A Game-Theoretic Approach to Simulation of Data-Parameterized Systems

Orna Grumberg[1], Orna Kupferman[2], and Sarai Sheinvald[2]

[1] Department of Computer Science, The Technion, Haifa 32000, Israel
[2] School of Computer Science and Engineering, Hebrew University, Jerusalem 91904, Israel

Abstract. This work focuses on data-parameterized abstract systems that extend standard modelling by allowing atomic propositions to be parameterized by variables that range over some infinite domain. These variables may range over process ids, message numbers, etc. Thus, abstract systems enable simple modelling of infinite-state systems whose source of infinity is the data. We define and study a simulation pre-order between abstract systems. The definition extends the definition of standard simulation by referring also to variable assignments. We define VCTL* – an extension of CTL* by variables, which is capable of specifying properties of abstract systems. We show that VCTL* logically characterizes the simulation pre-order between abstract systems. That is, that satisfaction of VACTL*, namely the universal fragment of VCTL*, is preserved in simulating abstract systems. For the second direction, we show that if an abstract system $\mathcal{A}_2$ does not simulate an abstract system $\mathcal{A}_1$, then there exists a VACTL formula that distinguishes $\mathcal{A}_1$ from $\mathcal{A}_2$. Finally, we present a game-theoretic approach to simulation of abstract systems and show that the prover wins the game iff $\mathcal{A}_2$ simulates $\mathcal{A}_1$. Further, if $\mathcal{A}_2$ does not simulate $\mathcal{A}_1$, then the refuter wins the game and his winning strategy corresponds to a VACTL formula that distinguishes $\mathcal{A}_1$ from $\mathcal{A}_2$. Thus, the many appealing practical advantages of simulation are lifted to the setting of data-parameterized abstract systems.

1 Introduction

In system verification, we check that an implementation satisfies its specification. Both the implementation and the specification describe the possible behaviors of the system at different levels of abstraction. If we represent the implementation $\mathcal{I}$ and the specification $\mathcal{S}$ using Kripke structures, then the formal relation that captures satisfaction in the linear approach is *trace containment*: $\mathcal{S}$ trace-contains $\mathcal{I}$ iff it is possible to generate by $\mathcal{S}$ every (finite and infinite) sequence of observations that can be generated by $\mathcal{I}$. The notion of trace containment is logically characterized by linear temporal logics such as LTL in the sense that $\mathcal{S}$ trace-contains $\mathcal{I}$ iff every LTL formula that holds in $\mathcal{S}$ holds also in $\mathcal{I}$. Unfortunately, it is difficult to check trace containment (complete for PSPACE [29]). The formal relation that captures satisfaction in the branching approach is *tree containment*: $\mathcal{S}$ tree-contains $\mathcal{I}$ iff it is possible to embed in the unrolling of $\mathcal{S}$ every (finite and infinite) tree of observations that can be embedded in the unrolling of $\mathcal{I}$. The notion of tree containment is equivalent to the notion of simulation, as defined by Milner [24]: $\mathcal{S}$ tree-contains $\mathcal{I}$ iff $\mathcal{S}$ simulates $\mathcal{I}$; that is, we can relate each state of

$\mathcal{I}$ to a state of $\mathcal{S}$ so that two related states i and s agree on their observations and every successor of i is related to some successor of s [26].

Simulation has several theoretical and practical appealing properties. First, like trace containment, simulation is robust: for universal branching temporal logics (where only universal path quantification is allowed) such as ACTL and ACTL* (the universal fragments of the computation tree logics CTL and CTL*), we have that $\mathcal{S}$ simulates $\mathcal{I}$ iff every formula that holds in $\mathcal{S}$ holds also in $\mathcal{I}$ [2, 15]. Second, unlike trace containment, the definition of simulation is local, as the relation between two states is based only on their successor states. As a result, simulation can be checked in polynomial time [7, 1]. The locality advantage is so compelling as to make simulation useful also to researchers that favor trace-based specification: in automatic verification, simulation is widely used as an efficiently computable sufficient condition for trace containment [19]; in manual verification, trace containment is most naturally proved by exhibiting local witnesses such as simulation relations or refinement mappings (a restricted form of simulation relations) [20, 22, 23, 8].

In addition to the use of simulation in *step-wise refinement*, where a pre-order between an implementation and its specification is checked, simulation is helpful in coping with the state-explosion problem, as it enables the verification process to proceed with respect to an over-approximation of the implementation: instead of verifying $\mathcal{I}$, we abstract some of its details and generate a (typically much smaller) system $\mathcal{I}'$ that simulates $\mathcal{I}$. Verification then proceeds with respect to $\mathcal{I}'$, either proving that it satisfies the specification (in which case we can conclude that so does $\mathcal{I}$) or not, in which case the abstraction is refined [6].

Abstraction and simulation have turned out to be key methods in coping with the state-explosion problem. In particular, by abstracting elements of the system that have an infinite domain, one can verify infinite-state systems. Often, the source of infinity in a system is data that range over an unbounded or infinite domain, such as content of messages, process ids, etc. Traditional abstraction methods either hide the data or abstract its value by finite-domain predicates [27]. In [12, 13], we introduced *abstract systems*, which finitely and naturally represent infinite-state systems in which the source of infinity is data that range over an infinite domain. Formally, an abstract system is a finite Kripke structure whose atomic propositions are parameterized by variables ranging over the infinite domain. A transition of the system may reset a subset of the variables, freeing them of their previous assignment. The different concrete computation trees, or *concretizations*, of an abstract system are obtained by legally assigning concrete domain values to the variables along an unwinding of the abstract system.

For example, consider the system presented in Figure 1. It represents a simple communication protocol, where the variable x represents the message id. The concretization presented in Figure 1 is obtained by assigning values to x in a way that agrees with the resets in the protocol: when the transition from the $timeout$ state is taken, the message is resent with the same message id. When the transition from the ack state is taken, x is reset and may be reassigned, as reflected in the concretization.

Evidently, abstract systems are capable of describing communication protocols with unboundedly many processes, systems with interleaved transactions each carrying a unique id, buffers of messages with an infinite domain, and many more.

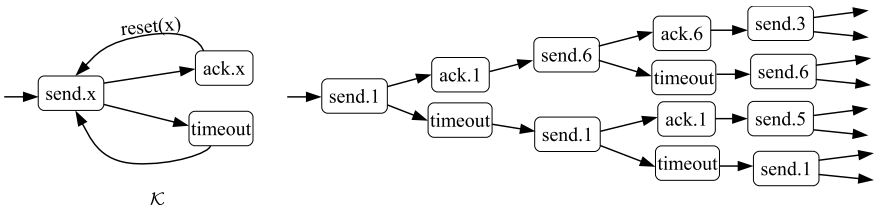

Fig. 1. A simple communication protocol and a possible concretization.

In this paper we combine the clean treatment of data over infinite domain with the theoretical and practical advantages of simulation. We define simulation between abstract systems, study its properties, and describe a logical characterization for it.

The challenge of specifying behaviors with an infinite component has led to the development of various formalisms. One class of formalisms consists of variants of automata over infinite alphabets. Types of such automata include *register automata* [28], which have a finite set of registers, each of which may contain a letter from the infinite alphabet. Register automata have been extensively studied and may include features like alternation, two-wayness, a nondeterministic change of the content of registers, and further generalizations [25, 18, 13]. *Pebble automata* [25, 30] place pebbles on the input word in a stack-like manner, and in *data automata* [4, 3], the infinite alphabet consists of pairs – a letter from a finite alphabet and a data value from some infinite domain. The finite alphabet is accessed directly and the data is accessed indirectly via the equivalence relation it induces on the set of positions. Finally, closest to our abstract systems are nondeterministic finite automata with variables [11].

The second class of formalisms consists of extensions of temporal logics. An extension in which atomic propositions are parameterized by a variable that ranges over the set of processes ids was studied in [5, 10]. These works are tailored for the setting of parameterized systems, and are also restricted to systems in which (almost) all components are identical. In Constraint LTL [9], atomic propositions may be constraints like $x < y$, and formulas are interpreted over sequences of assignments of variables to values in $\mathbb{N}$ or $\mathbb{Z}$. Unlike our approach, the focus is on reasoning about sequences of numerical values. In [21, 16], LTL and CTL have been extended with a freeze quantifier, which is used for storing values from an infinite domain in a register.

In [12], we introduced *variable* LTL (VLTL), a first-order extension of LTL. Like abstract systems, VLTL uses atomic propositions parameterized with variables to describe the behaviors of computations that may carry infinitely many values. For example, the VLTL formula $\psi = \forall x; \mathrm{G}(send.x \rightarrow \mathrm{F} receive.x)$ states that for every value d in the domain, whenever a message with content d is sent, then a message with content d is eventually received. In this work, we define the logic VCTL*, a first-order extension of CTL*. Similarly to CTL*, the logic VCTL* has existential and universal path quantification. Similarly to VLTL, it also has existential and universal quantification over variables. While VLTL is interpreted over infinite computations of abstract systems, VCTL* is interpreted over their computation trees.

As an example, consider a variable x that ranges over process ids. The VCTL* formula $\varphi_1 = \exists x; \mathrm{AFAG}\neg idle.x$ states that there exists a process that is eventually not idle along all paths. The formula $\varphi_2 = \mathrm{AF}\exists x; \mathrm{AG}\neg idle.x$ states that every path eventually reaches a point where there exists a process that is never idle from that point on along all paths. The formula $\varphi_3 = \mathrm{AFA}\exists x; \mathrm{G}\neg idle.x$ states that along every path, from some point on, there exists a process that is never idle. Finally, $\varphi_4 = \mathrm{AFAG}\exists x; \neg idle.x$ states that from some point on, in every step of every path, there exists a process that is never idle. Note that the formulas are not equivalent, and they demonstrate the power of the ability to nest the variable quantifiers within the formula.[3] In particular, in φ_3 (and not in φ_2) different paths may have different non-idle processes, and in φ_4 (and not in φ_3), this may be a different process at each step.

Our goal is to define a simulation relation for abstract systems in a way that preserves their behaviors. Preserving the behavior of the variables raises some challenges. Consider two states q_1 and q_2 of abstract systems $\mathcal{A}_1$ and $\mathcal{A}_2$. In standard simulation, q_1 and q_2 may be matched only if they agree on the labeling of the atomic propositions. In the abstract systems setting, we need, in addition, to make sure that the variables that parameterize these atomic propositions can be matched. For example, if q_1 is labeled by $a.x_1$ and q_2 is labeled by $a.x_2$, where a is an atomic proposition and x_1, x_2 are variables, then a simulation that matches q_1 with q_2 must make sure that every value that is assigned to x_1 can also be assigned to x_2. This latter condition is no longer local, as the value assigned to an occurrence of x_2 depends both on the history of the behavior before reaching q_2, and on the behavior of other states in which x_2 occures. Thus, a simulation relation between $\mathcal{A}_1$ and $\mathcal{A}_2$ must keep a memory component for the variables, and the challenge is to keep the definition of simulation as local as possible, with the global elements being restricted to the variables only.

We manage tracking the behavior of the variables locally by adding a function that maps the variables of $\mathcal{A}_2$ to the variables of $\mathcal{A}_1$ to each tuple in the simulation relation. Thus, our simulation relation consists of triplets: a state of $\mathcal{A}_1$, a state of $\mathcal{A}_2$, and a function over the variables. We present an algorithm for computing a maximal simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$. The function component may create a simulation relation of size that is exponential in the number of variables. While this is worse than the polynomial size of standard simulation, one should bear in mind that trace containment for abstract systems is undecidable, as opposed to the PSPACE complexity of standard trace containment. We argue that our definition is indeed robust with respect to VACTL* – the universal fragment of VCTL*. First, if $\mathcal{A}_2$ simulates $\mathcal{A}_1$, denoted $\mathcal{A}_1 \preceq \mathcal{A}_2$, then every VACTL* formula that is satisfied in $\mathcal{A}_2$ is also satisfied in $\mathcal{A}_1$. The second direction is much harder, and we show that if $\mathcal{A}_1 \not\preceq \mathcal{A}_2$, then we can construct a VACTL formula that $\mathcal{A}_2$ satisfies, and $\mathcal{A}_1$ does not.

A *simulation game* for (non-abstract) systems A_1 and A_2 is played between two players: a *prover*, who wishes to prove that A_2 simulates A_1, and a *refuter*, who wishes to prove the contrary. In each round of the game, the refuter advances along A_1 and the prover advances along A_2, aiming to match every move of the refuter by moving to a state with the same label. A simulation relation from A_1 to A_2 induces a winning strategy for the prover. Also, if A_2 does not simulate A_1, then a winning strategy for

[3] In VLTL, variable quantification is restricted to appear only at the head of the formula.

the refuter exists, and it induces an ACTL formula that holds only in A_2. We define a game-theoretic approach to simulation between abstract systems $\mathcal{A}_1$ and $\mathcal{A}_2$. As in the standard setting, the game proceeds in rounds along both abstract systems, where in every step the prover attempts to match the state it chooses in $\mathcal{A}_2$ with the state in $\mathcal{A}_1$ chosen by the refuter. Here, however, matching refers also to the variables, and the prover must make sure that the variables in the states it chooses can be assigned the same values that can be assigned to the variables in the states that the refuter chooses. As explained above, this property is not local, making the game more sophisticated: While in the traditional setting the game proceeds along a single path in each system, in our setting the refuter may split the game to continue temporarily along two different paths – those along which he plans to force the prover to use inconsistent assignments.

As in the traditional setting, a simulation relation induces a winning strategy for the prover. Also, if $\mathcal{A}_2$ does not simulate $\mathcal{A}_1$, then a winning strategy is induced by a VACTL formula that holds only in $\mathcal{A}_2$. The need to refer to the variables makes the game and its correctness proof complicated. In particular, in case the refuter wins thanks to the inability of the prover to correctly handle the variables, the distinguishing formula captures this by requirements that refer to variable assignments. The construction of the formula relies on our algorithm for computing a simulation relation from $\mathcal{A}_1$ to $\mathcal{A}_2$. The time complexity of our algorithm is exponential in the number of variables, and accordingly, so is the depth of the formula. Still, the appealing properties of the game in the traditional setting are preserved, in particular the fact that the players have memoryless strategies.

In Section 7 we discuss directions for future research and the practical applications of defining the simulation pre-order in the setting of abstract systems.

2 Preliminaries

Kripke Structures and Simulation Let AP be a finite set of atomic propositions. A *Kripke structure* is a tuple $A = \langle Q, q^0, AP, R, L\rangle$, where Q is a finite set of states, q^0 is an initial state, $R \subseteq Q \times Q$ is a total transition relation, and $L : Q \to 2^{AP}$ maps each state to the set of atomic propositions that hold in the state. We sometimes refer to Kripke structures as systems.

Let $A_1 = \langle Q_1, q_1^0, AP, R_1, L_1\rangle$ and $A_2 = \langle Q_2, q_2^0, AP, R_2, L_2\rangle$ be two Kripke structures over the same set of atomic propositions. A *simulation* [24] from A_1 to A_2 is a relation $H \subseteq Q_1 \times Q_2$ such that for every $\langle q_1, q_2\rangle \in H$, we have that $L_1(q_1) = L_2(q_2)$, and for every $\langle q_1, q_1'\rangle \in R_1$ there exists $\langle q_2, q_2'\rangle \in R_2$ such that $\langle q_1', q_2'\rangle \in H$. If $\langle q_1^0, q_2^0\rangle \in H$, then we say that A_2 *simulates* A_1, denoted $A_1 \preceq A_2$.

It is well known [14] that if $A_1 \preceq A_2$, then for every $\varphi \in \text{ACTL}^\star$, if $A_2 \models \varphi$ then $A_1 \models \varphi$. If $A_1 \not\preceq A_2$, then there exists an ACTL formula φ such that $A_2 \models \varphi$ and $A_1 \not\models \varphi$. We call such a formula a *distinguishing formula for* A_1 *and* A_2.

A *simulation game* [17] for A_1 and A_2 is a protocol for two players: a prover $\mathcal{P}$, who wishes to prove that $\mathcal{A}_1 \preceq \mathcal{A}_2$, and a refuter $\mathcal{R}$, who wishes to prove that $\mathcal{A}_1 \not\preceq \mathcal{A}_2$. A position in the game is a pair $\langle q_1, q_2\rangle \in Q_1 \times Q_2$, where q_1 is the location of $\mathcal{R}$ and q_2 is the location of $\mathcal{P}$. A play in the game starts in $\langle q_1^0, q_2^0\rangle$. If $L_1(q_0^1) \neq L_2(q_0^2)$, then $\mathcal{R}$ wins. Otherwise, the play continues as follows. Let $\langle q_1, q_2\rangle$ be the current position.

In each round, $\mathcal{R}$ chooses a transition $\langle q_1, q_1' \rangle \in R_1$, and $\mathcal{P}$ responds by choosing a transition $\langle q_2, q_2' \rangle \in R_2$, such that $L_1(q_1') = L_2(q_2')$. The play then continues from position $\langle q_1', q_2' \rangle$. Hence, a play in the game induces two paths, one in A_1 and one in A_2. If at some point in the play, $\mathcal{P}$ is unable to respond with a suitable move, then $\mathcal{R}$ wins. Otherwise, the play continues forever and $\mathcal{P}$ wins.

It holds that $A_1 \preceq A_2$ iff $\mathcal{P}$ has a winning memoryless strategy in the game. Equivalently, $A_1 \not\preceq A_2$ iff $\mathcal{R}$ has a winning strategy. The latter corresponds to a distinguishing formula for A_1 and A_2.

Abstract Systems An *abstract system* is a tuple $\mathcal{A} = \langle Q, q^0, AP, X, R, L, S \rangle$, where Q, q^0, and R are as in Kripke structures, and

- X is a set of variables.
- AP is a set of atomic propositions that may be parameterized by variables from X.
- $L : Q \to 2^{AP \cup (AP \times X)}$ maps each state to the set of atomic propositions that hold in the state. We require that for every $q \in Q$ and $a \in AP$, the set $L(q)$ contains at most one occurrence of a.
- $S : R \to 2^X$ maps each transition to the set of variables that are reset along the transition.

A *concretization* of $\mathcal{A}$, which is a concrete computation tree of $\mathcal{A}$, is obtained by assigning values to the variables, as we explain next.

A Σ-labeled tree is a pair $\langle T, l \rangle$ where $T = \langle V, E \rangle$ is a directed tree with a set of nodes V and a set of edges E, and $l : V \to \Sigma$ is a function that labels each node in the tree by a letter from Σ.

Let $\langle T, l \rangle$ be the $2^{AP \cup (AP \times X)}$-labeled tree obtained by unwinding $\mathcal{A}$ from q^0. Formally, $T = \langle V, E \rangle$ is such that $V \subseteq Q^*$, where $q^0 \in V$ is the root of T, and $w \cdot q \cdot q'$ is in V, for $w \in Q^\star$ and $q, q' \in Q$, iff $w \cdot q \in V$ and $\langle q, q' \rangle \in R$, in which case $\langle w \cdot q, w \cdot q \cdot q' \rangle \in E$. Also, $l(w \cdot q) = L(q)$. Note that we can associate with each edge in the tree the set of variables that are reset along it, namely these reset in the transition in $\mathcal{A}$ that induces the edge.

Let $\mathcal{D}$ be an infinite domain. A *$\mathcal{D}$-concretization of $\mathcal{A}$* is an infinite $2^{AP \cup (AP \times \mathcal{D})}$-labeled tree $\langle T, l_f \rangle$ obtained from $\langle T, l \rangle$ by assigning values from $\mathcal{D}$ to the occurrences of the variables in every node in T in a way that agrees with the resets along the transitions of $\mathcal{A}$. That is, for every node s of T we assign a function $f_s : X \to \mathcal{D}$. The labeling $l_f(s)$ of s is obtained by replacing every variable x in $l(s)$ by $f_s(x)$. The functions f_s satisfy the following property. Let s be a node in T, let x be a variable, and let s' be a node in the subtree rooted in s. If x is not reset along the path from s to s', then $f_s(x) = f_{s'}(x)$. That is, all the occurrences of x in a subtree of T with root s that are not reset along the path in the abstract system that leads from s to them, are assigned the same $d \in \mathcal{D}$ in $\langle T, l_f \rangle$. Note that the tree $\langle T, l_f \rangle$ is in fact an infinite state system over $AP \cup (AP \times \mathcal{D})$.

3 VCTL*

In this section, we define *variable* CTL* (VCTL*, for short) – a first order extension of CTL* that handles infinite data. Like abstract systems, VCTL* uses atomic proposi-

tions that are parameterized with variables. In addition, VCTL* uses quantifiers over the variables that may be nested within the formula.

The syntax of VCTL* includes formulas of two types: state formulas and path formulas. Let X be a set of variables, and let AP be a set of atomic propositions that may be parameterized by variables from X. A VCTL* state formula is $p \in AP$, $a.x \in AP \times X$, $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$, $\exists x; \varphi_1$, or $\mathrm{E}\psi_1$, for VCTL* state formulas φ_1 and φ_2 and a VCTL* path formula ψ_1. A VCTL* path formula is φ_1, $\neg\psi_1$, $\psi_1 \vee \psi_2$, $\mathrm{X}\psi_1$, $\psi_1 \mathrm{U} \psi_2$, or $\exists x; \psi_1$, for a VCTL* state formula φ_1 and VCTL* path formulas ψ_1 and ψ_2. Finally, VCTL* is the set of all closed VCTL* state formulas.

We turn to define the semantics of VCTL*. Let $\mathcal{A}$ be an abstract system with a labeling function L, let φ be a VCTL* formula. We say that $\mathcal{A}$ satisfies φ (denoted $\mathcal{A} \models \varphi$) if for every infinite domain $\mathcal{D}$, every $\mathcal{D}$-concretization of $\mathcal{A}$ satisfies φ. Thus, it is left to define the semantics of VCTL* with respect to concretizations of abstract systems. As usual with first order logic, since we define the semantics inductively over open formulas, we define the semantics also w.r.t. an assignment $t : X \to \mathcal{D}$ over the variables. For closed formulas, the semantics is independent of an assignment to the variables.

Let $\langle T, I_f \rangle$ be a $\mathcal{D}$-concretization of $\mathcal{A}$, let s be a node in T, and let $\pi = s_0, s_1, s_2, \ldots$ be an infinite path in T. We assume that $\langle T, I_f \rangle$ is fixed and use $s \models_t \varphi$ and $\pi \models_t \psi$ to indicate that s satisfies φ under the assignment t, and similarly for π and ψ. The relation $\models_t$ is defined inductively as follows. Consider a state formula φ.

- If $\varphi \in AP$ or the outermost operator in φ is $\neg$, $\vee$, or E, then the definition is as in CTL*.
- If $\varphi = a.x \in AP \times X$, then $s \models_t \varphi$ iff $a.d \in l(s)$ and $t(x) = d$.
- If $\varphi = \exists x; \varphi_1$, then $s \models_t \varphi$ iff there exists $d \in \mathcal{D}$ such that $s \models_{t[x \leftarrow d]} \varphi_1$.

Consider a path formula ψ.

- If ψ is a state formula or the outermost operator in ψ is $\neg$, X, or U, then the definition is as in CTL*.
- If $\psi = \exists x; \psi_1$, then $\pi, i \models_t \psi$ iff there exists $d \in \mathcal{D}$ such that $\pi, i \models_{t[x \leftarrow d]} \psi$.

We use the usual abbreviations G ("always") and F ("eventually") for temporal operators, the path quantifier A ("for all paths"), and the $\forall$ quantifier over variables.

The logic VACTL* (*universal* VACTL*) is the fragment of VCTL* in which negation is restricted to atomic propositions and only the A path quantifier is allowed. The logic VACTL is the fragment of VACTL* in which temporal operators cannot be nested without a path quantifier between them. These fragments join the previously defined fragment VLTL [12], which is the set of all VCTL* formulas of the form $\mathrm{A}\psi$, where ψ is a path formula that does not contain path quantifiers and all its variable quantifiers are at the head of the formula.

Remark 1. It is possible to augment the definition of abstract systems and VCTL* formulas to include inequalities over the set of variables, say $x_1 \neq x_2$. This restricts the set of legal concretizations and possible assignments, respectively. It is easy to extend our results to a setting with such an augmentation.

4 Simulation of Abstract Systems

In this section we define a simulation pre-order between abstract systems. Let $\mathcal{A}_1 = \langle Q_1, q_1^0, AP, X_1, R_1, L_1, S_1\rangle$ and $\mathcal{A}_2 = \langle Q_2, q_2^0, AP, X_2, R_2, L_2, S_2\rangle$ be abstract systems over the same set of atomic propositions.

We want to define simulation in such a way that if $\mathcal{A}_2$ simulates $\mathcal{A}_1$, then every behavior of $\mathcal{A}_1$ is exhibited in $\mathcal{A}_2$. In standard simulation, state q_1 of system A_1 may be matched with state q_2 of system A_2 only if q_1 and q_2 are equally labeled. In abstract systems, we need, in addition, to assure a match in the variables parameterizing the atomic propositions in q_1 and q_2. If, for example, $L_1(q_1)$ contains $a.x_1$ and $L_2(q_2)$ contains $a.x_2$, for $a \in AP$, $x_1 \in X_1$, and $x_2 \in X_2$, we want to match q_1 with q_2 only if every value that can be assigned to x_1 can also be assigned to x_2. Accordingly, a simulation relation H also indicates that when matching q_1 with q_2, the variable x_2 in q_2 is matched with the variable x_1 in q_1.

Formally, a simulation relation H from $\mathcal{A}_1$ to $\mathcal{A}_2$ consists of triplets of type $\langle q_1, q_2, f\rangle$, where q_1 is a state of $\mathcal{A}_1$, q_2 is a state of $\mathcal{A}_2$, and $f : X_2 \to X_1 \cup \{\top, \bot\}$ is a function that maps the variables that parameterize atomic propositions that hold in q_2 to the variables that parameterize atomic propositions that hold in q_1, and also contains information about previous and future matches. We elaborate, and explain the role of $\top$ and $\bot$, below.

We impose some restrictions on f that ensure that it matches the variables in a suitable way. We say that f is a *match for* q_1 *and* q_2 if $L_1(q_1)$ is equal to the set obtained from $L_2(q_2)$ by replacing every x_2 in q_2 with $f(x_2)$. Note that in order for f to be a match for q_1 and q_2, for every variable $x_2 \in X_2$ that appear in q_2 it must hold that $f(x_2) \notin \{\top, \bot\}$. We require that if $\langle q_1, q_2, f\rangle \in H$, then f is a match for q_1 and q_2. That is, f must correctly match the variables locally.

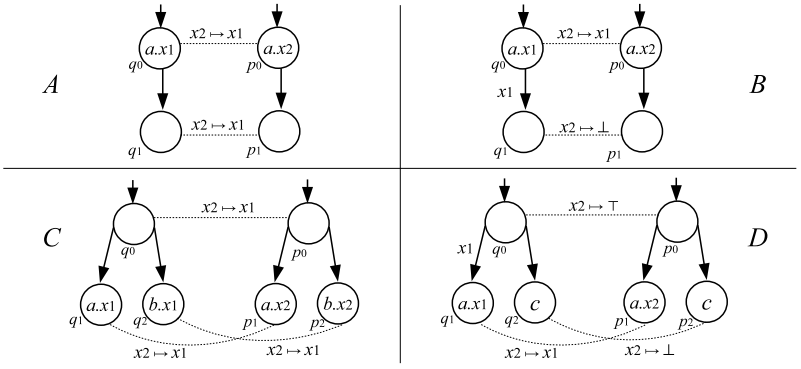

Fig. 2. Cases A, B, C and D.

However, locally matching the variables is not enough. Consider Case A presented in Figure 2. The local match for q_0 and p_0 is $x_2 \mapsto x_1$. The variables x_1 and x_2 are not reset in the transitions $\langle q_0, q_1\rangle$ and $\langle p_0, p_1\rangle$. Then H remembers that the value of x_2 is bound to the value of x_1, by setting $x_2 \mapsto x_1$ also when matching q_1 with p_1.

Next, consider Case B. Again, the local match for q_0 and p_0 is $x_2 \mapsto x_1$. Since x_1 is reset in $\langle q_0, q_1 \rangle$ and x_2 is not reset in $\langle p_0, p_1 \rangle$, then x_2 cannot be matched with a variable in p_1, since it is still bound to the value x_1 was assigned. Then H remembers this when matching q_1 with p_1 by setting $x_2 \mapsto \bot$.

Now, consider Case C. Since x_2 is not reset along both $\langle p_0, p_1 \rangle$ and $\langle p_0, p_2 \rangle$, it must be mapped to the same variable both when matching q_1 with p_1, and when matching q_2 with p_2. The simulation sets $x_2 \mapsto x_1$ already when matching q_0 with p_0, even though x_1, x_2 do not appear in these states, and remembers this assignment when moving from p_0 to its two different subtrees. This forces both occurences of x_2 to be matched with the same (unreset) variable x_1.

Finally, consider Case D. The simulation must match x_2 with x_1 when matching q_1 with p_1. Unlike Case C, where both occurences of x_2 must be equally matched, here there is a single occurence of x_2, which can therefore be freely matched. x_1 is reset in $\langle q_0, q_1 \rangle$, and so x_2 cannot be mapped to x_1 when matching q_0 with p_0 (unlike Case C). On the other hand, x_2 is not reset in $\langle p_0, p_1 \rangle$, and so H must match x_2 with a non-$\bot$ value when matching q_0 with p_0. H solves this by allowing x_2 to remain "uncommitted" when matching q_0 with p_0, by setting $x_2 \mapsto \top$. To ensure that x_2 occurs only once, H sets $x_2 \mapsto \bot$ when matching q_2 with p_2, which means that x_2 cannot occur along the subtree from p_2, unless it is reset.

We now formalize these ideas. Let f be a match for q_1 and q_2, and let $\langle q_1, q_1' \rangle \in R_1$ and $\langle q_2, q_2' \rangle \in R_2$. We say that a function $f' : X_2 \to X_1 \cup \{\top, \bot\}$ is *consistent with f w.r.t $\langle q_1, q_1' \rangle$ and $\langle q_2, q_2' \rangle$* if f' is a match for q_1' and q_2', and

- If $f(x_2) \in X_1 \cup \{\bot\}$ and $x_2 \notin S_2(\langle q_2, q_2' \rangle)$ and $f(x_2) \notin S_1(\langle q_1, q_1' \rangle)$, then $f'(x_2) = f(x_2)$. In particular, since $\bot$ cannot be reset, if $f(x_2) = \bot$ and $x_2 \notin S_2(\langle q_2, q_2' \rangle)$, then also $f'(x_2) = \bot$. That is, if $x_2 \mapsto \bot$, then x_2 must be reset before it may be matched with a variable.
- If $f(x_2) = x_1$ for $x_1 \in X_1$ and $x_2 \notin S_2(\langle q_2, q_2' \rangle)$ and $x_1 \in S_1(\langle q_1, q_1' \rangle)$ then $f'(x_2) = \bot$. That is, if x_1 has been reset, then so must x_2 before it appears again.

Let F be the set of functions from X_2 to $X_1 \cup \{\top, \bot\}$. We say that $H \subseteq (Q_1 \times Q_2 \times F)$ is a *simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$* if for every $\langle q_1, q_2, f \rangle \in H$, the following holds. Let $\langle q_1, q_1^1 \rangle, \langle q_1, q_1^2 \rangle, \ldots \langle q_1, q_1^k \rangle$ be the transitions from q_1. Then there exist transitions $\langle q_2, q_2^1 \rangle, \langle q_2, q_2^2 \rangle, \ldots \langle q_2, q_2^k \rangle$ from q_2 and functions $f^1, f^2, \ldots f^k$ such that the following *simulation conditions* hold.

1. f^i is consistent with f w.r.t $\langle q_1, q_1^i \rangle$ and $\langle q_2, q_2^i \rangle$ for every $1 \leq i \leq k$.
2. $\langle q_1^1, q_2^1, f^1 \rangle, \langle q_1^2, q_2^2, f^2 \rangle, \ldots \langle q_1^k, q_2^k, f^k \rangle \in H$.
3. For every $x_2 \in X_2$, if $f(x_2) = \top$, then there exists at most one $1 \leq i \leq k$ such that $x_2 \notin S_2(\langle q_2, q_2^i \rangle)$ and $f^i(x_2) \neq \bot$.

If there exists a function f^0 that is a match for q_1^0 and q_2^0 such that $\langle q_1^0, q_2^0, f^0 \rangle \in H$, then we say that $\mathcal{A}_2$ *simulates* $\mathcal{A}_1$, and denote $\mathcal{A}_1 \preceq \mathcal{A}_2$.

Example 1. Consider the abstract systems $\mathcal{A}_1$ and $\mathcal{A}_2$ presented in Figure 3. It holds that $\mathcal{A}_1 \preceq \mathcal{A}_2$ by a simulation relation $\{\langle q_0, p_0, x \mapsto x_1 \rangle, \langle q_1, p_1, x \mapsto x_2 \rangle, \langle q_2, p_0, x \mapsto x_2 \rangle, \langle q_3, p_1, x \mapsto x_1 \rangle\}$. Notice that $\mathcal{A}_2$ uses not only fewer states, but also fewer variables.

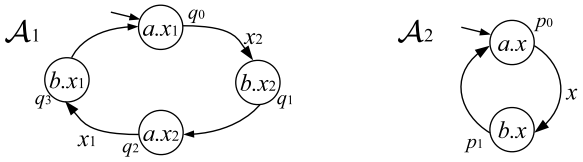

Fig. 3. The abstract systems $\mathcal{A}_1$ and $\mathcal{A}_2$.

Computing a Simulation We present an algorithm for computing a simulation relation from $\mathcal{A}_1$ to $\mathcal{A}_2$. The algorithm starts with the set of tuples that can be locally matched. Then, in every iteration, the tuples that violate one of the simulation conditions are omitted. The algorithm terminates when a fixed point is reached.

Let $H \subseteq Q_1 \times Q_2 \times F$ be a relation. Let $\langle q_1, q_2, f\rangle \in H$, and let $\langle q_1, q_1^1\rangle, \langle q_1, q_1^2\rangle, \ldots$ $\langle q_1, q_1^k\rangle$ be the transitions from q_1. We say that $\langle q_1, q_2, f\rangle$ is *good w.r.t.* H if there exist transitions $\langle q_2, q_2^1\rangle, \langle q_2, q_2^2\rangle, \ldots \langle q_2, q_2^k\rangle$ from q_2 and functions $f^1, f^2, \ldots f^k$ that meet the simulation conditions w.r.t. H.

We define a sequence of relations $H_0, H_1, \ldots$ as follows. First, H_0 is the set of all tuples $\langle q_1, q_2, f\rangle$ such that $q_1 \in Q_1$, $q_2 \in Q_2$, and f is a match for q_1, q_2. Then, for every $i > 0$, we define $H_i = \{\langle q_1, q_2, f\rangle | \langle q_1, q_2, f\rangle$ is good w.r.t. $H_{i-1}\}$. Notice that $H_0 \supseteq H_1 \supseteq H_2 \supseteq \ldots$, and since H_0 is finite, after finitely many i's a fixed point H^* is reached. Notice that H^* is a simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$. In fact, H^* is a maximal simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$. Indeed, every simulation G satisfies $G \subseteq H_0$, and it is easy to show by induction that for every i, the relation G is contained in H_i.

The complexity of this algorithm is exponential in the number of variables. This is not surprising, as there are pairs of abstract systems for which the size of a simulation relation is exponential in the number of variables (recall that $|F| = \mathcal{O}(|X_1|^{|X_2|})$).

5 A Logical Characterization of Simulation

In this section, we show that VACTL* logically characterizes the simulation pre-order for abstract systems. Formally, we prove the following.

Lemma 1. *If $\mathcal{A}_1 \preceq \mathcal{A}_2$, then for every $\varphi \in$ VACTL*, if $\mathcal{A}_2 \models \varphi$, then $\mathcal{A}_1 \models \varphi$.*

Lemma 1 follows from the following two properties. First, for an infinite domain $\mathcal{D}$, for every $\mathcal{D}$-concretization A_1 of $\mathcal{A}_1$ there exists a $\mathcal{D}$-concretization A_2 of $\mathcal{A}_2$, such that $A_1 \preceq A_2$ (by standard simulation). Second, standard simulation preserves VACTL*. Lemma 1 then follows from the semantics of VACTL* w.r.t. abstract systems.

Secondly, we prove that if $\mathcal{A}_1 \not\preceq \mathcal{A}_2$, then there exists a *distinguishing* VACTL *formula for $\mathcal{A}_1$ and $\mathcal{A}_2$*: a formula that $\mathcal{A}_2$ satisfies, and $\mathcal{A}_1$ does not satisfy.

Theorem 1. *If $\mathcal{A}_1 \not\preceq \mathcal{A}_2$, then there exists $\varphi \in$ VACTL, such that $\mathcal{A}_2 \models \varphi$ and $\mathcal{A}_1 \not\models \varphi$.*

The proof of Theorem 1 is constructive: φ is induced by the algorithm for computing a simulation in Section 4. For every tuple $\langle q_1, q_2, f\rangle$ that is removed from H_i

for some i, we construct a *semi-distinguishing formula*, which, roughly speaking, is an open VACTL formula that distinguishes between q_1 and q_2 under the assumption that f matches q_1 with q_2. As in the the standard setting, the distinguishing VACTL formula uses only the AX operator, but also uses quantifiers over the variables that refer to variable assignments.

Combining Lemma 1 and Theorem 1, we have the following.

Theorem 2. *$\mathcal{A}_1 \preceq \mathcal{A}_2$ iff for every $\varphi \in$ VACTL*, it holds that if $\mathcal{A}_2 \models \varphi$, then $\mathcal{A}_1 \models \varphi$.*

Thus, VACTL* offers a characterization of simulation of abstract systems, precisely as ACTL* offers a characterization of simulation of Kripke structures.

Example 2. Consider the abstract systems $\mathcal{A}_1$ and $\mathcal{A}_2$ presented in Figure 4. The relation H_0 is $\{\langle q_0, p_0, x_2 \mapsto x_1\rangle$,$\{\langle q_1, p_1, x_2 \mapsto x_1\rangle$, $\langle q_1, p_1, x_2 \mapsto \bot\rangle$,$\langle q_1, p_1, x_2 \mapsto \top\rangle$, $\langle q_2, p_2, x_2 \mapsto x_1\rangle\}$. When calculating H_1, the tuple $\langle q_1, p_1, x_2 \mapsto \bot\rangle$ is removed as it violates condition (2) of the simulation conditions. A semi-distinguishing formula $\mathrm{AX}b.x_2$ for $\langle q_1, p_1, x_2 \mapsto \bot\rangle$ is calculated by the inconsistency of $x_2 \mapsto x_1$ in $\langle q_2, p_2, x_2 \mapsto x_1\rangle$ with $x_2 \mapsto \bot$. In H_2, the tuple $\langle q_0, p_0, x_2 \mapsto x_1\rangle$ violates condition (2), and a semi-distinguishing formula for it, which also distinguishes $\mathcal{A}_1$ from $\mathcal{A}_2$, is $\varphi = \exists x_2; a.x_2 \wedge \mathrm{AX}(\mathrm{AX}b.x_2)$. Indeed, $\mathcal{A}_2 \models \varphi$, whereas $\mathcal{A}_1 \not\models \varphi$.

Next, consider the abstract systems $\mathcal{B}_1$ and $\mathcal{B}_2$. The states q_0 and p_0 appear in H_0 with the functions $x_2 \mapsto x_1, x_2 \mapsto x'_1, x_2 \mapsto \bot, x_2 \mapsto \top$. The function $x_2 \mapsto x_1$ cannot match q_2 with p_2, and this contradiction ultimately creates the semi-distinguishing formula $\exists x_2;\ \mathrm{AX}(a.x_2 \vee \mathrm{AX}b.x_2)$ for $\langle q_0, p_0, x_2 \mapsto x_1\rangle$. Similarly, the functions $x_2 \mapsto x'_1$ and $x_2 \mapsto \bot$ contradict matching q_1 with p_1 and q_2 with p_2, respectively, ultimately creating the semi-distinguishing formulas $\exists x_2;\ \mathrm{AX}((\forall x_2 \neg a.x_2) \vee a.x_2)$ for $\langle q_0, q_2, x_2 \mapsto x'_1\rangle$, and $\exists x_2;\ \mathrm{AX}(a.x_2 \vee \mathrm{AX}b.x_2)$ for $\langle q_0, p_0, x_2 \mapsto \bot\rangle$.

Finally, the function $x_2 \mapsto \top$ succeeds in matching q_1 and p_1 by $x_2 \mapsto x_1$, and q_2 and p_2 by $x_2 \mapsto x'_1$ or $x_2 \mapsto \top$, but then condition (3) is violated. A semi-distinguishing formula for $\langle q_0, p_0, x_2 \mapsto \top\rangle$ is $\exists x_2;\ \mathrm{AX}(a.x_2 \vee \mathrm{AX}(b.x_2) \vee (\forall x_2 \neg a.x_2) \vee \exists x_2; a.x_2)$.

While $\mathcal{B}_2$ satisfies all these formulas, the abstract system $\mathcal{B}_1$ does not satisfy $\exists x_2;\ \mathrm{AX}(a.x_2 \vee \mathrm{AX}(b.x_2))$.

6 A Game-Theoretic Approach to Simulation

We present a game-theoretic approach to simulation of abstract systems. As usual, the game players are the prover $\mathcal{P}$, who wishes to prove that $\mathcal{A}_1 \preceq \mathcal{A}_2$, and the refuter $\mathcal{R}$, who wishes to prove that $\mathcal{A}_1 \not\preceq \mathcal{A}_2$.

Recall that in the standard setting, in a game for systems A_1 and A_2, a game position is a pair of states $\langle q_1, q_2\rangle$ where q_1, q_2 are states of A_1 and A_2, respectively, and $\mathcal{R}$ is in location q_1 and $\mathcal{P}$ is in location q_2. A play continues along single paths in each system.

In abstract systems, the situation is a bit more complicated. First, in its every move, $\mathcal{P}$ must choose both a state q_2 and a function f that matches the variables of q_1 and q_2. Therefore, the game positions are tuples of the form $\langle q_1, q_2, f\rangle$. For $\mathcal{P}$ to prove the existence of simulation, f must be consistent w.r.t. the function it chose in its previous move, and with the transitions both players took from the previous position. Second,

following a single path may not suffice for $\mathcal{R}$ to prove that there is no simulation. We demonstrate these points below.

Consider the systems $\mathcal{A}_1$ and $\mathcal{A}_2$ presented in Figure 4. It holds that $\mathcal{A}_1 \not\preceq \mathcal{A}_2$, since x_1 is reset and can be assigned different values in q_0 and q_2, whereas x_2 must be assigned the same value in p_0 and p_2. Consider a possible play between $\mathcal{P}$ and $\mathcal{R}$. The refuter $\mathcal{R}$ must start at q_0, and $\mathcal{P}$ must choose p_0 and $x_2 \mapsto x_1$ to match q_0 with p_0. Then, $\mathcal{R}$ continues to q_1, and $\mathcal{P}$ must move to p_1 and choose $x_2 \mapsto \bot$, since x_1 is reset along $\langle q_0, q_1 \rangle$, and x_2 is not reset along $\langle p_0, p_1 \rangle$. Finally, $\mathcal{R}$ moves to q_2, and $\mathcal{P}$ must move to p_2. To match q_2 with p_2, $\mathcal{P}$ must choose $x_2 \mapsto x_1$, but this is inconsistent with the function $x_2 \mapsto \bot$ that $\mathcal{P}$ chose in its previous move, and $\mathcal{R}$ wins.

Next, consider the systems $\mathcal{B}_1$ and $\mathcal{B}_2$. It holds that $\mathcal{B}_1 \not\preceq \mathcal{B}_2$, since x_1 and x'_1 can be assigned different values in $\mathcal{B}_1$, whereas both occurences of x_2 in $\mathcal{B}_2$ must be equally assigned. In the first round, $\mathcal{P}$ chooses a function f_0 to match q_0 with p_0. Since the labeling of p_0, q_0 is empty, f_0 can match x_2 with either $x_1, x'_1, \bot$ or $\top$. If $f_0(x_2) = x_1$ (or $f_0(x_2) = \bot$), then $\mathcal{R}$ can follow q_0, q_2, q_3, forcing $\mathcal{P}$ to follow p_0, p_2, p_3, and fail finding a match for q_3 and p_3 that is consistent with $x_2 \mapsto x_1$ (or with $x_2 \mapsto \bot$). Similarly, if $f_0(x_2) = x'_1$, then $\mathcal{R}$ follows q_0, q_1, and $\mathcal{P}$ must follow p_0, p_1, and fails to match q_1 and p_1.

However, if $f_0(x_2) = \top$, then if $\mathcal{R}$ follows either q_1 or q_2, q_3, then $\mathcal{P}$ can respond with suitable functions $x_2 \mapsto x_1$ to match p_1, or $x_2 \mapsto \top, x_2 \mapsto x'_1$ to match p_2, p_3 respectively. Thus, by following a single path, $\mathcal{P}$ may violate condition (3) of the simulation conditions, and (wrongly) win the game. We conclude that the game rules must enforce condition (3). In this case, this means that $\mathcal{P}$ may not assign x_2 non-$\bot$ values along both q_1 and q_2.

To enforce condition (3), we allow $\mathcal{R}$ to continue to both q_1 and q_2 in the same move. Then, to follow condition (3), $\mathcal{P}$ must choose $x_2 \mapsto \bot$ in either q_1 or q_2. Choosing $x_2 \mapsto \bot$ in q_1 causes $\mathcal{P}$ to fail in q_1. If $\mathcal{P}$ chooses $x_2 \mapsto \bot$ in q_2, then $\mathcal{R}$ can continue from q_2 to q_3, again causing $\mathcal{P}$ to get stuck.

The same holds also for the systems $\mathcal{C}_1$ and $\mathcal{C}_2$. Consider the first round in a play and the possible choices of $\mathcal{P}$ to match q_0 with p_0. The variable x_2 must be matched with x_1 when matching q_1 with p_1, and so $\mathcal{P}$ fails if it chooses $x_2 \mapsto \bot$. Also, since x_1 is reset in $\langle q_0, q_1 \rangle$, and x_2 is not reset in $\langle p_0, p_1 \rangle$, then $\mathcal{P}$ fails if it chooses $x_2 \mapsto x_1$. Finally, if $\mathcal{P}$ chooses $x_2 \mapsto \top$, then $\mathcal{R}$ can split as in the case of $\mathcal{B}_1$ and $\mathcal{B}_2$, and cause $\mathcal{P}$ to get stuck in either q_1 or q_3.

Thus, as opposed to the classical setting, when there is no simulation, the inability to locally match states is not enough to refute the existence of a simulation. Also, a single path may not suffice to properly refute, and two paths are needed, as in the case of the systems $\mathcal{B}_1$ and $\mathcal{B}_2$, and $\mathcal{C}_1$ and $\mathcal{C}_2$. Note that two paths suffice, since x_2 may only be assigned a non-$\bot$ value along one path, causing $\mathcal{P}$ to get stuck in the other, in case that x_2 appears in both paths without being reset.

Accordingly, we define the game such that $\mathcal{R}$ continues along a single path (mode (a) in the game), but may, at any point, choose two states on two different paths splitting from its current location (mode (b) in the game). $\mathcal{P}$ then chooses matching states and functions for both states. Then, $\mathcal{R}$ chooses along which of the two paths the play continues from, and switches back to mode (a). $\mathcal{P}$ then responds accordingly.

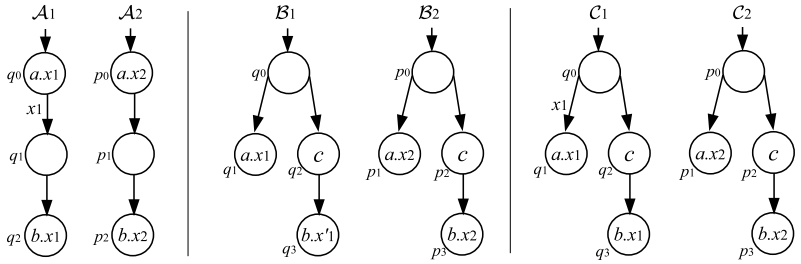

Fig. 4. The abstract systems $\mathcal{A}_1$ and $\mathcal{A}_2$, and $\mathcal{B}_1$ and $\mathcal{B}_2$, and $\mathcal{C}_1$ and $\mathcal{C}_2$.

We formally define the game. A play begins as follows. $\mathcal{R}$ starts from q_1^0, and $\mathcal{P}$ starts from q_2^0 and chooses some function f_0 such that f_0 is a match for q_1^0 and q_2^0. Then, the game continues in mode (a) according to the following rules.

Let the current position be $\langle q_1, q_2, f \rangle$. If the game is in mode (a),

1. $\mathcal{R}$ chooses a transition $\langle q_1, q_1' \rangle \in R_1$ and moves to q_1', or switches to mode (b),
2. If $\mathcal{R}$ hasn't switched to mode (b), then $\mathcal{P}$ chooses a transition $\langle q_2, q_2' \rangle \in R_2$, and a function f' that is consistent with f w.r.t. $\langle q_1, q_1' \rangle$ and $\langle q_2, q_2' \rangle$, and moves to q_2'.

If the game is in mode (b),

1. $\mathcal{R}$ chooses two different transitions $\langle q_1, q_1' \rangle, \langle q_1, q_1'' \rangle \in R_1$.
2. $\mathcal{P}$ chooses a transition $\langle q_2, q_2' \rangle$ and a function f' such that f' is consistent with f w.r.t. $\langle q_1, q_1' \rangle$ and $\langle q_2, q_2' \rangle$, and dually a transition $\langle q_2, q_2'' \rangle$ and a function f'' for $\langle q_1, q_1'' \rangle$. Further, for every $x_2 \in X_2$ such that $f(x_2) = \top$, either $f'(x_2) = \bot$ or $f''(x_2) = \bot$.
3. $\mathcal{R}$ chooses between the game positions $\langle q_1', q_2', f' \rangle$ and $\langle q_1'', q_2'', f'' \rangle$, and switches to mode (a) from the chosen position.

Notice that a round does not exactly alternate between $\mathcal{R}$ and $\mathcal{P}$; from mode (a), $\mathcal{R}$ can switch to mode (b) and continue to choose two transitions as described. Also, after $\mathcal{R}$ chooses a single position at the last step of mode (b), it continues to choose a transition in the first step in mode (a), as described.

If at some point $\mathcal{P}$ cannot continue according to the game rules, then $\mathcal{R}$ wins. Otherwise, the play continues forever and $\mathcal{P}$ wins.

The simulation game indeed captures simulation of abstract systems: both players have a winning strategy, according to the existence or lack thereof of a suitable simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$.

Theorem 3. *$\mathcal{P}$ has a winning strategy in the simulation game for $\mathcal{A}_1$ and $\mathcal{A}_2$ iff $\mathcal{A}_1 \preceq \mathcal{A}_2$, and $\mathcal{R}$ has a winning strategy in the simulation game for $\mathcal{A}_1$ and $\mathcal{A}_2$ iff $\mathcal{A}_1 \not\preceq \mathcal{A}_2$.*

The winning strategy of $\mathcal{P}$ corresponds to a suitable simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$, whereas the winning strategy of $\mathcal{R}$ corresponds to a distinguishing VACTL formula for $\mathcal{A}_1$ and $\mathcal{A}_2$. Despite the more complicated nature of the game, the winning strategies of both players, just like in the standard setting, are memoryless.

7 Discussion and Future Work

We have shown that the properties of standard simulation can be lifted to the setting of abstract systems. In this section we describe further theoretical challenges of abstract systems and simulation and their practical applications.

Consider a system $\mathcal{S}$ that describes the behavior of n processes. Typically, each process is associated with a set of atomic propositions, parameterized with its id. For example, it is common to see atomic propositions like $try_1, \ldots, try_n$ and $grant_1, \ldots, grant_n$. Consider now the system $\mathcal{S}'$ obtained by uniting all "underlying" atomic propositions, try and $grant$ in our example, and parameterizing them by a variable that ranges over the domain of processes ids. As another example, consider a system $\mathcal{S}$ in which messages from a list $L = \{l_1, \ldots, l_n\}$ of possible messages can be sent, and atomic propositions are parameterized by messages from L, say atomic propositions are $send_{l_1}, \ldots, send_{l_n}$ and $receive_{l_1}, \ldots, receive_{l_n}$. Again, we can obtain a system $\mathcal{S}'$ that abstracts the content of the messages and uses atomic propositions $send$ and $receive$, parameterized by variables that range over L. Note that the behavior of $\mathcal{S}$ with respect to the different values of processes ids or messages may be different. thus $\mathcal{S}'$ over-approximates $\mathcal{S}$. Clearly, $\mathcal{S}'$ is also simpler than $\mathcal{S}$.

It is straightforward to extend the definition of simulation to cases in which only the simulating system is abstract, and to do it in such a way that $\mathcal{S}'$ simulates $\mathcal{S}$. Essentially, as noted above, $\mathcal{S}'$ includes all the behaviors with respect to all values of processes ids or messages, and also abstracts their number. We can thus use $\mathcal{S}'$ not only for the verification of $\mathcal{S}$, but also, in case we abstract identical processes, for checking whether the specification is sensitive to their number, and for generalizing the reasoning to an arbitrary number of processes. We plan to formalize this method, namely define the simulation relation, a refinement procedure, and the connection to other methods of verification of parameterized systems.

The simulation pre-order defined here is the branching-time counterpart of trace containment, which is undecidable for abstract systems. In standard systems, the computational efficiency of the branching-time approach is carried over in the model-checking problem for CTL. In [12, 13] we studied the model-checking problem for VLTL and abstract systems and pointed to useful fragments for which the complexity of the problem is in PSPACE – as is LTL model checking. We plan to study the model-checking problem for VCTL and abstract systems, and check whether the advantage of the branching approach is carried over also to this setting.

References

1. R. Paige B. Bloom. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Science of Computer Programming*, 24:189–220, 1996.
2. S. Bensalem, A. Bouajjani, C. Loiseaux, and J. Sifakis. Property preserving simulations. In *CAV 1992*, volume 663 of *LNCS*, pages 260–273. Springer, 1992.
3. M. Bojańczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3):1–48, 2009.
4. M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *LICS 2006*, pages 7–16, 2006.

5. M.C. Browne, E.M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59:115–131, 1988.
6. E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV 2000*, volume 1855 of *LNCS*, pages 154–169. Springer, 2000.
7. R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics-based tool for the verification of concurrent systems. *ACM TOPLAS*, 15:36–72, 1993.
8. W. Damm and A. Pnueli. Verifying out-of-order executions. In *Proc. 9th Conf. on Correct Hardware Design and Verification Methods*, pages 23–47. Chapman & Hall, 1997.
9. S. Demri and De. Dsouza. An automata-theoretic approach to constraint ltl. In *FST TCS 2002*, LNCS, 2002.
10. S. German and A. Prasad Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39:675–735, 1992.
11. O. Grumberg, O. Kupferman, and S. Sheinvald. Variable automata over infinite alphabets. volume 6031 of *LNCS*, pages 561–572. Springer, 2010.
12. O. Grumberg, O. Kupferman, and S. Sheinvald. Model checking systems and specifications with parameterized atomic propositions. In *ATVA*, pages 122–136, 2012.
13. O. Grumberg, O. Kupferman, and S. Sheinvald. An automata-theoretic approach to reasoning about parameterized systems and specifications. In *ATVA*, pages 397–411, 2013.
14. O. Grumberg and D.E. Long. Model checking and modular verification. volume 527 of *LNCS*, pages 250–265. Springer, 1991.
15. O. Grumberg and D.E. Long. Model checking and modular verification. *ACM Transactions on Programming Languagues and Systems*, 16(3):843–871, 1994.
16. S. Hallé, R. Villemaire, and O. Cherkaoui. Ctl model checking for labelled tree queries. In *TIME*, pages 27–35, 2006.
17. T.A. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. *Information and Computation*, 173(1):64–81, 2002.
18. M. Kaminski and D. Zeitlin. Extending finite-memory automata with non-deterministic reassignment. In Csuhaj-Varjú and Z. E., Ézik, editors, *AFL*, pages 195–207. In eds.:, 2008.
19. Y. Kesten, N. Piterman, and A. Pnueli. Bridging the gap between fair simulation and trace containment. In *CAV 2003*, volume 2725 of *LNCS*, pages 381–393. Springer, 2003.
20. L. Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languagues and Systems*, 5:190–222, 1983.
21. R. Lazic. Safely freezing LTL. In *FST STC 2006*, pages 381–392. Springer, 2006.
22. N. A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 137–151, 1987.
23. N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
24. R. Milner. An algebraic definition of simulation between programs. In *Proc. 2nd Int. Joint Conf. on Artificial Intelligence*, pages 481–489. British Computer Society, 1971.
25. F. Neven, T. Schwentick, and V. Vianu. Towards regular languages over infinite alphabets. In *MFCS '01*, pages 560–572, London, UK, 2001. Springer-Verlag.
26. A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. pages 510–584, Volume 224, LNCS, Springer, 1985.
27. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *CAV 1997*, volume 1254, pages 72–83. Springer, 1997.
28. Y. Shemesh and N. Francez. Finite-state unification automata and relational languages. *Information and Computation*, 114:192–213, 1994.
29. L.J. Stockmeyer and A.R. Meyer. Word problems requiring exponential time. In *Proc. 5th ACM Symp. on Theory of Computing*, pages 1–9, 1973.
30. T. Tan. *Pebble Automata for Data Languages: Separation, Decidability, and Undecidability*. PhD thesis, Technion - Computer Science Department, 2009.