
Checking Semantical Consistency based on Observational Simulations

Vlad Rusu* — Dorel Lucanu**

* *Inria Rennes Bretagne Atlantique
Campus de Beaulieu
35042 Rennes Cedex, France
Vlad.Rusu@inria.fr*

** *Department of Computer Science
Alexandru Ioan Cuza University
Iasi, Romania
dlucanu@info.uaic.ro*

ABSTRACT. We study the semantical consistency between two operational specifications of systems at two different levels of abstraction. We propose the notion of observational simulations to compare systems specified in terms of observational transition systems. We give a procedure for automatically detecting the absence of observational simulations between two observational transition systems (hence, their semantical inconsistency). The procedure also automatically finds executions of one specification that simulate a given execution of the other specification.

RÉSUMÉ. Nous nous intéressons ici à la cohérence sémantique entre deux spécifications opérationnelles d'un même système, situées à deux niveaux d'abstraction différents. Nous proposons la notion de simulations observationnelles pour comparer les systèmes spécifiés en terme de systèmes de transitions observationnels. Nous donnons une procédure qui permet de détecter l'inexistence d'une simulation observationnelle entre deux systèmes de transitions observationnels donnés (et donc, leur incohérence sémantique). Notre procédure détecte également des exécutions d'une des spécifications qui simulent une exécution donnée de l'autre spécification.

KEYWORDS: semantical consistency, observational simulations

MOTS-CLÉS: cohérence sémantique, simulations observationnelles

1. Introduction and Motivation

Assume that one wants to build a complex and critical embedded system. A reasonable approach is to first give a high-level description, expressed in a dedicated language such as, e.g., MARTE. From this high-level description, a lower -level de-

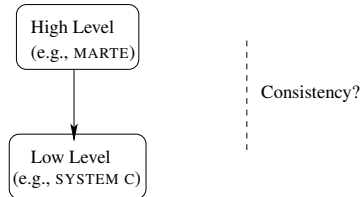


Figure 1. *Semantical consistency.*

description of the system can be generated, e.g., using *model transformations* - for example, the lower-level description of the system could be written in SYSTEM C. (We mention embedded systems, the MARTE and SYSTEM C languages, and model transformations, to motivate our approach - the approach is not limited only to these technologies.) A natural question that arises is whether the two representations of the system are *semantically consistent*: do the executions of one level "match" with the executions of the other one? And if this is the case, with which ones? Also, higher-level and lower-level executions may have different "granularities" - one step at one level may correspond to several steps at the other level. Then, what does "matching" of executions mean? This paper proposes a tentative formal framework and solutions for these problems.

The paper is organised as follows. In Section 2 we define *observational transition systems* and *observational simulations* as general, abstract models for, respectively, the executions of systems, and the "matching" between those executions. Then, *semantical consistency* amounts to *checking whether an observational simulation* between two given observational transition systems *exists*; and, if this is the case, to compute one. To solve these problems, we propose a construction called a *simulation tree* over two observational transition systems. We show that a certain condition on the simulation tree is equivalent to the existence of an observational simulation between the two observational transition systems (from which the simulation tree was constructed). This provides us with a procedure for automatically detecting the inexistence of an observable simulation, by exploring the simulation tree up to a finite depth. The procedure can also be used to find executions of one system that simulate a given, finite execution of the other one. In our hypothetical embedded-system design, executions obtained by *emulating* a SYSTEM C description of an embedded system could be "traced back" to the higher-level MARTE description; hence, if the emulation revealed an error, our procedure traces the error back to the high-level description, enabling users to fix it. In Section 3 we present related and future work and conclude.

2. Observational Transition Systems and their Observational Simulations

Definition 1 (Observational Transition System (OTS) [OGA 08]) *An OTS is a tuple $\langle A, a_0, \rightarrow, O, \omega \rangle$ where A is a nonempty, possibly infinite set of states; $a_0 \in A$ is the initial state; $\rightarrow \subseteq A \times A$ is the (total) transition relation; O is a nonempty, possibly infinite set of observations; and $\omega : A \rightarrow O$ is the observation function.*

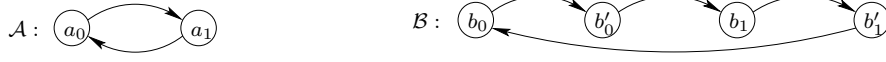


Figure 2. Two observational transition systems \mathcal{A}, \mathcal{B} with observation set $\{0, 1\}$ and observation functions defined by $\omega_{\mathcal{A}}(a_i) = \omega_{\mathcal{B}}(b_i) = \omega_{\mathcal{B}}(b'_i) = i$, for $i = 0, 1$.

That is, observational transition systems are just transition systems, together with an observation domain and an observation function mapping states to observations. The following notions are rather standard for transition systems: an *execution* is a possibly infinite sequence of states $\pi = a_0, \dots, a_n, \dots$ such that for $a_i \rightarrow a_{i+1}$ for $i = 0, 1, \dots$. For technical reasons we also consider executions that start in states different from the initial state. For a finite execution π , we denote its *length* by $len(\pi)$. Also, for a state a , we denote by $exec(a)$ the set of executions π such that $\pi(0) = a$.

Definition 2 (Matching, inspired from [MAR 08]) For two observational transition systems $\mathcal{A} = (A, a_0, \rightarrow_{\mathcal{A}}, O, \omega_{\mathcal{A}})$ and $\mathcal{B} = (B, b_0, \rightarrow_{\mathcal{B}}, O, \omega_{\mathcal{B}})$ having the same observation set O , and for two executions π of \mathcal{A} and ρ of \mathcal{B} , we say that π is O -matched by ρ if there exist two strictly increasing functions $\alpha, \beta : \mathbb{N} \rightarrow \mathbb{N}$ such that $\alpha(0) = \beta(0) = 0$, and such that for all $i, j, k \in \mathbb{N}$, $\alpha(i) \leq j < \alpha(i+1)$ and $\beta(i) \leq k < \beta(i+1)$ imply $\omega_{\mathcal{B}}(\rho(j)) = \omega_{\mathcal{A}}(\pi(k))$.

Example 1 Consider in Fig. 2 and the (unique) infinite executions $\pi \in exec(a_0)$ and $\rho \in exec(b_0)$. Then, each of these executions is $\{0, 1\}$ -matched by the other one.

Definition 3 (Observational Simulation) For observational transition systems $\mathcal{A} = (A, a_0, \rightarrow_{\mathcal{A}}, O, \omega_{\mathcal{A}})$ and $\mathcal{B} = (B, b_0, \rightarrow_{\mathcal{B}}, O, \omega_{\mathcal{B}})$ and states $a \in A$, $b \in B$, we say that a is observationally simulated by b , denoted by $a \lesssim_O b$, if for all $\pi \in exec(a)$, there exists $\rho \in exec(b)$ such that ρ O -matches π . We say that \mathcal{A} is observationally simulated by \mathcal{B} , denoted by $\mathcal{A} \lesssim_O \mathcal{B}$, if their initial states a_0, b_0 satisfy $a_0 \lesssim_O b_0$.

Example 2 Each of the observational transition systems \mathcal{A}, \mathcal{B} in Figure 2 is observationally simulated by the other one - based on the matching executions in Example 1.

Next, we propose a construction called a *simulation tree* and study its properties.

Definition 4 (Simulation Tree) Given two observational transition systems $\mathcal{A} = (A, a_0, \rightarrow_{\mathcal{A}}, O, \omega_{\mathcal{A}})$ and $\mathcal{B} = (B, b_0, \rightarrow_{\mathcal{B}}, O, \omega_{\mathcal{B}})$, the simulation tree of \mathcal{A} by \mathcal{B} is the tree $\mathcal{T}_{\mathcal{A}, \mathcal{B}}$ whose root r , set of nodes \mathcal{N} , and set of edges \mathcal{E} , are defined as follows:

- $r = \langle a_0, \{b_0\} \rangle$ if $\omega_{\mathcal{A}}(a_0) = \omega_{\mathcal{B}}(b_0)$, $r = \langle a_0, \emptyset \rangle$ otherwise
- $\mathcal{N} \subseteq \{ \langle a, S \rangle \in A \times \mathcal{P}(B) \mid \forall b \in S. \omega_{\mathcal{B}}(b) = \omega_{\mathcal{A}}(a) \}$;
- \mathcal{E} and \mathcal{N} are the smallest sets satisfying the following conditions:

- if $\langle a, S \rangle \in \mathcal{N}$, $a \rightarrow_{\mathcal{A}} \hat{a}$, and $\omega_{\mathcal{A}}(\hat{a}) = \omega_{\mathcal{A}}(a)$, then $\langle \hat{a}, S \rangle \in \mathcal{N}$ and $(\langle a, S \rangle, \langle \hat{a}, S \rangle) \in \mathcal{E}$;

- if $\langle a, S \rangle \in \mathcal{N}$, $a \rightarrow_{\mathcal{A}} \hat{a}$, and $\omega_{\mathcal{A}}(\hat{a}) \neq \omega_{\mathcal{A}}(a)$: let $\hat{S} = \{\hat{b} \in B \mid \exists b \in S. \exists \rho \in \text{exec}(b). \rho(\text{len}(\rho)) = \hat{b} \wedge \omega_{\mathcal{A}}(\hat{a}) = \omega_{\mathcal{B}}(\hat{b}) \wedge \forall k \in \{0.. \text{len}(\rho)-1\}. \omega_{\mathcal{B}}(\rho(k)) = \omega_{\mathcal{B}}(b)\}$. Then, $\langle \hat{a}, \hat{S} \rangle \in \mathcal{N}$ and $(\langle a, S \rangle, \langle \hat{a}, \hat{S} \rangle) \in \mathcal{E}$.

For example, in Figure 2 the simulation tree of \mathcal{A} by \mathcal{B} starting at (a_0, b_0) is a linear sequence infinitely repeating the pattern $\langle a_0, \{b_0\} \rangle - \langle a_1, \{b_1\} \rangle$. In general, the nodes of a simulation tree $\mathcal{T}_{\mathcal{A}, \mathcal{B}}$ have the form $\langle a, S \rangle$, satisfying the invariant that all the states in their second component : $b \in S$, have the same observations as the state in their first component a . As usual in trees, there are edges from each node to its children; the children of a node "say" what the automaton \mathcal{B} "does" when the automaton \mathcal{A} "makes" a step, depending on whether the destination of the step has the same observation as its origin, or not. Accordingly, the set of children of a node $\langle a, S \rangle$ is the union of:

- the set of nodes $\langle \hat{a}, S \rangle$ such that $a \rightarrow_{\mathcal{A}} \hat{a}$ and $\omega_{\mathcal{A}}(\hat{a}) = \omega_{\mathcal{A}}(a)$; here, the first component \hat{a} is obtained by taking one execution step from a , and it has the same observation as a ; then, the second component S remains unchanged. Note that this is enough to preserve the above-mentioned tree invariant regarding observations.

- the set of nodes $\langle \hat{a}, \hat{S} \rangle$ such that $a \rightarrow_{\mathcal{A}} \hat{a}$ and $\omega_{\mathcal{A}}(\hat{a}) \neq \omega_{\mathcal{A}}(a)$: now, \hat{a} is obtained by one execution step from a , but its observation is *different* from that of a . Then, the transition system \mathcal{B} "builds" the set \hat{S} by taking executions ρ from states $b \in S$ to states $\hat{b} = \rho(\text{len}(\rho)) \in \hat{S}$, such that (i) the observation of \hat{b} equals the "new" observation $\omega_{\mathcal{A}}(\hat{a})$, and (ii) ρ does not have a proper subsequence with that property. Note that this also preserves the tree invariant on nodes regarding observations. Note also that such states \hat{b} may not exist, in which case the resulting node is $\langle \hat{a}, \emptyset \rangle$.

These observations are formalised in the following theorem - the paper's main result.

Theorem 1 *For observational transition systems \mathcal{A}, \mathcal{B} , $\mathcal{A} \lesssim_O \mathcal{B}$ iff for all nodes $\langle a, S \rangle$ of $\mathcal{T}_{\mathcal{A}, \mathcal{B}}$, $S \neq \emptyset$. Moreover, for any execution π of \mathcal{A} , consider the path of the form $(\langle \pi(i), S_i \rangle)_{i \geq 0}$ in $\mathcal{T}_{\mathcal{A}, \mathcal{B}}$, whose projection on its first component is π . Then, all executions ρ of \mathcal{B} satisfying the condition $\rho(i) \in S_i$ for all $i \geq 0$ are O -matches for π .*

The theorem contains two statements. The first one naturally suggests a procedure for detecting the absence of an observational simulation: explore a *finite prefix* of $\mathcal{T}_{\mathcal{A}, \mathcal{B}}$ - since $\mathcal{T}_{\mathcal{A}, \mathcal{B}}$ is infinite, in general - and stop if node of the form $\langle a, \emptyset \rangle$ is found; we say the simulation tree found a *counterexample*. An equivalent, optimised procedure consists in stopping the exploration when a node of the form $\langle a', S' \rangle$ is encountered during the exploration, and there already is a node $\langle a', S \rangle$ with $S \subseteq S'$ in the tree, and a path from $\langle a', S \rangle$ to $\langle a', S' \rangle$. This is because the set S of states of \mathcal{B} possibly simulating a' can only grow, and S is already nonempty - otherwise the exploration would have already stopped at $\langle a', S \rangle$. We say the simulation tree found a *circularity*. (Such counterexamples and circularities are illustrated on a simple example below.)

Probably the most useful practical application of this work is finding executions of \mathcal{B} that match a given finite execution of \mathcal{A} , as discussed at the end of Section 1. The second statement in Theorem 1 says that the simulation tree can be used exactly for that purpose (actually, one only needs to build the *finite, linear* path $(\langle \pi(i), S_i \rangle)_{0 \leq i \leq \text{len}(\pi)}$ in the tree, which saves us an exponential blowup, making the procedure manageable). Note the *if* implication and the absence of the *only if*. The reason is that our simulation tree does not contain *all* sequences ρ of \mathcal{B} matching a given sequence π of \mathcal{A} (but if there is a matching sequence, at least one will be represented in the simulation tree).

To further illustrate simulation trees we represent in Fig. 3 a simple program, together with three observational transition systems generated from it. Their observation functions, are, from left to right: *the set of program counter values* in which the program may be at a given time; *the (single) program counter value* in which the program may be; and *the program counter value, together with the positiveness/negativeness of x* .

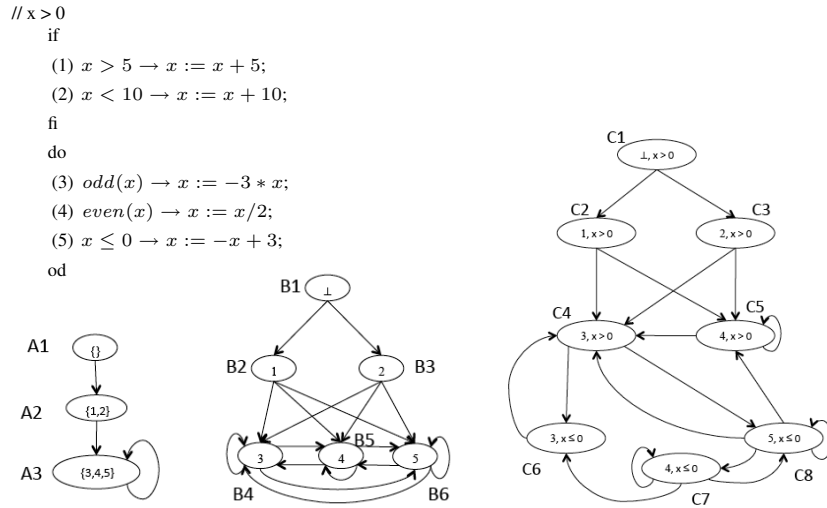


Figure 3. A nondeterministic program, and three observational transition systems.

We denote the three observational transitional system by $\mathcal{A}, \mathcal{B}, \mathcal{C}$ from left to right. Figure 4 depicts the simulation trees $\mathcal{T}_{\mathcal{A}, \mathcal{C}}$ and $\mathcal{T}_{\mathcal{B}, \mathcal{C}}$. The former detects a circularity, and illustrates the fact that \mathcal{C} observationally simulates \mathcal{A} . The latter detects a counterexample, and illustrates the fact that \mathcal{C} does not observationally simulate \mathcal{B} .

3. Conclusion, Related work, and Future Work

We present a formal framework and some initial solutions to the problem of checking semantical consistency between two operational specifications of a system at two different levels of abstraction. The framework involves a notion of observational simu-

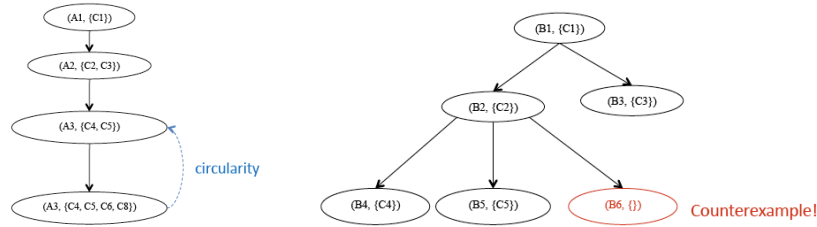


Figure 4. *Simulation trees: (left) \mathcal{C} simulates \mathcal{A} (right) \mathcal{C} does not simulate \mathcal{B} .*

lations between observational transition systems, and the definition of *simulation trees* that encode observational simulations between observational transition systems.

Due to space limitations we only cite here a few related works. Simulations in general are well-known in the area of formal verification; various forms of simulations are discussed in the book [CLA 99]. A general form of simulations, called *stuttering simulations*, is studied in an algebraic setting in [MAR 08]. Our observational simulations are a particular case of stuttering simulations, tailored to the model of observational transition systems. A richer model of observational transition systems (over an algebraic signature) is employed in [OGA 08], and inductive theorem proving is used to *check* whether a *given* relation between states of such systems is a simulation. Our approach is different: we automatically *compute* (currently, only up to a bounded number of steps) the simulations between two observational transition systems; this allows us to effectively *find* matching executions between two specifications.

In future work we shall use the *circular coinduction* proof technique implemented in the CIRC theorem prover [LUC 09] to compute observational simulations.

4. References

- [CLA 99] CLARKE E. ., GRUMBERG O., PELED D., *Model Checking*, MIT Press, 1999.
- [LUC 09] LUCANU D., GORIAC E.-I., CALTAIS G., ROSU G., “CIRC: A Behavioral Verification Tool Based on Circular Coinduction”, KURZ A., LENISA M., TARLECKI A., Eds., *CALCO*, vol. 5728 of *Lecture Notes in Computer Science*, Springer, 2009, p. 433-442.
- [MAR 08] MARTÍ-OLIET N., MESEGUER J., PALOMINO M., “Algebraic Stuttering Simulations”, *Electr. Notes Theor. Comput. Sci.*, vol. 206, 2008, p. 91-110.
- [OGA 08] OGATA K., FUTATSUGI K., “Simulation-based Verification for Invariant Properties in the OTS/CafeOBJ Method”, *Electr. Notes Theor. Comput. Sci.*, vol. 201, 2008, p. 127-154.