and under-abstractions. The distances between systems is bounded by the distances between their abstractions.

Given a *BIA* $F = (Q,q^0,A^I,A^O,\delta)$ and an equivalence relation $\sim \subseteq Q\times Q$ on the state space satisfying the following condition:

$$\text{If } p\sim p',\ p\xrightarrow{\sigma_I} q \text{ and } p'\xrightarrow{\sigma_I} q' \text{ for some } \sigma_I\in A^I \text{ are in } \delta \text{ then } q\sim q'$$

We define a $\forall\exists$ *abstraction* as a *BIA* $F^{\forall\exists} = (S,[q^0],A^I,A^O,\delta^{\forall\exists})$, where S consists of the equivalence classes of some equivalence relation on Q and

$$\begin{aligned}\delta^{\forall\exists} \;=\; & \{(s,\sigma_I,s') \mid \sigma_I\in A^I \text{ and } \forall q\in s,\exists q'\in s' : (q,\sigma_I,q')\in\delta\}\cup\\ & \{(s,\sigma_O,s') \mid \sigma_O\in A^O \text{ and } \exists q\in s,\exists q'\in s' : (q,\sigma_O,q')\in\delta\}\end{aligned}$$

Similarly we define the $\exists\forall$ abstraction *BIA* with the transition relation defined as follows:

$$\begin{aligned}\delta^{\exists\forall} \quad = \quad & \{(s,\sigma_I,s') \mid \sigma_I \in A^I \text{ and } \exists q \in s, \exists q' \in s' : (q,\sigma_I,q') \in \delta\} \cup \\ & \{(s,\sigma_O,s') \mid \sigma_O \in A^O \text{ and } \forall q \in s, \exists q' \in s' : (q,\sigma_O,q') \in \delta\}\end{aligned}$$

Theorem 10. *Let f be one of the objectives in $\{LimAvg, Disc_\lambda\}$ and F, G be arbitrary BIAs with M_O, M_I as error models, then the following inequalities hold:*

$$d^f(F^{\forall\exists} \otimes M_O, G^{\exists\forall} \otimes M_I) \leq d(F \otimes M_O, G \otimes M_I) \leq d^f(F^{\exists\forall} \otimes M_O, G^{\forall\exists} \otimes M_I)$$

Proof. Let π^2 be the optimal positional Player 2 strategy in the game $H_{F\otimes M_O, G\otimes M_I}$ we construct a positional Player 2 strategy π^2_* in $H_{F^{\forall\exists}\otimes M_O, G^{\exists\forall}\otimes M_I}$ that is going to ensure the needed value. When defining the strategy, we need to distinguish between two cases:

Input actions Let the state be (s_F, σ_I, s_G) for some $\sigma_I \in A^I$. We pick a state $q_F \in s_F$ and $q_G \in s_G$ such that strategy π^2 can ensure the value from the state (q_F, σ_I, q_G). Let π^2 reach state $(q_F, \#, q'_G)$ by playing action σ_I. Then π^2_* plays action σ_I to a state $(s_F, \#, [q'_G])$.

Output actions Similarly as in the previous case let the state be (s_F, σ_O, s_G) for some $\sigma_O \in A^O$. We pick a state $q_F \in s_F$ and $q_G \in s_G$ such that strategy π^2 ensures the value from the state (q_F, σ_O, q_G). If the state reached by π^2 is $(q'_F, \#, q_G)$ then π^2_* reaches a state $([q'_F], \#, q_G)$.

From every play conformant to π^2_* we can extract a play conformant to π^2 such that their values are equal. This concludes the first inequality. The proof of the second inequality is similar, but considers the optimal Player 1 strategy. □

3.4. Synthesis from Incompatible specifications

In practice it is often the case that systems are specified by multiple requirements on their behavior. In general, it is possible that there is no implementation satisfying all the criteria. However, one can try to obtain an implementation that has the distance to every requirement smaller than a certain threshold. In this subsection, we will show how to synthesize an implementation *BIA* given a set of incompatible specification *BIAs*.

The synthesis from incompatible requirements was presented for input enabled systems, that have states partitioned into input and output states in [4]. In our setting of *BIA*, the systems are not input-enabled, and in general there are states where both input and output actions are enabled.

We will say that the specifications *BIAs* S_i and output error models M_i for $1 \leq i \leq k$ are *incompatible* iff there exists no implementation *BIA* I satisfying for all i that the distance $d(S_i \otimes M_i, I) = 0$. We will first consider the decision problem:

Incompatible Specifications Decision Problem. Given a set of *BIA* specifications S_i together with the output error models M_i for $1 \leq i \leq k$ and a threshold

vector $\nu = (v_1, v_2, \ldots, v_k)$ decide whether there exists an implementation BIA I such that for all $1 \leq i \leq k$ we have that $d(S_i \otimes M_i, I) \leq v_i$.

In the current setting every synthesized implementation would not contain any output actions as having output actions gives more power to Player 1 in the alternating simulation game. Moreover, the implementation *BIA* should be an open reactive system that is deterministic in its output actions. Therefore, we additionally require that from every state in the implementation *BIA* I there exists exactly one output action edge.

We show that the incompatible specification decision problem is coNP-complete by a reduction to a decision problem in 2-player games with multiple limit-average threshold objectives, where edges are labeled with k-tuples of weights. The objective is formally defined as follows:

Multi Limit-average Threshold objective $MLimAvg_\nu$ where $\nu \in \mathbb{R}^k$ is a boolean objective. A path ρ is mapped to 1 if and only if there exists an $1 \leq i \leq k$, such that the *LimAvg* of the i^{th} component in the sequence of k-tuples of weights of ρ is greater than the i^{th} component of the vector ν.

Theorem 11. *The incompatible specification decision problem is coNP-complete for a fixed k.*

Proof. Given a set of specification *BIAs* S_i, output error models M_i for $1 \leq i \leq k$ and a threshold vector $\nu = (v_1, v_2, \ldots, v_k)$, we construct a 2-player game with a $MLimAvg_\nu$ objective such that Player 2 has a winning strategy iff there exists an implementation *BIA* I satisfying for all $1 \leq i \leq k$ that $d(S_i \otimes M_i, I) \leq v_i$.

For simplicity we assume that all the specifications share the input action alphabet A^I and the output action alphabet A^O. Given a specification *BIA* S_i composed with an output error model M_i we modify the weighted composed system $S_i \otimes M_i$ in the following way. In general the model is not input enabled, i.e., there are states where some of the input action may not be available. We make the system input enabled by adding an additional absorbing state $\perp$ and whenever an input action σ_I is not available in state $s \in Q_i$ we add an edge $s \xrightarrow{\sigma_I} \perp$. For every action in $A^I \cup A^O$ we add a selfloop on $\perp$ with weight $-\infty$. We refer to this modified system as $S_{\perp,i}$. Note that the system remains input deterministic. We construct a game $H^* = (S, S_1, S_2, E, s^0)$ with a $MLimAvg_\nu$ objective defined by the vector ν and a weight function $\omega : E \to \mathbb{N}^k$. The states of the game are $S = (Q^1 \times \ldots \times Q^k \times (A^I \cup A^O) \times \{D, E\}) \cup (Q^1 \times \ldots \times Q^k \times \{A, B, C\})$, where Q^i denotes the states of $S_{\perp,i}$. Player 1 states in the game are having A or B in the last component, the remaining states are Player 2 states, i.e., the last component of the state is C, D or E. We will refer to states with the last component A as A-states and similarly for B, C, D and E. For an intuitive understanding see Figure 3.

- Whenever the game is in an A-state $(s_1, s_2, \ldots, s_k, A)$ Player 1 can decide whether he wishes to play an *input* or an *output* action. The weight ω of the edges going out from A-states is 0 in all components. Also the first k components of the A-states do not change by playing the *input* or an *output* action.

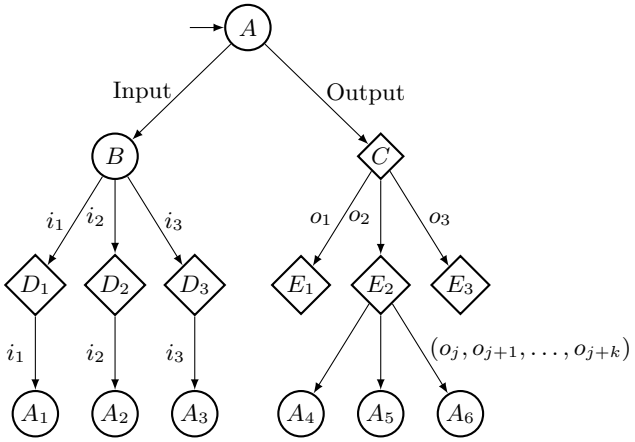

Figure 3: $MLimAvg_\nu$ game template

- If Player 1 chooses to play an *input* action the game reaches a B-state $(s_1, s_2, \ldots, s_k, B)$. Player 1 can then choose the next action σ_I from the input action alphabet A^I and reaches a D-state $(s'_1, s'_2, \ldots, s'_k, \sigma_I, D)$. The first k components are updated according to the transition function δ_i of $S_{\perp,i}$, i.e., for all i we have that $(s_i, \sigma_I, s'_i) \in \delta_i$. Note that the update is well defined due to input determinism. Let the state $\perp$ be reached in component j, then by the construction of $S_{\perp,j}$, Player 1 cannot exceed the threshold vector in j^{th} component, i.e., Player 1 looses the possibility to exceed the threshold in component j.

- In the D-state $(s_1, s_2, \ldots, s_k, \sigma_I, D)$ Player 2 is is not having any choice and is allowed only to repeat the input action σ_I played in the previous turn and the weight is 0 in all components.

- If Player 1 chooses to play an *output* action the game reaches a C-state $(s_1, s_2, \ldots, s_k, C)$ from which Player 2 selects an output $\sigma_O \in A^O$, and reaches E-state $(s_1, s_2, \ldots, s_k, \sigma_O, E)$, note that the first k components do not change during this transition. As in the previous case the weight is 0 in all components.

- The last case are the transitions from the E state $(s_1, s_2, \ldots, s_k, \sigma_O, E)$. Intuitively in this step Player 2 is responding to action played by Player 1 in the implementation (which in the end is the action σ_O played by Player 2 in the C state $(s_1, s_2, \ldots, s_k, C)$). He simultaneously responds in all the k specification components by playing a k-tuple of output actions $(\sigma_{O1}, \sigma_{O2}, \ldots, \sigma_{Ok})$. The A-state reachable can be chosen among states $(s'_1, s'_2, \ldots, s'_k, A)$ that satisfy for all $1 \leq i \leq k$ that $s_i \overset{\sigma_{Oi}}{\rightarrow} s'_i$. The penalty of this edge in the i^{th} component is $3 \cdot M_i(\sigma_O, \sigma_{Oi})$.

(a) Given an implementation *BIA* I and $1 \leq i \leq k$ consider the game $H_{S_i \otimes M_i, I}$ and let π_i^2 be the optimal Player 2 strategy in the game. By results on games with *LimAvg* objectives we have that the strategies π_i^2 are memoryless. We define a finite-memory strategy π^2 for Player 2 with state space of I as its memory.

- Whenever the state is $(s_1, s_2, \ldots, s_k, \sigma_I, D)$ and the memory is l, the strategy plays the only allowed action σ_I and updates its memory according to the implementation transition function δ_l, i.e., the new memory is l' for $(l, \sigma_I, l') \in \delta_l$. Note that due to input determinism the successor is well defined.

- If the state is $(s_1, s_2, \ldots, s_k, C)$ and the memory is l, the strategy π^2 plays the unique output action σ_O available in the state l of the implementation. The memory is then updated to l', where $(l, \sigma_O, l') \in \delta_l$. As the implementation is deterministic in its output actions, the location l' is well defined.

- Whenever the state is $(s_1, s_2, \ldots, s_k, \sigma_O, E)$ and the memory is l', the i^{th} component s_i' of the successor state $(s_1', s_2', \ldots, s_k', A)$ is chosen according to the strategy π_i^2 in the state (s_i, σ_O, l) of the game $H_{S_i \otimes M_i, I}$, i.e., $s_i' = \pi_i^2(s_i, \sigma_O, l)$. The action played in the k-tuple of actions is the action labeling the transition (s_i, σ_{Oi}, s_i') in $S_{\perp,i}$.

From every run ρ of the game H^* that is conformant to the finite-memory strategy π^2 we remove the A-states together with their outgoing edges and project the run ρ on its i^{th} component. By this construction we obtain a run ρ^i in the game $H_{S_i \otimes M_i, I}$. The *LimAvg* of the run ρ^i is either equal to $-\infty$ or the run is conformant to the optimal Player 2 strategy π_i^2 in the game. Moreover, if the *LimAvg* of ρ^i is not $-\infty$ then it equals to the *LimAvg* of the i^{th} projection of weights of run ρ. Therefore, the *LimAvg* of i^{th} component is bounded by the distance $d(S_i \otimes M_i, I)$. It follows that the Player 2 finite-memory strategy π^2 is a winning strategy.

(b) In the other direction assume there exists a finite-memory winning strategy π^2 for Player 2 in the game H^*. We will construct a product of the game H^* with the memory of strategy π^2. In the product all the transitions that are not played according to the strategy π^2 are removed, we will denote this construction by $H^* \restriction_{\pi^2}$.

In the following we merge the states from the template depicted in Figure 3. First note that every state C-state has a unique E-state successor in $H^* \restriction_{\pi^2}$. Similarly every E-state has a unique A-state successor and given an B-state and an input action σ_I the D-state successor (and therefore the A successor is determined as well). We merge all the states from the template into a single state, satisfying that the only enabled output action is the action labeling the transition from the C-state to the E-state with the deterministic update to the unique A-successor. Similarly we allow all input actions with the deterministic updates to the unique A-state successors.

Now assume towards contradiction that there exists i such that $d(S_i \otimes M_i, I) > v_i$, then the Player 1 strategy that ensures the value can be interpreted in the game H^* ensuring the same $LimAvg$ in the i^{th} component. Hence, we reach a contradiction with the assumption that the strategy π^2 is winning in the game H^*.

By the results of [8, 4] we have that solving $MLimAvg_\nu$ games for finite-memory strategies is coNP-complete in the size of the game H^*, which is polynomial in the size of the input, assuming fixed k. Therefore the Incompatible Specification Decision Problem is coNP-complete. □

One can also be interested into synthesizing an optimal implementation interface without explicitly specifying the threshold vector. This motivates the following problem:

Incompatible Specification Optimization Problem. Given specification *BIAs* S_i and output error models M_i for $1 \leq i \leq k$ and a bound $\epsilon > 0$ the optimization problem is to find an implementation *BIA* I^* such that for all other implementations I we have that $\max_{i\in\{1,\ldots,k\}} d(S_i \otimes M_i, I^*) \leq \max_{i\in\{1,\ldots,k\}} d(S_i \otimes M_i, I)$. We will refer to these implementations as ϵ-optimal implementations.

The problem of finding an ϵ-optimal implementation can be solved by performing a binary search on the space of thresholds. This leads to a *NEXP* algorithm for fixed k, ϵ, and W, where W is the absolute value of the maximum weight in the error models.

4. Case Studies

We present two case studies illustrating the interface simulation distances framework. In the first one, we describe a message transfer protocol for sending messages over an unreliable medium. This case study also serves to illustrate the behavior of the distance under interface composition. The second case study is on error correcting codes. In both cases, we use the limit average objective.

4.1. Message Transmission Protocol

Consider a *BIA Send* in Figure 4. It receives a message via the input *send?*. It then tries to send this message over an unreliable medium using the output *transmit!*. In response to *transmit?*, it can receive an input *ack?* signifying successful delivery of the message, or an input *nack?* signifying failure. It can then try to *transmit!* again (unboundedly many times), or it can abort using the output *abort!*. *Send* will be our specification interface.

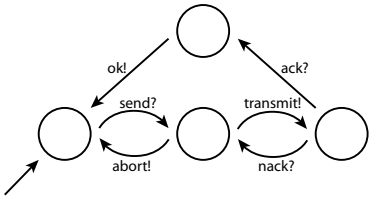

Figure 4: *BIA Send*

We consider two implementation interfaces *SendOnce* and *SendTwice* (Figures 5 and 6). *SendOnce* tries to send the message only once and if it does not succeed it reports a failure by sending *fail!* output. The second implementation

SendTwice tries to send the message twice and it reports a failure only if the transmission fails two times in a row. These implementation interfaces thus differ from the specification interface which can try to transmit the message an unbounded number of times. In particular, *SendOnce* or *SendTwice* do not refine the specification *Send* in the classical boolean sense.

In order to compute distances between *Send* on one hand, and *SendOnce* and *SendTwice*, we first define an error model. The output error model M_O we consider allows to play an output action *fail!* instead of *abort!* with penalty 1. We construct two games: $H_{Send \otimes M_O, SendOnce}$ and $H_{Send \otimes M_O, SendTwice}$. The goal of Player 1 is to make Player 2 cheat by playing *abort!* as often as possible. Therefore, whenever Player 1 has a choice between *ack?* and *nack?* the optimal strategy is going to play *nack?*. This agrees with the intuition that the difference between *Send* and *SendOnce* (*SendTwice*) is manifested in the case when the transmission fails.

The resulting distances are $d(Send \otimes M_O, SendOnce) = \frac{1}{4}$ and $d(Send \otimes M_O, SendTwice) = \frac{1}{6}$. According to the computed distances *SendTwice* is closer to the specification than *SendOnce*, as it tries to send the message before reporting a failure more times.

In order to illustrate the behavior of the interface simulation distance under composition of interfaces, we compose the interfaces *Send*, *SendOnce*, and *SendTwice* with an interface modeling the unreliable medium. The interface *Medium* in Figure 7 models an interface that fails to send a message at most two times in a row. The resulting systems $Send \parallel Medium$, $SendOnce \parallel Medium$ and $SendTwice \parallel Medium$ can be seen on Figure 10, 8 and 9. Again we can construct two games and compute their values. We obtain: $d((Send \parallel Medium) \otimes M_O, SendOnce \parallel Medium) = \frac{1}{8}$, and $d((Send \parallel Medium) \otimes M_O, SendTwice \parallel Medium) = 0$. As expected, when the *Medium* cannot fail two times in a row the implementation *SendTwice* is as good as the specification and therefore the distance would be 0. We remark that if we would change the model of the *Medium* to the one that never fails, both the distances would be 0.

4.2. Error Correcting Codes

Error correcting codes are a way to ensure reliable information transfer through a noisy environment. An error correcting code is for our purposes a

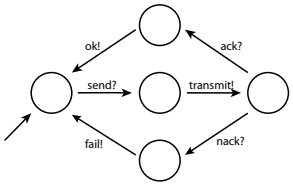

Figure 5: Implementation *SendOnce*

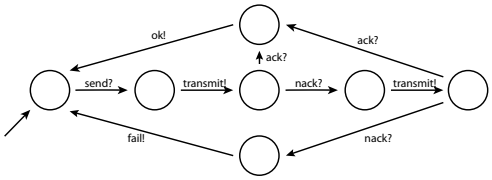

Figure 6: Implementation *SendTwice*

function that assigns every binary input string a fixed length *codeword* – again a binary string — that is afterwards transmitted. A natural way how to improve the chances of a correct transfer is to encode each message into a codeword by adding redundant bits. These codewords might get corrupted during the transmission, but the redundancy will cause that codewords are not close to each other (according to Hamming distance), and therefore it is possible to detect erroneous transfer, and sometimes even to correct some of the errors. Note that in what follows, we consider a situation where the only type of error allowed during the transmission is a bit flip.

We consider the well-known standard (n, M, d)-*code*, where n is the length of the code words, M is the number of different original messages, and d is the minimal Hamming distance between codewords. For instance, if we are given an error correcting code such that the minimal distance between codewords is 3 (i.e. an $(n, M, 3)$-code for some n and M), then whenever a single bit flip occurs we receive a string that is not among the codewords. However, there exists a unique codeword such that it has the minimal distance to the received string. The received string can be then corrected to this codeword.

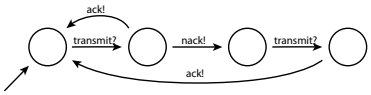

Figure 7: The *Medium*

We consider two different error correcting codes C_1 and C_2. Both codes encode 2 bit strings into 5 bit codewords. The codes are given in the following table:

$C_1(00) = 00000$	$C_1(01) = 00101$	$C_2(00) = 00000$	$C_2(01) = 01101$
$C_1(10) = 10110$	$C_1(11) = 11011$	$C_2(10) = 10110$	$C_2(11) = 11011$

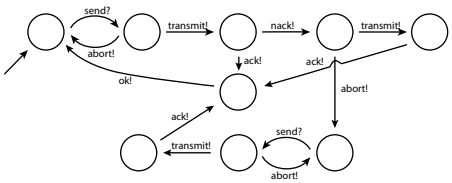

Figure 10: *Send* ∥ *Medium*

Note that C_1 is a $(5, 4, 2)$ code, i.e., its codewords have length 5, it encodes 4 words and the minimal Hamming distance between two codewords is 2. The minimal distance 2 ensures that when decoding the codeword a single bit flip can be detected, however, not corrected. On the other hand the code C_2 is a $(5, 4, 3)$ code and therefore can detect 2 bit flips and correct a single bit flip.

We model as BIAs the codes C_1 and C_2 and their transmission over a network

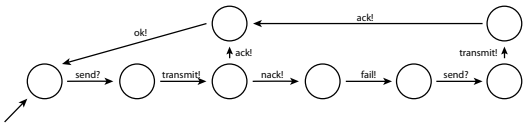

Figure 8: The *SendOnce* ∥ *Medium*

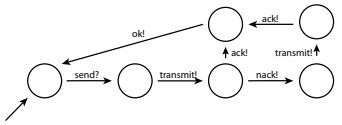

Figure 9: *SendTwice* ∥ *Medium*

where bit flips can occur. We construct the *BIAs* F_{C_1} and F_{C_2} according to the scheme presented in Figure 11 (this scheme is inside a loop and thus occurs repeatedly in both the BIAs). The first action is the input of a two-bit word that should be transmitted. The input word is then encoded according to C_1 (in F_{C_1}), or C_2 (in F_{C_2}). Then a sequence of five actions *flip* (or *noflip*) determines whether a bit flip occurs on the corresponding position. Depending on the *flip*/*noflip* sequence received and the error correcting code used, the final output is the decoding of the received string, with possibly some of the corrupted bits detected or repaired. More precisely, on an input x, F_{C_1} can detect a single bit flip, and could in this case send an *error* output. In case of more flips, it can even output a symbol different from the input x. Similarly, on an input x, F_{C_2}, in case of a single bit flip, can detect and correct the bit flip, and output the message x. If there are multiple flips it outputs a string different from the input x. As a specification interface, we consider a BIA F_{Spec} that uses the schema from Figure 11, but always outputs its input message, no matter what sequence of actions *flip* or *noflip* it receives.

We compose all three automata with a *BIA* F_{Error} modeling the allowed number of bit flips. Let F_{Error} allow only a single bit flip in 5 bits. The output error model M_O allows the Player 2 to play all the output 2 bit strings together with the *error* actions interchangeably. Then the corresponding values of the games are as follows: (a) $d((F_{C_{Spec}} \parallel F_{Error}) \times M_O, F_{C_1} \parallel F_{Error}) = 0$, and (b) $d((F_{C_{Spec}} \parallel F_{Error}) \times M_O, F_{C_2} \parallel F_{Error}) = \frac{1}{7}$. This shows the that the code C_2 performs better than the code C_1, as it can not only detect bit flips, but can also correct a single bit flip. In case we would use a F_{Error} that could do multiple bit flips in 5 bits, then distances of both codes would be the same.

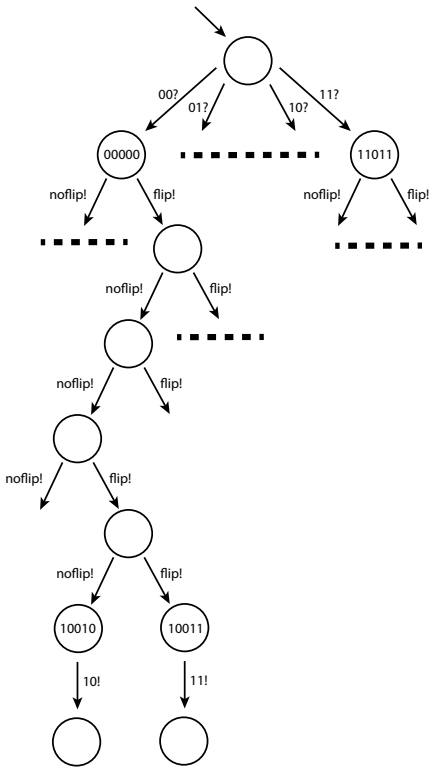

Figure 11: Code C_2

5. Conclusion

Summary. This paper extends the quantitative notion of simulation distances [5] to automata with inputs and outputs. This distance relaxes the boolean notion of refinement and allows us to measure the "desirability"

of an interface with respect to a specification, or select the best fitting interface from several choices that do not refine a specification interface in the usual boolean sense. We show that the interface simulation distance is a directed metric, i.e., it satisfies reflexivity and the triangle inequality. Moreover, the distance can only decrease when the interface are composed with a third interface, which allows us to decompose the specification into simpler parts. Furthermore, we show that the distance can be bounded from above and below by considering abstractions of the interfaces and one can automatically synthesize an interface out of incompatible requirements.

Future work. Defining the interface simulation distance for broadcast interface automata is one particular instance of a more general idea. We plan to examine the properties of the distance on other types of I/O automata, with differing notions of composition, with internal actions, or timed automata and automata modeling resource usage. Second, we plan to investigate probabilistic versions of the simulation distances, which would be useful in cases where there is a probability distribution on possible environment inputs. Third, we plan to perform larger case studies to establish which error models and accumulating functions (LimAvg, $Disc_\lambda$, etc.) are most useful in practice.

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

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

[3] P. Černý, M. Chmelík, T. Henzinger, and A. Radhakrishna. Interface simulation distances. In *GandALF*, pages 29–42, 2012.

[4] P. Černý, S. Gopi, T. Henzinger, A. Radhakrishna, and N. Totla. Synthesis from incompatible specifications. In *EMSOFT*, pages 53–62, 2012.

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

[6] P. Černý, T. Henzinger, and A. Radhakrishna. Simulation distances. *TCS*, 413(1):21–35, 2012.

[7] A. Chakrabarti, L. de Alfaro, T. Henzinger, and M. Stoelinga. Resource interfaces. In *EMSOFT*, pages 117–133, 2003.

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

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

[10] L. de Alfaro and T. Henzinger. Interface automata. In *ESEC / FSE*, pages 109–120, 2001.

[11] L. de Alfaro and T. Henzinger. Interface-based design. *Engineering theories of software intensive systems*, pages 83–104, 2005.

[12] L. de Alfaro, T. Henzinger, and M. Stoelinga. Timed interfaces. In *EMSOFT*, pages 108–122, 2002.

[13] L. de Alfaro, R. Majumdar, V. Raman, and M. Stoelinga. Game refinement relations and metrics. *Logical Methods in Computer Science*, 4(3), 2008.

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

[15] L. Doyen, T. Henzinger, B. Jobstmann, and T. Petrov. Interface theories with component reuse. In *EMSOFT*, pages 79–88, 2008.

[16] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, 1987.

[17] D. Jackson. Enforcing design constraints with object logic. In *SAS*, pages 1–21, 2000.

[18] A. Kanade, R. Alur, F. Ivančić, S. Ramesh, S. Sankaranarayanan, and K.C. Shashidhar. Generating and analyzing symbolic traces of Simulink/Stateflow models. In *Computer Aided Verification*, pages 430–445. Springer, 2009.

[19] K. Larsen, U. Nyman, and A. Wasowski. Modal I/O automata for interface and product line theories. *Programming Languages and Systems*, page 6479, 2007.

[20] T. Mancini, F. Mari, A. Massini, I. Melatti, F. Merli, and E. Tronci. System level formal verification via model checking driven simulation. In *Computer Aided Verification*, pages 296–312. Springer, 2013.

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

[22] D. Yellin and R. Strom. Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst.*, 19(2):292–333, 1997.

[23] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158(1&2):343–359, 1996.