Theoretical Computer Science 597 (2015) 18–39

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Error-pruning in interface automata ☆

Ferenc Bujtor *, Walter Vogler

Inst. f. Informatik, Universität Augsburg, Germany

ARTICLE INFO

Article history:
Received 9 April 2014
Received in revised form 18 May 2015
Accepted 21 June 2015
Available online 3 July 2015
Communicated by J.-F. Raskin

Keywords:
Concurrency
Interface automata
Parallel composition
Optimistic/pessimistic approach
Fully abstract precongruence

ABSTRACT

De Alfaro and Henzinger introduced interface automata to model and study behavioural types. These come with alternating simulation as refinement and with a specific parallel composition: if one component receives an unexpected input, this is regarded as an error and the resp. error states are removed with a special pruning operation. In this paper, we return to the foundations of interface automata and study how refinement and parallel composition should be defined best.

We take as basic requirement that an implementation must be error-free, if the specification is. For three variants of error-free, we consider the coarsest precongruence for parallel composition respecting the basic requirement. We find that pruning proves to be relevant in all cases and point out an important subtlety for systems that are not input-deterministic.

1. Introduction

Interface automata as introduced by de Alfaro and Henzinger e.g. in [7] are an abstract description of the communication behaviour of a system or component in terms of input and output actions. Based on this behavioural type, one can study whether two systems are compatible if put in parallel, and one can define a refinement for specifications. Essential for such a setting is that the refinement relation is a precongruence for parallel composition; in particular, if we refine two compatible specifications, it must be guaranteed that the refined specifications are compatible again.

A basic intuition here is that outputs are under the control of the respective system: if one component in a composition provides an output for another, the latter must synchronize by performing the same action as input; if this is not possible, the whole system might malfunction – such a catastrophic error state has to be avoided. In contrast to the I/O-automata of [12], interface automata are not input enabled. Instead, a missing input in a state corresponds to the requirement that an environment must not send this input to this state.

There are two essential design decisions in the approach of [7] that we will scrutinize in this paper. First, the approach is optimistic: an error state is not a problem, if it cannot be reached in a helpful environment. This is reflected in the details of parallel composition, where from a standard product automaton all states are removed that can reach an error state just by locally controlled, i.e. output and internal, actions (often called *pruning*). Although this definition has some intuitive justification, its details appear somewhat arbitrary. This is also the case for the second decision to take some alternating

☆ This research was supported by DFG-project 'Foundations of Heterogenous Specifications Using State Machines and Temporal Logic' VO 615/12-1. An extended abstract with the same title has appeared in SOFSEM 2014, LNCS 8327, pp. 162–173.

* Corresponding author.
E-mail addresses: bujtor@informatik.uni-augsburg.de (F. Bujtor), walter.vogler@informatik.uni-augsburg.de (W. Vogler).

http://dx.doi.org/10.1016/j.tcs.2015.06.047
0304-3975/

simulation as refinement relation. Actually, the same authors used a slightly different relation for a slightly larger class of automata in the earlier [6]; no real argument is given for the change.

Here, we will work out to what degree these design decisions can be *justified* from some more basic and, hopefully, more agreeable ideas. We model components as labelled transition systems (LTSs) with disjoint input and output actions and an internal action, quite like the interface automata of [7]. So as not to exclude any possibilities prematurely, our LTS have explicit error states. For these Error-IO-Transition Systems (EIO), we consider a standard parallel composition where, additionally, error states occur as described above; a composed system also reaches an error state if one of the components reaches one.

An undisputable requirement for a refinement relation is that an error-free specification should only be refined by an error-free system. This can be understood as a basic refinement relation, which is parametric in the exact meaning of error-free: in the optimistic view, error-free means that no error state can be reached by locally controlled actions only; in the pessimistic view (cf. e.g. [2]), a system is error-free only if no error state is reachable at all.

For modular reasoning, which is at the heart of the approach under study, the refinement relation $\sqsubseteq$ must be a precongruence: if a component of a parallel composition is replaced by a refinement, the composition itself gets refined, i.e. $S_1 \sqsubseteq S_2$ implies $S_1 \mid S \sqsubseteq S_2 \mid S$. Since the basic relations fail to be precongruences in each case, we will characterize (or at least approximate) the resp. coarsest precongruence for parallel composition that is contained in the basic relation. Such a *fully abstract* precongruence is optimal for preserving error-freeness, since it does not distinguish components unnecessarily.

In the optimistic case, the precongruence can be characterized as (componentwise) inclusion for a pair of trace sets; the definition of one of these uses pruning on traces. With this characterization we can prove that, essentially, each EIO is equivalent w.r.t the precongruence to one without error states, where the latter can be obtained by pruning the former almost as in [7]. Thus, we can work with EIO without error states, i.e. with interface automata and (almost) with the parallel composition of [7], but our pruning is *proven* to be correct.

While this justifies the first design decision in [7], our precongruence shows that alternating simulation is unnecessarily strict. This is not really new. A setting with input and outputs where unexpected inputs lead to errors has been studied long before [7] for speed-independent (thus asynchronous) circuits by Dill in [8]. The difference is that Dill does not start from an operational model as we do (in particular, there is no parallel composition for LTS), but on a semantic level with pairs of trace sets; he requires these pairs to be input enabled. On this semantic level, he also uses pruning; a normalized form of his pairs coincides with our pairs. Essentially, the full abstraction result can also be found in [4], though for a slightly different parallel composition and only for a congruence. Since that paper starts from a declarative approach, our presentation and proofs are more direct, and they prepare the reader for the succeeding sections.

In [4], EIO (called Logic IOLTS there) are seen as an alternative framework to interface automata, and an error state is actually added to normalize an EIO. We see error states only as a tool to study interface automata and would prefer to remove them in the end; with this view, we discovered a subtle point about pruning. Interface automata in [7] are deterministic w.r.t. input actions. Since we do not require this here, our pruning is a bit *different* from the one in [7]. In fact, the interface automata in [6] are also not input deterministic, but pruning used there is *the same* as the one in [7]. As a consequence, Theorem 1 of [6] claiming associativity for parallel composition is wrong; in our setting, it is easily proven.

It might seem that we have actually prescribed pruning in our optimistic approach since we consider only locally reachable errors as relevant and pruning removes exactly those states that can reach an error locally. To fortify the justification of pruning, we turn to a 'hyper-optimistic' approach next, where only internally reachable errors are relevant. With this *more generous* notion of error-free we obtain a slightly *stricter* precongruence and characterize it. The characterization is again based on pruning; the new idea is to extend traces with a set of outputs removed during pruning. This is an interesting precongruence but, compared to our first one, it looks unnecessarily complicated.

Finally, we turn to a pessimistic approach where every reachable error is relevant as advocated e.g. in [2]. For this case, we only approximate the fully abstract precongruence: we describe a precongruence contained in the resp. basic relation, which is based on three trace sets and again employs pruning. We sketch how one might get the fully abstract precongruence, but this will be technically so involved as to make it unattractive. Even without a characterization, we can show that the fully abstract precongruence is again *stricter* than the optimistic one, although the notion of being error-free is *less generous*.

The next section will explain some basic notions. Sections 3 to 5 describe in turn our results for the optimistic, hyper-optimistic and pessimistic approach. Finally, in Section 6, we conclude with a comparison and give arguments why we will prefer the optimistic variant in the future.

2. Definitions and notation

First we define our scenario. In interface automata, internal actions have different names. The sets of all action names must be disjoint for two automata to be composable; hence, standard α-conversion for the names of internal actions is not fully supported.[1] To improve this, our EIOs have just one internal, unobservable action τ. Furthermore, they have as additional component a set of error states; such states can be created in a parallel composition.

[1] Strictly speaking, α-conversion in a refinement step is allowed in [7], since there is no requirement regarding the alphabets of internal actions. But this is certainly an oversight since, as an effect, refinement does not preserve compatibility, because it does not even preserve composability; cf. Proposition 13.