Theoretical Computer Science 560 (2014) 348–363

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Interface simulation distances ☆

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

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

ARTICLE INFO

Article history:
Received 4 March 2013
Received in revised form 18 June 2014
Accepted 25 August 2014
Available online 10 September 2014

Keywords:
Alternating simulation
Quantitative analysis
Games

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.

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 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

☆ This research was partially supported by the European Research Council (ERC) Advanced Investigator Grant (267989: 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 (2011).

* Corresponding author.

http://dx.doi.org/10.1016/j.tcs.2014.08.019
0304-3975/

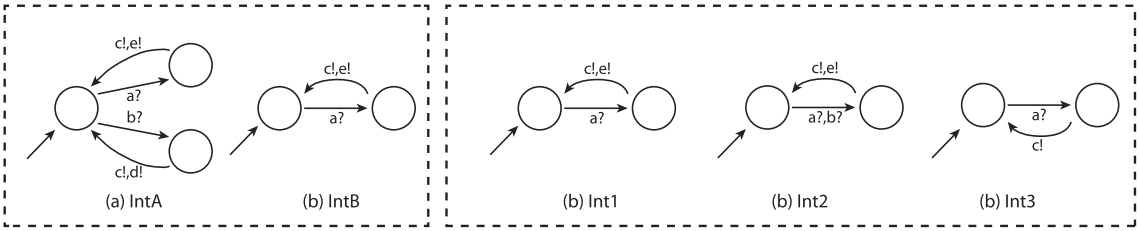

Fig. 1. Interface examples.

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 Fig. 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