Information and Computation 249 (2016) 28–75

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Dynamic input/output automata: A formal and compositional model for dynamic systems

Paul C. Attie[a,*], Nancy A. Lynch[b]

[a] Department of Computer Science, American University of Beirut, Beirut, Lebanon
[b] MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA

ARTICLE INFO

Article history:
Received 24 June 2013
Received in revised form 15 February 2016
Available online 21 March 2016

Keywords:
Dynamic systems
Formal methods
Semantics
Automata
Process creation
Mobility

ABSTRACT

We present dynamic I/O automata (DIOA), a compositional model of dynamic systems. In DIOA, automata can be created and destroyed dynamically, as computation proceeds, and an automaton can dynamically change its signature, i.e., the set of actions in which it can participate.

DIOA features operators for *parallel composition*, *action hiding*, *action renaming*, a notion of *automaton creation*, and a notion of behavioral subtyping by means of *trace inclusion*. DIOA can model mobility, using signature modification, and is hierarchical: a dynamically changing system of interacting automata is itself modeled as a single automaton.

We also show that parallel composition, action hiding, action renaming, and (subject to some technical conditions) automaton creation are all monotonic with respect to trace inclusion: if one component is replaced by another whose traces are a subset of the former, then the set of traces of the system as a whole can only be reduced.

1. Introduction

Many modern distributed systems are *dynamic*: they involve changing sets of components, which are created and destroyed as computation proceeds, and changing capabilities for existing components. For example, programs written in object-oriented languages such as Java involve objects that create new objects as needed, and create new references to existing objects. Mobile agent systems involve agents that create and destroy other agents, travel to different network locations, and transfer communication capabilities.

To describe and analyze such distributed systems rigorously, one needs an appropriate *mathematical foundation*: a state-machine-based framework that allows modeling of individual components and their interactions and changes. The framework should admit standard modeling methods such as parallel composition and levels of abstraction, and standard proof methods such as invariants and simulation relations. As dynamic systems are even more complex than static distributed systems, the development of practical techniques for specification and reasoning is imperative. For static distributed systems and concurrent programs, compositional reasoning is proposed as a means of reducing the proof burden: reason about small components and subsystems as much as possible, and about the large global system as little as possible. For dynamic systems, compositional reasoning is *a priori* necessary, since the environment in which dynamic software components (e.g., software agents) operate is continuously changing. For example, given a software agent B, suppose we then refine B to

* Corresponding author.
E-mail addresses: paul.attie@aub.edu.lb (P.C. Attie), lynch@csail.mit.edu (N.A. Lynch).

http://dx.doi.org/10.1016/j.ic.2016.03.008
0890-5401/© 2016 Elsevier Inc. All rights reserved.

generate a new agent A, and we prove that A's externally visible behaviors are a subset of B's. We would like to then conclude that replacing B by A, within *any* environment does not introduce new, and possibly erroneous, behaviors.

One issue that arises in systems where components can be created dynamically is that of *clones*. Suppose that a particular component is created twice, in succession. In general, this can result in the creation of two (or more) indistinguishable copies of the component, known as clones. We make the fundamental assumption in our model that this situation does not arise: components can always be distinguished, for example, by a logical timestamp at the time of creation. This absence of clones assumption does not preclude reasoning about situations in which an automaton A_1 cannot be distinguished from another automaton A_2 *by the other automata in the system*. This could occur, for example, due to a malicious host which "replicates" agents that visit it. We distinguish between such replicas at the meta-theoretic level by assigning unique identifiers to each. These identifiers are not available to the other automata in the system, which remain unable to tell A_1 and A_2 apart, for example in the sense of the "knowledge" [16] about A_1 and A_2 which the other automata possess.

Static mathematical models like I/O automata [24] could be used to model dynamic systems, with the addition of some extra structure (special Boolean flags) for modeling dynamic aspects. For example, in [22], dynamically-created transactions were modeled as if they existed all along, but were "awakened" upon execution of special *create* actions. However, dynamic behavior has by now become so prevalent that it deserves to be modeled directly. The main challenge is to identify a small, simple set of constructs that can be used as a basis for describing most interesting dynamic systems.

In this paper, we present our proposal for such a model: the *Dynamic I/O Automaton (DIOA) model*. Our basic idea is to extend I/O automata with the ability to change their signatures dynamically, and to create other I/O automata. We then combine such extended automata into global *configurations*. Our model provides:

1. parallel composition, action hiding, and action renaming operators;
2. the ability to dynamically change the signature of an automaton; that is, the set of actions in which the automaton can participate;
3. the ability to create and destroy automata dynamically, as computation proceeds; and
4. a notion of externally visible behavior based on sets of traces.

Our notion of externally visible behavior provides a foundation for abstraction, and a notion of behavioral subtyping by means of trace inclusion. Dynamically changing signatures allow us to model mobility, by enforcing the constraint that only automata at the same location may synchronize on common actions. This capability is not present in a static model with extra structure (e.g., boolean flags). Modeling a mobile agent in a static setting would be difficult at best, and would result in a contrived and over-complicated model (how would you simulate location and signature change?) that would lose the benefits of simple and direct representation that our model affords.

Our model is hierarchical: a dynamically changing system of interacting automata is itself modeled as a single automaton that is "one level higher." This can be repeated, so that an automaton that represents such a dynamic system can itself be created and destroyed. This allows us to model the addition and removal of entire subsystems with a single action. This would also be quite difficult to represent naturally in a static model.

As in I/O automata [24,25], there are three kinds of actions: input, output, and internal. A trace of an execution results by removing all internal actions, replacing each state by its external signature (i.e., the input and output actions), and finally replacing blocks of identical external signatures by a single representative. We use the set of traces of an automaton as our notion of external behavior. We show that parallel composition is monotonic with respect to trace inclusion: if we have two systems $A = A_1 \parallel \cdots \parallel A_i \parallel \cdots \parallel A_n$ and $A' = A_1 \parallel \cdots \parallel A'_i \parallel \cdots \parallel A_n$ consisting of n automata, executing in parallel, then if the traces of A_i are a subset of the traces of A'_i (which it "replaces"), then the traces of A are a subset of the traces of A'. We also show that action hiding (convert output actions to internal actions) and action renaming (change action names using an injective map) are monotonic with respect to trace inclusion, and, finally, we show that, if we have a system X in which an automaton A is created, and a system Y in which an automaton B is created "instead of A", and if the traces of A are a subset of the traces of B, then the traces of X will be a subset of the traces of Y, but only under certain conditions. Specifically, systems X and Y must create A and B, respectively, in "corresponding" states. A state of X and a state of Y correspond iff they are the last states of finite executions of X and Y which have the same trace. Otherwise, monotonicity of trace inclusion can be violated by having the system X create the replacement A in more contexts than those in which Y creates B, resulting in X possessing some traces which are not traces of Y. This phenomenon appears to be inherent in situations where the creation of new automata can depend upon global conditions (as in our model) and can be independent of the externally visible behavior (trace). Our monotonicity results imply that trace equivalence is a congruence with respect to parallel composition, action hiding, and action renaming.

Our results enable a refinement methodology for dynamic systems that is independent of specific methods of establishing trace inclusion. Different automata in the system can be refined using different methods, e.g., different simulation relations such as forward simulations or backward simulations, or by using methods not based on simulation relations. This provides more flexibility in refinement than a methodology which, for example, shows that forward simulation is monotonic with respect to parallel composition, since in the latter every automaton must be refined using forward simulation.

We defined the DIOA model initially to support the analysis of *mobile agent systems*, in a joint project with researchers at Nippon Telephone and Telegraph. Creation and destruction of agents are modeled directly within the DIOA model. Other