Simulation Relations and Applications in Formal Methods

Kim Guldstrand Larsen[1 [0000−0002−5953−3384]], Christian Schilling[1 [0000−0003−3658−1065]], and Jiří Srba[1 [0000−0001−5551−6547]]

Department of Computer Science, Aalborg University, Aalborg, Denmark
{kgl,christianms,srba}@cs.aau.dk)

Abstract. We survey the research on application of equivalence checking to formal methods, with a particular focus on the notion of simulation and bisimulation as well as of modal refinement on modal transition systems. We discuss the algorithmic aspects of efficiently computing (bi)simulation relations, the extension to infinite state systems, and existing tool support. We then present results related to simulation and bisimulation checking on timed and hybrid systems and highlight the connections to automata theory.

Keywords: Simulation · Bisimulation · Transition system · Hybrid system · Automata theory.

1 Foreword and Outline

In this chapter we review simulation relations and their success story in formal methods. Written at the occasion of Tom Henzinger's 60th anniversary, we focus on three areas where he substantially contributed to this topic, and describe selected works in detail. Section 3 covers the general computation of simulation relations for finite and infinite transition systems. Section 4 covers simulation relations in timed and hybrid systems. Section 5 covers simulation relations in automata theory. We begin with a general introduction to simulation relations and related concepts in the next section.

2 Transition Systems, Simulation, and Bisimulation

In this section we outline the basic notions underlying the topics discussed in the later sections: transition systems, simulation and bisimulation relations, modal transition systems, and modal refinement. For a detailed introduction to these topics we refer to the literature [37,14,46,10].

2.1 Transition Systems

Simulation relations are defined over (labeled) transition systems. These are structures to abstractly describe the behavior of systems and coincide mathematically with directed edge-labeled graphs. A transition system consists of a

(finite or infinite) set of states (the nodes in the graph) and a set of transitions connecting pairs of states (the directed edges in the graph).

Definition 1 (Transition system). *A* transition system *is a triple* $TS = (S, \Lambda, T)$ *where* S *is a set of* states, Λ *is a set of* labels, *and* $T \subseteq S \times \Lambda \times S$ *is a* transition relation *whose elements are called* transitions.

We write $s \xrightarrow{a} t$ for transition $(s, a, t) \in T$ and generalize it to a sequence of transitions such that $s_0 \xrightarrow{a_1 \ldots a_n} s_n$ if there are states $s_1, \ldots, s_{n-1} \in S$ where $s_{i-1} \xrightarrow{a_i} s_i$ for all $i = 1, \ldots, n$. We use the following terminology: a sequence of transitions is a *path*, the projection of a path to the states is a *run*, and the projection of a path to the labels is a *trace*. Figure 1 shows an example transition system over labels a and b with seven states and eight transitions.

Transition systems are rather general models. Many systems encountered in computer science can be modeled with a transition system. Once a system has been modeled as a transition system, one can use all the tools that have been developed for their analysis. But this generality comes with a price: transition systems for most interesting systems are huge or even infinite. This implies that typical graph algorithms such as state space search are expensive or may not even terminate.

The main approach to reduce the size of transition systems, possibly even making an infinite transition system finite, is through quotienting. For an equivalence relation $\equiv \subseteq X \times X$ we denote the equivalence class of $e \in X$ by $[e]_\equiv$. Given a transition system and an equivalence relation over its states, the induced *quotient transition system* is obtained by merging all equivalent states and remapping the corresponding transitions. Formally:

Definition 2 (Quotient transition system). *Given a transition system* $TS = (S, \Lambda, T)$ *and an equivalence relation* $\equiv \subseteq S \times S$, *the* quotient transition system *is* $TS/_\equiv = (S/_\equiv, \Lambda, T/_\equiv)$ *with states* $S/_\equiv = \{[s]_\equiv \mid s \in S\}$ *and transitions* $T/_\equiv = \{([s]_\equiv, a, [t]_\equiv) \mid (s, a, t) \in T\}$.

Note that we have not put any restriction on $\equiv$ above. Thus we cannot guarantee many properties about the quotient system. What is true for any relation $\equiv$ is that reachability is preserved: if there is a path $s \xrightarrow{w} t$ in TS, then there is a path $[s]_\equiv \xrightarrow{w} [t]_\equiv$ in $TS/_\equiv$. Guaranteeing more interesting properties, such as trace equivalence, requires more structure in the relation $\equiv$. Prominent families of relations that provide such structure are simulations and bisimulations.

2.2 Simulation and Bisimulation

The concept of simulation relations, or simulations for short, dates back to Milner [83]. Roughly speaking, a state s' simulates a state s if any transition from s to a state t can be matched by a transition with the same label leading from s' to a state t' such that t' again simulates the state t.

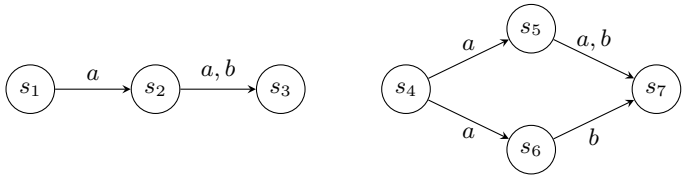

Fig. 1. A transition system.

Definition 3 (Simulation). *Given a transition system $TS = (S, \Lambda, T)$, a binary relation $R \subseteq S \times S$ is a* simulation *if for all $a \in \Lambda$ and $s, s' \in S$ with $(s, s') \in R$ the following holds: for all $t \in S$ such that $s \xrightarrow{a} t$ there exists $t' \in S$ such that $s' \xrightarrow{a} t'$ and $(t, t') \in R$.*

If $(s, s') \in R$, we say that s' simulates s. While simulations are locally defined, they also provide the global guarantee that, intuitively, s' has at least the same set of traces as s (and hence simulation implies trace inclusion). Note that for a given transition system there may exist multiple simulations, and that simulations need neither be reflexive nor transitive. However, the union of all simulations $\preccurlyeq$ is itself a simulation (namely the coarsest one), which is a reflexive and transitive relation called the *simulation preorder*. In the following, we are typically interested in the simulation preorder. The simulation preorder $\preccurlyeq$ induces a *similarity* relation $\simeq \subseteq S \times S$ such that $s \simeq t$ if and only if $s \preccurlyeq t$ and $t \preccurlyeq s$. Thus $\simeq$ is the maximal symmetric subset of $\preccurlyeq$ and hence an equivalence relation. A symmetric simulation $R \subseteq S \times S$ is called a *bisimulation* [86]. The following definition is equivalent:

Definition 4 (Bisimulation). *Given a transition system $TS = (S, \Lambda, T)$, a binary relation $R \subseteq S \times S$ is a* bisimulation *if for all $a \in \Lambda$ and $s, s' \in S$ with $(s, s') \in R$ the following holds:*

- *for all $t \in S$ such that $s \xrightarrow{a} t$ there exists $t' \in S$ such that $s' \xrightarrow{a} t'$ and $(t, t') \in R$, and*
- *for all $t' \in S$ such that $s' \xrightarrow{a} t'$ there exists $t \in S$ such that $s \xrightarrow{a} t$ and $(t, t') \in R$.*

As for simulations, bisimulations are not unique and they may be neither reflexive nor transitive. Again, the union of all bisimulations $\sim$ is itself a bisimulation (namely the coarsest one), which is an equivalence relation called *bisimilarity*. In the following, we are typically interested in the bisimilarity relation.

Observe that similarity $\simeq$ is generally coarser than bisimilarity $\sim$ (as witnessed in the example below). Thus computing the similarity relation may be more interesting; for instance, when used for quotienting, it yields a smaller transition system. However, computing a simulation relation is typically harder than computing the corresponding bisimulation relation [68].

Consider the transition system in Figure 1. The states s_3 and s_7 have no outgoing transitions, so they cannot be distinguished and must be similar and

$\preccurlyeq$	s_1	s_2	s_3	s_4	s_5	s_6	s_7
s_1	✓			✓			
s_2		✓			✓		
s_3	✓	✓	✓	✓	✓	✓	✓
s_4	✓			✓			
s_5		✓			✓		
s_6		✓			✓	✓	
s_7	✓	✓	✓	✓	✓	✓	✓

$\sim$	s_1	s_2	s_3	s_4	s_5	s_6	s_7
s_1	✓						
s_2		✓			✓		
s_3			✓				✓
s_4				✓			
s_5		✓			✓		
s_6						✓	
s_7			✓				✓

Table 1. Simulation preorder $\preccurlyeq$ and bisimilarity $\sim$ for the transition system in Figure 1. A pair in the relation is marked with ✓. The shaded cells correspond to similarity $\simeq$.

bisimilar: $s_3 \preccurlyeq s_7 \preccurlyeq s_3$ and $s_3 \sim s_7$. The states s_2 resp. s_5 only have transitions with a and b to s_3 resp. s_7, which we already know are bisimilar; hence the same holds for s_2 and s_5: $s_2 \preccurlyeq s_5 \preccurlyeq s_2$ and $s_2 \sim s_5$. The state s_6 on the other hand has no outgoing a-transition and hence cannot simulate (nor bisimulate) s_2 and s_5, but those states simulate s_6: $s_6 \preccurlyeq s_2$ and $s_6 \preccurlyeq s_5$. The state s_1 clearly simulates s_4, but the converse also holds due to s_5: $s_1 \preccurlyeq s_4 \preccurlyeq s_1$. Yet s_1 and s_4 are not bisimilar because of s_6. The complete relations are given in Table 1. The shaded cells mark the similarity relation $\simeq$; observe that here similarity $\simeq$ is strictly coarser than bisimilarity $\sim$, as it includes the pairs (s_1, s_4) and (s_4, s_1).

There is also a game-theoretic characterization of the simulation preorder (see [93,95] for an early proposal and discussion and [82] for a detailed introduction). Consider the two-player "imitation game" where each player has a token on one state. In each round, the first player, called antagonist, moves their token along a transition and the second player, called protagonist, has to move the other token along a transition of the same label. The game ends if a player cannot move, making the other player win. Otherwise, the game is infinite and the protagonist wins. We have that $s \preccurlyeq t$ holds for states s and t if, in a game where the antagonist's token starts on s and the protagonist's token starts on t, the protagonist has a winning strategy. That means: a rational protagonist can always imitate the antagonist's move.

A similar characterization exists for bisimilarity. The only change of rules is that, at the beginning of each round, the antagonist may choose to swap the tokens. Hence the antagonist has more chances to win. From this characterization it is clear that bisimilarity is symmetric and generally finer than similarity.

In the context of model checking, simulation and bisimulation are alternatively defined for *Kripke structures*, which are transition systems with state labels. In that case, the additional requirement for a state s simulating a state t is that the labels of s and t are identical. If we consider the labels of a state "public information," this is in line with the original intuition that s simulates t if an observer cannot determine whether a trace starting from t may have started from s instead. See also [14] for a discussion of these two system models.

We have motivated simulations for the purpose of quotienting to reduce the size of a transition system. More generally, one can use simulations to establish a formal relation between transition systems. If we consider transition systems with initial states, then a transition system TS_1 simulates a transition system TS_2 if for each initial state of TS_2 there is a simulating initial state of TS_1. Simulation can be used as a formal proof that a reactive system, e.g., a controller, works correctly in an adversary environment: to stay in the game-theoretic view, the environment takes the role of the antagonist and the controller takes the role of the protagonist. Similarly, one can establish levels of abstractions of systems, e.g., a specification and an implementation, which are also called refinement mappings [1]. Quotienting is one way to find such abstractions (see [19,30,36] for some examples).

2.3 Modal Transition Systems and Modal Refinement

An important practical application of equivalence checking is a component-based software development and stepwise refinement process. Assuming that we have a system specification expressed as a transition system (that is usually generated from some higher-level specification language), our aim is to refine (possibly in several steps) the given component specification by adding more implementation details, while preserving a suitable equivalence/preorder with the initial specification. However, should bisimulation be used as the notion of equivalence, we are required to describe all the implementation details already in the specification because bisimulation is a too strict notion requiring that any action of the specification must be matched in the implementation and vice versa. On the other hand, the notion of simulation allows us to create several variants of the given specification, as it only requires that any refined process must be simulated by its specification (and hence every behavior of the refined process is guaranteed to be sound). The drawback is that an empty implementation (where the initial state is deadlocked) is trivially simulated by any given specification, whereas it is clearly not an intended product implementation.

It is hence clear that for a usable stepwise refinement process, we need to use a relation/preorder that is less strict than bisimulation but at the same time it must enforce some minimum system behavior that a simulation relation cannot guarantee. One possible answer to this problem was suggested by Larsen and Thomsen [75] (for an overview paper see also [10]) in terms of *modal transition systems* and the notation of *modal refinement*.

In a modal transition system, transitions are split into *may* and *must* transitions: any refinement of a specification is then allowed to implement any may-transition, and at the same time it is required to preserve any must-transition. This allows to encode some minimal required process behavior while at the same time allowing for different variants during the refinement process as may-transitions are not mandatory to be implemented.

Definition 5 (Modal transition system). *A* modal transition system *is a tuple* $TS = (S, \Lambda, T_{may}, T_{must})$ *where* S *is a set of* states, Λ *is a set of* labels, *and* $T_{must} \subseteq T_{may} \subseteq S \times \Lambda \times S$ *are* may *resp.* must *transition relations.*

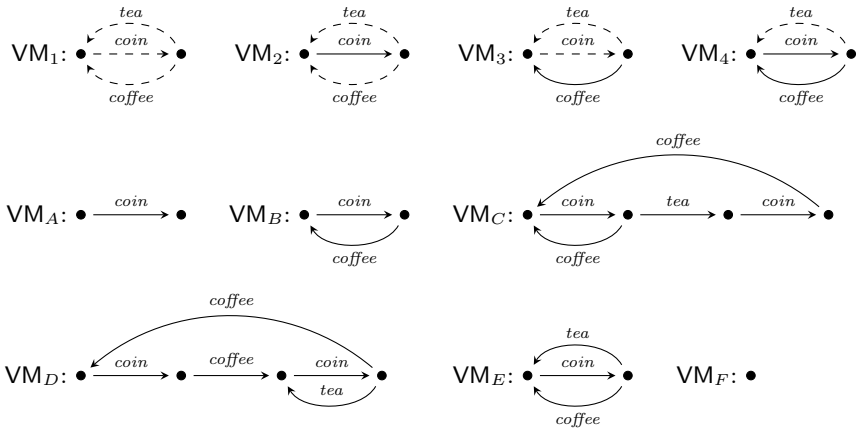

Fig. 2. Four specifications of a vending machine, VM_1-VM_4, and six different implementations VM_A-VM_F.

We write $s \xrightarrow{a} t$ for transition $(s, a, t) \in T_{must}$ and $s \overset{a}{\dashrightarrow} t$ for transition $(s, a, t) \in T_{may} \setminus T_{must}$. Intuitively, every must-transition of the specification must be matched in the refined process, and every may-transition of a refined process must be matched in the specification. This gives rise to a co-inductive definition of a modal refinement, generalizing the notion of bisimulation to modal transition systems.

Definition 6 (Modal refinement). *Given a modal transition system* $TS = (S, \Lambda, T_{may}, T_{must})$*, a binary relation* $R \subseteq S \times S$ *is a* modal refinement *if for all* $a \in \Lambda$ *and* $s, s' \in S$ *with* $(s, s') \in R$ *the following holds:*

- *for all* $t \in S$ *such that* $s \overset{a}{\dashrightarrow} t$ *there exists* $t' \in S$ *such that* $s' \overset{a}{\dashrightarrow} t'$ *and* $(t, t') \in R$*, and*
- *for all* $t' \in S$ *such that* $s' \xrightarrow{a} t'$ *there exists* $t \in S$ *such that* $s \xrightarrow{a} t$ *and* $(t, t') \in R$*.*

We say that s is a modal refinement of s' if there exists a modal refinement relation R such that $(s, s') \in R$. The definition of modal refinement resembles that of simulation/bisimulation. Indeed, if every may-transition is also a must-transition ($T_{may} = T_{must}$), modal refinement and bisimulation coincide. On the other hand, if the modal transition system does not contain any must transitions, the obtained notion of modal refinement defines a simulation relation.

In Figure 2 the authors of [16] demonstrate four possible specifications of a vending machine and six possible implementations. The first specification VM_1 does not require any behavior (contains no must-transitions), but it allows to execute a coin-transition, after which both tea and coffee-transitions may be executed. All six implementations $\mathsf{VM}_A, \ldots, \mathsf{VM}_F$ are refinements of VM_1. The second specification VM_2 requires that the coin-transition must be present but

implementing the tea and coffee-transitions is optional. This excludes the machine VM_F that contains only the deadlock state from being a refinement of VM_2. The third specification VM_3 requires that, if a coin is inserted, at least the coffee-transition must be implemented. This excludes VM_A from being a correct implementation of VM_3 while all other implementations (including VM_F) are in refinement with VM_3. Finally, the specification VM_4 ensures that inserting the coin and providing a coffee is mandatory while returning tea is optional. There are four valid refinements of VM_4, namely VM_B, VM_C, VM_D and VM_E.

Hence modal transition systems in connection with modal refinement relations allow for a stepwise refinement process in system modeling and generalize the notion of simulation and bisimulation, allowing to be more specific which transitions should be preserved in which direction. For further results about modal transition systems and its extensions, we refer to [10,11,12,73,15,17].

3 Computing Simulations on Finite and Infinite Graphs

In this section we review results about computing (bi)simulations for finite and infinite graphs. The work of Tom Henzinger we summarize here is [88]. That paper has two main contributions. The first contribution is an efficient algorithm to compute the simulation preorder on finite transition systems, together with an extension to infinite transition systems. The second contribution is an algorithm to compute the similarity relation for a class of hybrid automata, which is discussed in the next section.

Two states are *trace equivalent* if the set of outgoing traces coincide. Trace equivalence is coarser than similarity. The authors advocate similarity as a middle ground between bisimilarity and trace equivalence, for the following reasons.

First there is the aspect of computational complexity. Computing bisimilarity for finite transition systems with n states and m transitions is an $O(m \log n)$ problem [85]. Computing trace equivalence is a PSPACE-complete problem [94]. The authors propose an $O(mn)$ algorithm for computing the simulation preorder (a similar result has been independently obtained at the same time [20]). For an overview of the algorithms for computing bisimulation we refer to [4]. Similarly, deciding if two systems are in modal refinement can be done in polynomial time, while checking whether every implementation of one specification is also an implementation of another specification becomes EXPTIME-complete [18].

Second, similarity is coarser than bisimilarity, and so the corresponding quotient is smaller, but at the same time the quotient still preserves useful properties. On the one hand, two states are bisimilar if and only if they satisfy the same formulae in branching temporal logic (CTL or CTL*). Hence for such logics, similarity is not of interest. On the other hand, two states are trace equivalent if and only if they satisfy the same formulae in the widely used linear temporal logic (LTL). Moreover, two states are similar if and only if they satisfy the same formulae in branching temporal logic without quantifier switches. (See [14] for proofs of these statements.) Thus computing the similarity quotient is useful.

Third, as a consequence of the smaller quotient, transition systems with infinite bisimilarity quotient may still yield a finite similarity quotient, which allows for effective computations and decision procedures. The authors extend their algorithm to a symbolic algorithm applicable to infinite but "effectively presented" transition systems, in the line of [21,77]. The symbolic algorithm terminates if the similarity quotient is finite.

3.1 Further Results About Infinite State Systems

The decidability of bisimilarity, equivalence with a given finite state system, as well as the regularity problem (does there exist a finite state system equivalent to the given system) were extensively studied for different types of process algebra generating infinite state systems (for an overview see [91]; further details about the used techniques can be found in [84,24,70]). The classes of infinite state systems include the process algebras BPA (basic process algebra) and BPP (basic parallel processes) that support a pure sequential resp. parallel composition, their generalization PA (process algebra) allowing to mix both the sequential and parallel operators, as well as transition systems described by Petri nets (a model of parallel processes allowing for synchronization) and pushdown automata (adding a finite control-state unit to the BPA processes).

All these systems can be uniformly described by the formalism of *process rewrite systems* suggested by Mayr [80], and the results indicate (for references consult [91]) that for the simple process algebras BPA and BPP, bisimulation and regularity are decidable, and bisimilarity of a BPA or BPP process with a given finite state system can be decided even in polynomial time. Decidability of bisimulation, equivalence with a finite state system and regularity is preserved for the class of pushdown automata, however, with higher complexity bounds. Decidability of bisimulation and regularity for PA process algebra remains an open problem, while for Petri nets only regularity and equivalence with a finite state system remains decidable while bisimulation checking becomes undecidable by the application of the defender's forcing technique [64].

3.2 Tools for Equivalence Checking

There exist a number of tools supporting equivalence/bisimulation checking, including Edinburgh Concurrency Workbench and its successors [32,31,33], and tools like CADP [44], mCRL2 [23], TAPAs [26] and FDR3 [47]. Several of these tools rely on the fixed-point calculation of the bisimilarity. More recently, on-the-fly methods based on dependency graphs [40] have been used in tools like CAAL [9] and allow for an early termination without the need of enumerating the full state space.

4 Simulation Relations for Timed and Hybrid Systems

In this section we review simulation relations in the context of hybrid systems [5,54], i.e., dynamical systems with mixed discrete and continuous behavior.

4.1 Timed Systems

The *timed automaton* model, introduced by Alur and Dill [6,7], is an established formalism for describing the behavior of real-time systems. Initially, the focus was on model checking with respect to a variety of logics, and only later the notions of (timed and untimed) bisimulation and simulation were considered.

Given a finite set of clocks C, we denote by $\Phi(C)$ all conjunctions of simple clock constraints of the form $x \bowtie k$, where $x \in C$, $\bowtie \in \{<, \leq, =, \geq, >\}$, $k \in \mathbb{N}$. The semantics of clocks is given by a *clock valuation* $v : C \to \mathbb{R}_{\geq 0}$ assigning non-negative real values to clocks. Thus a clock constraint ϕ denotes a set of clock valuations. By $v \in \phi$ we mean that the valuation v satisfies the constraint ϕ. If v is a clock valuation and $d \in \mathbb{R}_{\geq 0}$, $v + d$ is the clock valuation such that $(v + d)(x) = v(x) + d$ for all $x \in C$. Also, if $r \subseteq C$, we denote by $v[r]$ the clock valuation where $v[r](x) = 0$ if $x \in r$ and $v[r](x) = v(x)$ if $x \notin r$.

Definition 7. *A timed automaton is a tuple $\mathcal{A} = (Loc, \ell_0, C, \Lambda, I, T)$ where Loc is a finite set of locations, $\ell_0 \in Loc$ is the initial location, C is a finite set of clocks, Λ is a set of labels, $I : Loc \to \Phi(C)$ is a mapping assigning invariants to locations, and $T \subseteq Loc \times \Phi(C) \times \Lambda \times 2^C \times Loc$ is a set of transitions.*

The semantics of a timed automaton $\mathcal{A}$ is given by an infinite-state *timed* transition system, where states are location-valuation pairs (ℓ, v). Transitions are labeled with either discrete labels from Λ or delays from $\mathbb{R}_{\geq 0}$ as follows:

- $(\ell, v) \xrightarrow{a} (\ell', v')$ iff $v \in g$ and $v' = v[r]$ for some transition $(\ell, g, a, r, \ell') \in T$,
- $(\ell, v) \xrightarrow{d} (\ell, v + d)$ iff $v + d \in I(\ell)$.

We shall denote by $(\ell, v) \xrightarrow{\epsilon} (\ell, v')$ that $(\ell, v) \xrightarrow{d} (\ell, v')$ for some delay $d \in \mathbb{R}_{\geq 0}$, and refer to the transition as a *time-abstracted transition*, and the resulting transition system as the *untimed* transition system.

The notions of bisimulation and simulation may readily be applied to timed automata based on either the timed or untimed transition system semantics. Now consider the four timed automata from [3] depicted in Figure 3. Here A and X are not timed bisimilar as $(X, y = 0) \xrightarrow{2} \xrightarrow{a}$ cannot be matched by $(A, y = 0)$. However, it can be seen that A and X are untimed bisimilar and that A is timed simulated by X. Considering A and U, it can be seen that they are not even untimed bisimilar (and hence not timed bisimilar): the transition $(A, y = 0) \xrightarrow{2} (A, y = 2)$ leads to a state that cannot perform a, while from $(U, y = 0)$, a is always enabled regardless of the delay. However, U timed simulates A. Finally, it can be argued that U and U' are timed bisimilar.

Several decision problems for timed automata are settled using the so-called *region* graph construction (see, e.g., [3]), essentially partitioning the infinite state-space of a timed automaton into a *finite* number of equivalence classes that are stable with respect to untimed bisimulation. From this region graph, the decidability of the untimed versions of bisimulation and simulation for timed automata follows (see, e.g., [76]).

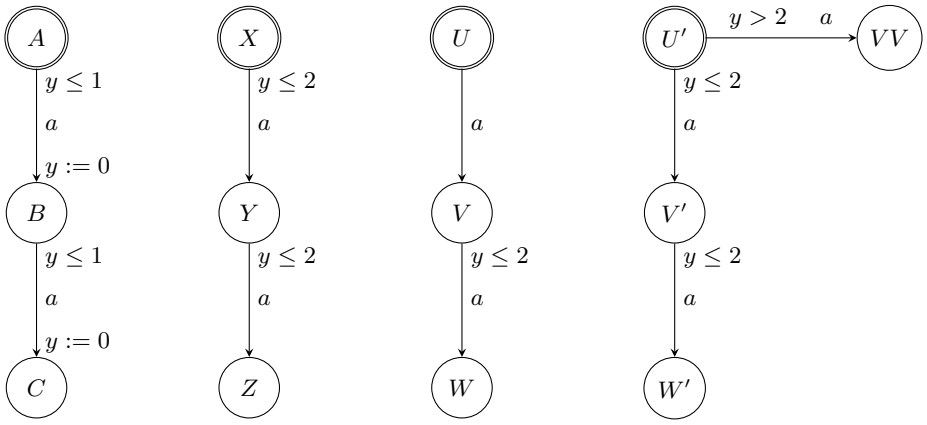

Fig. 3. Four timed automata. The initial locations are marked with double circles.

In contrast to the untimed setting, the decidability of timed bisimulation (and timed simulation) was for some time an open question, which was of particular importance to the several real-time process calculi that were developed in parallel with (and essentially equivalent to) timed automata at the time, e.g., the calculus of TCCS by Wang Yi [98]. However, in 1992 [28], Karlis Cerans conclusively demonstrated decidability of timed bisimulation through an elegant application of regions to a product construction, essentially constituting a timed game between two timed automata to be shown timed bisimilar. The decidability of synthesis for timed games in [13] provides a proof of decidability of timed bisimilarity. Another alternative proof can be obtained by reducing timed bisimulation to model checking using the characteristic formula construction given in [72].

4.2 Tools for Analyzing Timed Systems

The first tool supporting analysis of timed bisimulation (in fact a timed version of modal refinement) was the tool EPSILON [29], directly basing its implementation on the region construction by Cerans [28]. Only two years later, in 1995, the tool UPPAAL [74] was launched at the very first edition of TACAS [22]; UPPAAL is by now the standard tool for analyzing timed automata. Strongly encouraged by Tom Henzinger, the branch UPPAAL TIGA, supporting the synthesis for timed games, was launched in 2005 [27]. Later, supporting refinement between timed modal specification (in the shape of timed I/O automata), the branch ECDAR—effectively replacing EPSILON—was launched in 2010 [38].

4.3 Beyond Timed Systems

Recall Tom Henzinger's work in [88] from Section 3. As a concrete example for applying similarity, the authors discuss a subclass of rectangular hybrid

automata [58]. In that subclass, all constraints and continuous dynamics are described by Cartesian products of (nonempty, possibly open, and possibly unbounded) intervals $[l, u] \subseteq \mathbb{R}_{\geq 0}$ whose end points $l, u \in \mathbb{N}$ are natural numbers. (The restriction to nonnegative numbers is crucial.) In particular, the dynamics are described by drifting clocks. For this class of systems (which induces an uncountable but effectively presented transition system), it is known that in two dimensions the bisimilarity quotient is infinite [53]. The authors prove that the similarity quotient is finite. (It was conjectured that this result holds in higher dimensions as well, but the authors later showed that simulation and bisimulation equivalence degenerate to equality in more than two dimensions [57].) An intuitive explanation is that similarity corresponds to the intersection of the two finite bisimilarity relations obtained by looking at the extremal slopes of the drifting clocks. As a direct consequence, model checking LTL (or non-quantifier-switching branching temporal logic) formulae is decidable.

The generalization to automata where the intervals in one dimension are not restricted to nonnegative reals yields infinite similarity quotients, but the quotient tiles the plane in a regular manner such that LTL model checking is possible with a procedure based on a pushdown ω-automaton that stores the integer part of the second dimension. When relaxing the nonnegativity constraints such that both dimensions can range into the negative reals, then already the bisimilarity quotient is finite [53]. The first model checker for rectangular and linear hybrid automata was HYTECH [55].

Simulation relations and abstraction have also been applied to general nonlinear hybrid systems (e.g., [56,71,96]). A popular notion in the context of hybrid systems is *approximate (bi)simulation* [49,48]. The idea is that two systems need not be identical but remain bounded by some distance, possibly with two parameters [66]. Some notable use cases are control systems with inputs [87] and digital control systems [79].

5 Simulation Relations and Automata Theory

In this section we review simulation relations in the context of automata theory.

5.1 Fair Simulation

The work of Tom Henzinger we summarize here is [59]. The main contribution of that paper is a definition of *fair simulation*—a notion of simulation under fairness constraints—that can be computed in polynomial time. The paper presents an automata-theoretic algorithm to compute a fair simulation relation and also has applications in automata theory, which we shall describe later.

Determining trace inclusion (i.e., whether the set of traces starting in some state is included in the set of traces starting in another state) for a finite transition system is PSPACE-complete [94]. Analogously, one can define tree inclusion: A state s tree-includes a state t if every tree of observations that can be embedded in the unrolling of the transition system starting in t can also be embedded

in the unrolling of the transition system starting in s. Tree inclusion happens to coincide with similarity.

For liveness properties, one typically only considers the fair behaviors of a transition system. A prominent example is Büchi fairness: an infinite trace is fair if some event (such as reading a certain label) repeats infinitely often.

One main benefit of (ordinary) simulation over trace inclusion is the computation in polynomial time. Fair trace inclusion is a straightforward generalization of trace inclusion. But the corresponding generalization of simulation is not obvious. There have been other proposals how to define fair simulation. Two of them [50,78] are discussed in the paper, but they do not provide a practical computational advantage over fair trace inclusion: computing the fair simulation in [50] is PSPACE-complete [69] and computing the fair simulation in [78] is still NP-complete [62].

The newly suggested definition[1] of fair simulation uses a game-theoretic characterization and is based on a strategy: state t fairly simulates state s if there exists a strategy such that for every fair run $s_0 s_1 \cdots$ emerging from $s_0 = s$ the run $t_0 t_1 \cdots$ emerging from $t_0 = t$ of the same length and induced by the strategy is also fair and we have that t_i fairly simulates s_i for each $i \geq 0$.

The new definition of fair simulation lies strictly between those in [50,78]. For vacuous fairness constraints, all three definitions coincide with simulation. For deterministic systems, all three definitions coincide with fair trace inclusion.

Fair simulation as defined above enjoys the theoretical property that it is monotonic: given a fair simulation, every (ordinary) simulation that is a superset is also a fair simulation. Then it follows that a finite-state strategy suffices, and even a memoryless strategy for Büchi fairness constraints. The authors reduce the problem of computing a fair simulation relation (respectively finding a winning strategy) to the nonemptiness problem of tree automata. The algorithm to check weak (Büchi) or strong (Streett) fairness constraints is polynomial in the size of the transition system, and in the latter case exponential in the size of the Streett constraint.

As an alternative characterization, the authors extend tree inclusion to fair tree inclusion: a computation tree is fair if all infinite paths correspond to fair behaviors. A state t fairly tree-includes a state s if every fair computation tree starting in s can also be started in t.

Yet another characterization of fair simulation is that fair similarity is the coarsest abstraction that preserves equivalence of all formulae in the fair universal fragment (defined by the authors) of the alternation-free μ-calculus [19].

Fair simulation has been applied in multiple contexts. In [61] the authors show that fair *bi*simulation preserves equivalence in the fair (but not necessarily universal) fragment of the alternation-free μ-calculus and in CTL*. Fair simulation can be checked in a compositional framework using assume-guarantee reasoning [60]. More recently, fair simulation has been extended to tree automata and probabilistic automata, both with Büchi acceptance condition [97]. In particular,

[1] The original definition is given for transition systems with labeled states. Here we use an adaptation to labeled transitions.

that approach models Büchi automata coalgebraically. In the next subsection, we describe common applications of fair simulation and other simulation relations in the broader context of automata theory.

5.2 Language Inclusion and Minimization

Simulation relations play a prominent role in automata theory as an efficient way to solve hard problems in practice. In particular, simulation is a sufficient condition for trace inclusion.

One hard problem in automata theory is language inclusion: is the language of automaton $\mathcal{A}$ included in the language of automaton $\mathcal{B}$? A candidate sufficient condition could be that the initial state of $\mathcal{B}$ simulates the initial state of $\mathcal{A}$. However, most automata have an acceptance condition, often represented by a set of accepting states F. In such cases, ordinary simulation of the initial state does not imply language inclusion. In addition, the acceptance condition needs to be taken into account in the definition of the simulation relation.

A second important problem in automata theory is to reduce the size (usually the number of states) of an automaton. Traditionally, such procedures are said to *minimize* the automaton, although many of them do not actually find a minimum result; this is however intended because exact minimization is typically a hard problem itself [65,89,45]. Minimization is motivated because many algorithms operating on automata scale with the automaton size. The common approach to minimization is quotienting: merging states according to an equivalence relation.

Consider the class of finite automata, for which the acceptance criterion is simple. In this case, one can resort to *direct simulation*: for p to directly simulate q, in addition to ordinary simulation we also require that $p \in F$ if $q \in F$. For deterministic finite automata (DFA), the corresponding direct bisimulation quotient coincides with the canonical minimal DFA, which can be found efficiently [63]. For nondeterministic finite automata, the direct simulation can be used for language inclusion and minimization [39].

A three-step algorithm to reduce a Kripke structure to a unique simulation-equivalent minimum is described in [25]. The algorithm first constructs a quotient structure, then eliminates transitions, and finally deletes unreachable states.

One of the most applied automaton classes in the literature is the Büchi automaton. It has its main application in the context of LTL model checking, where an LTL formula is converted to a Büchi automaton. This automaton can get very large, so minimization is an important tool.

The works in [90] and [42] use simulation relations to minimize Büchi automata. The authors of [90] use direct and backward simulation (essentially direct simulation, but swapping the direction of the transitions in the automaton). The authors of [42] use a notion of fair simulation that is stronger than the one in [59] but more efficient to implement.

A systematic study of simulation and bisimulation relations for minimization and language inclusion of Büchi automata is presented in [43]. The authors describe how to compute the direct simulation [39] and the fair simulation by

Henzinger et al. [59] in a unified framework by reduction to a parity game. The computation is more efficient than the previous algorithm in [59]. Fair simulation as defined before can be used for checking language inclusion but cannot be used for minimization (the obtained language may change). The authors propose a *delayed simulation* for the purpose of minimizing Büchi automata.

An alternative minimization algorithm based on fair simulation with some careful handling to preserve the language is proposed in [51]. An extension of fair and delayed simulation to generalized Büchi automata is given in [67].

Further works focus on stronger reductions for nondeterministic Büchi automata, which in the game view means to give more power to the protagonist.

Etessami proposes the extension to *k-simulation* [41]. The protagonist has k tokens that can all advance at the same time and be redistributed at any time. This can be seen as applying the subset construction with sets of size up to k. For a fixed k, computing k-simulations is polynomial. For $k = n$, k-simulation corresponds to trace inclusion. Since k-simulation is not transitive, so the transitive closure must be chosen for quotienting.

Clemente describes two simulation relations [34]. The first relation is *fixed-word simulation*: here the antagonist initially chooses a word. After that, both players need to move according to the next letter in the word, as usual. This gives more power to the protagonist because the suffix is known at any time. The author shows that adding multiple tokens (see k-simulation above) does not add more power. But fixed-word simulation is PSPACE-complete. The second relation is *proxy simulation*, which is a refinement of backward simulation. It can be seen as iteratively re-playing the game in the new quotient automaton until there is no change. Delayed proxy simulation allows for language-preserving quotienting. Computing a proxy simulation is polynomial.

Mayr and Clemente propose further simulation techniques both for transition pruning and state quotienting [81]. For the latter purpose, the authors introduce *k-lookahead simulation*: the antagonist takes k steps, then the protagonist takes up to k steps, and finally the antagonist backtracks the steps that were not taken. The advantage over k-simulation [41] is its efficiency: one only needs to store at most n^2 configurations. Notably, this approach works well to shrink random automata, which typically do not contain structure to exploit. In a recent work, the same authors use combinations of backward and forward trace inclusion and simulation relations for testing language inclusion [35]. The paper also contains a technical overview of different proposals from the literature.

Simulation relations have also been applied to other classes of automata. Tree automata generalize finite automata from words to tree structures. A minimization algorithm for bottom-up tree automata based on two types of simulation relations is proposed in [2]. The first type is *downward simulation*, which generalizes backward simulation for finite automata. The second type is *upward simulation*, which is however not language-preserving. The authors describe a combination that, in the worst case, leads to the same reduction as with downward simulation, but the additional knowledge from the upward simulation often yields better results.

The visibly pushdown automaton (VPA) [8] is a restricted form of a pushdown automaton where the stack access is determined by the input symbol. Srba considers several types of simulation and bisimulation [92]. The standard bisimulation game is used to determine bisimilarity. The problems of computing the respective preorders and equivalences are all EXPTIME-complete, and PSPACE-complete if the stack is a counter. For the purpose of minimization, Heizmann et al. extend bisimulation to visibly pushdown automata for quotient-based minimization [52]. One difficulty of VPA is that, unlike for finite automata, merging states (and hence simulation) is not transitive: given three states q_1, q_2, q_3, if merging q_1 and q_2 or q_2 and q_3 preserves the language, merging all three states may still alter the language.

Acknowledgments

This research was partly supported by DIREC - Digital Research Centre Denmark and the Villum Investigator Grant S4OS.

References

1. Abadi, M., Lamport, L.: The existence of refinement mappings. Theor. Comput. Sci. **82**(2), 253–284 (1991), https://doi.org/10.1016/0304-3975(91)90224-P
2. Abdulla, P.A., Holík, L., Kaati, L., Vojnar, T.: A uniform (bi-)simulation-based framework for reducing tree automata. Electron. Notes Theor. Comput. Sci. **251**, 27–48 (2009), https://doi.org/10.1016/j.entcs.2009.08.026
3. Aceto, L., Ingólfsdóttir, A., Larsen, K.G., Srba, J.: Reactive Systems: Modelling, Specification and Verification. Cambridge University Press (2007)
4. Aceto, L., Ingólfsdóttir, A., Srba, J.: The algorithmics of bisimilarity. In: Advanced Topics in Bisimulation and Coinduction, Cambridge tracts in theoretical computer science, vol. 52, pp. 100–172. Cambridge University Press (2012)
5. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Hybrid Systems. LNCS, vol. 736, pp. 209–229. Springer (1992), https://doi.org/10.1007/3-540-57318-6_30
6. Alur, R., Dill, D.L.: Automata for modeling real-time systems. In: ICALP. LNCS, vol. 443, pp. 322–335. Springer (1990), https://doi.org/10.1007/BFb0032042
7. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994), https://doi.org/10.1016/0304-3975(94)90010-8
8. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC. pp. 202–211. ACM (2004), https://doi.org/10.1145/1007352.1007390
9. Andersen, J.R., Andersen, N., Enevoldsen, S., Hansen, M.M., Larsen, K.G., Olesen, S.R., Srba, J., Wortmann, J.K.: CAAL: concurrency workbench, Aalborg edition. In: ICTAC. LNCS, vol. 9399, pp. 573–582. Springer (2015), https://doi.org/10.1007/978-3-319-25150-9_33
10. Antonik, A., Huth, M., Larsen, K.G., Nyman, U., Wasowski, A.: 20 years of modal and mixed specifications. Bull. EATCS **95**, 94–129 (2008)
11. Antonik, A., Huth, M., Larsen, K.G., Nyman, U., Wasowski, A.: Complexity of decision problems for mixed and modal specifications. In: FOSSACS. LNCS, vol. 4962, pp. 112–126. Springer (2008), https://doi.org/10.1007/978-3-540-78499-9_9

12. Antonik, A., Huth, M., Larsen, K.G., Nyman, U., Wasowski, A.: EXPTIME-complete decision problems for modal and mixed specifications. ENTCS **242**(1), 19–33 (2009), https://doi.org/10.1016/j.entcs.2009.06.011
13. Asarin, E., Maler, O., Pnueli, A.: Symbolic controller synthesis for discrete and timed systems. In: Hybrid Systems. LNCS, vol. 999, pp. 1–20. Springer (1994), https://doi.org/10.1007/3-540-60472-3_1
14. Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)
15. Bauer, S.S., Juhl, L., Larsen, K.G., Srba, J., Legay, A.: A logic for accumulated-weight reasoning on multiweighted modal automata. In: TASE. pp. 77–84. IEEE Computer Society (2012), https://doi.org/10.1109/TASE.2012.9
16. Benes, N., Křetínský, J., Larsen, K.G., Srba, J.: On determinism in modal transition systems. Theor. Comput. Sci. **410**(41), 4026–4043 (2009), https://doi.org/10.1016/j.tcs.2009.06.009
17. Beneš, N., Křetínský, J., Larsen, K.G., Møller, M.H., Srba, J.: Parametric modal transition systems. In: ATVA. LNCS, vol. 6996, pp. 275–289. Springer (2011), https://doi.org/10.1007/978-3-642-24372-1_20
18. Beneš, N., Křetínský, J., Larsen, K.G., Srba, J.: Checking thorough refinement on modal transition systems is EXPTIME-complete. In: ICTAC. LNCS, vol. 5684, pp. 112–126. Springer (2009), https://doi.org/10.1007/978-3-642-03466-4_7
19. Bensalem, S., Bouajjani, A., Loiseaux, C., Sifakis, J.: Property preserving simulations. In: CAV. LNCS, vol. 663, pp. 260–273. Springer (1992), https://doi.org/10.1007/3-540-56496-9_21
20. Bloom, B., Paige, R.: Transformational design and implementation of a new efficient solution to the ready simulation problem. Sci. Comput. Program. **24**(3), 189–220 (1995), https://doi.org/10.1016/0167-6423(95)00003-B
21. Bouajjani, A., Fernandez, J., Halbwachs, N.: Minimal model generation. In: CAV. LNCS, vol. 531, pp. 197–203. Springer (1990), https://doi.org/10.1007/BFb0023733
22. Brinksma, E., Cleaveland, R., Larsen, K.G., Margaria, T., Steffen, B. (eds.): Tools and Algorithms for Construction and Analysis of Systems, First International Workshop, TACAS '95, Aarhus, Denmark, May 19-20, 1995, Proceedings, LNCS, vol. 1019. Springer (1995), https://doi.org/10.1007/3-540-60630-0
23. Bunte, O., Groote, J.F., Keiren, J.J.A., Laveaux, M., Neele, T., de Vink, E.P., Wesselink, W., Wijs, A., Willemse, T.A.C.: The mCRL2 toolset for analysing concurrent systems - improvements in expressivity and usability. In: TACAS. LNCS, vol. 11428, pp. 21–39. Springer (2019), https://doi.org/10.1007/978-3-030-17465-1_2
24. Burkart, O., Caucal, D., Moller, F., Steffen, B.: Verification on infinite structures. In: Handbook of Process Algebra, pp. 545–623. North-Holland / Elsevier (2001), https://doi.org/10.1016/b978-044482830-9/50027-8
25. Bustan, D., Grumberg, O.: Simulation based minimization. In: CADE. LNCS, vol. 1831, pp. 255–270. Springer (2000), https://doi.org/10.1007/10721959_20
26. Calzolai, F., Nicola, R.D., Loreti, M., Tiezzi, F.: TAPAs: A tool for the analysis of process algebras. Trans. Petri Nets Other Model. Concurr. **1**, 54–70 (2008), https://doi.org/10.1007/978-3-540-89287-8_4
27. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: CONCUR. LNCS, vol. 3653, pp. 66–80. Springer (2005), https://doi.org/10.1007/11539452_9
28. Cerans, K.: Decidability of bisimulation equivalences for parallel timer processes. In: CAV. LNCS, vol. 663, pp. 302–315. Springer (1992), https://doi.org/10.1007/3-540-56496-9_24

29. Cerans, K., Godskesen, J.C., Larsen, K.G.: Timed modal specification - theory and tools. In: CAV. LNCS, vol. 697, pp. 253–267. Springer (1993), https://doi.org/10.1007/3-540-56922-7_21
30. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. **16**(5), 1512–1542 (1994), https://doi.org/10.1145/186025.186051
31. Cleaveland, R., Parrow, J., Steffen, B.: The concurrency workbench. In: Automatic Verification Methods for Finite State Systems. LNCS, vol. 407, pp. 24–37. Springer (1989), https://doi.org/10.1007/3-540-52148-8_3
32. Cleaveland, R., Parrow, J., Steffen, B.: The concurrency workbench: A semantics-based tool for the verification of concurrent systems. ACM Trans. Program. Lang. Syst. **15**(1), 36–72 (1993), https://doi.org/10.1145/151646.151648
33. Cleaveland, R., Sims, S.: The NCSU concurrency workbench. In: CAV. LNCS, vol. 1102, pp. 394–397. Springer (1996), https://doi.org/10.1007/3-540-61474-5_87
34. Clemente, L.: Büchi automata can have smaller quotients. In: ICALP. LNCS, vol. 6756, pp. 258–270. Springer (2011), https://doi.org/10.1007/978-3-642-22012-8_20
35. Clemente, L., Mayr, R.: Efficient reduction of nondeterministic automata with application to language inclusion testing. Log. Methods Comput. Sci. **15**(1) (2019), https://doi.org/10.23638/LMCS-15(1:12)2019
36. Dams, D., Gerth, R., Grumberg, O.: Abstract interpretation of reactive systems. ACM Trans. Program. Lang. Syst. **19**(2), 253–291 (1997), https://doi.org/10.1145/244795.244800
37. Dams, D., Grumberg, O.: Abstraction and abstraction refinement. In: Handbook of Model Checking, pp. 385–419. Springer (2018), https://doi.org/10.1007/978-3-319-10575-8_13
38. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: ECDAR: an environment for compositional design and analysis of real time systems. In: ATVA. LNCS, vol. 6252, pp. 365–370. Springer (2010), https://doi.org/10.1007/978-3-642-15643-4_29
39. Dill, D.L., Hu, A.J., Wong-Toi, H.: Checking for language inclusion using simulation preorders. In: CAV. LNCS, vol. 575, pp. 255–265. Springer (1991), https://doi.org/10.1007/3-540-55179-4_25
40. Enevoldsen, S., Larsen, K.G., Mariegaard, A., Srba, J.: Dependency graphs with applications to verification. Int. J. Softw. Tools Technol. Transf. **22**(5), 635–654 (2020), https://doi.org/10.1007/s10009-020-00578-9
41. Etessami, K.: A hierarchy of polynomial-time computable simulations for automata. In: CONCUR. LNCS, vol. 2421, pp. 131–144. Springer (2002), https://doi.org/10.1007/3-540-45694-5_10
42. Etessami, K., Holzmann, G.J.: Optimizing Büchi automata. In: CONCUR. LNCS, vol. 1877, pp. 153–167. Springer (2000), https://doi.org/10.1007/3-540-44618-4_13
43. Etessami, K., Wilke, T., Schuller, R.A.: Fair simulation relations, parity games, and state space reduction for Büchi automata. SIAM J. Comput. **34**(5), 1159–1175 (2005), https://doi.org/10.1137/S0097539703420675
44. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: a toolbox for the construction and analysis of distributed processes. Int. J. Softw. Tools Technol. Transf. **15**(2), 89–107 (2013), https://doi.org/10.1007/s10009-012-0244-z
45. Gauwin, O., Muscholl, A., Raskin, M.: Minimization of visibly pushdown automata is NP-complete. Log. Methods Comput. Sci. **16**(1) (2020), https://doi.org/10.23638/LMCS-16(1:14)2020

46. Gentilini, R., Piazza, C., Policriti, A.: From bisimulation to simulation: Coarsest partition problems. J. Autom. Reason. **31**(1), 73–103 (2003), https://doi.org/10.1023/A:1027328830731
47. Gibson-Robinson, T., Armstrong, P.J., Boulgakov, A., Roscoe, A.W.: FDR3 - A modern refinement checker for CSP. In: TACAS. LNCS, vol. 8413, pp. 187–201. Springer (2014), https://doi.org/10.1007/978-3-642-54862-8_13
48. Girard, A., Julius, A.A., Pappas, G.J.: Approximate simulation relations for hybrid systems. Discret. Event Dyn. Syst. **18**(2), 163–179 (2008), https://doi.org/10.1007/s10626-007-0029-9
49. Girard, A., Pappas, G.J.: Approximation metrics for discrete and continuous systems. IEEE Trans. Autom. Control. **52**(5), 782–798 (2007), https://doi.org/10.1109/TAC.2007.895849
50. Grumberg, O., Long, D.E.: Model checking and modular verification. ACM Trans. Program. Lang. Syst. **16**(3), 843–871 (1994), https://doi.org/10.1145/177492.177725
51. Gurumurthy, S., Bloem, R., Somenzi, F.: Fair simulation minimization. In: CAV. LNCS, vol. 2404, pp. 610–624. Springer (2002), https://doi.org/10.1007/3-540-45657-0_51
52. Heizmann, M., Schilling, C., Tischner, D.: Minimization of visibly pushdown automata using partial Max-SAT. In: TACAS. LNCS, vol. 10205, pp. 461–478 (2017), https://doi.org/10.1007/978-3-662-54577-5_27
53. Henzinger, T.A.: Hybrid automata with finite bisimulations. In: ICALP. LNCS, vol. 944, pp. 324–335. Springer (1995), https://doi.org/10.1007/3-540-60084-1_85
54. Henzinger, T.A.: The theory of hybrid automata. In: LICS. pp. 278–292. IEEE Computer Society (1996), https://doi.org/10.1109/LICS.1996.561342
55. Henzinger, T.A., Ho, P.: HYTECH: the Cornell HYbrid TECHnology tool. In: Hybrid Systems. LNCS, vol. 999, pp. 265–293. Springer (1994), https://doi.org/10.1007/3-540-60472-3_14
56. Henzinger, T.A., Ho, P., Wong-Toi, H.: Algorithmic analysis of nonlinear hybrid systems. IEEE Trans. Autom. Control. **43**(4), 540–554 (1998), https://doi.org/10.1109/9.664156
57. Henzinger, T.A., Kopke, P.W.: State equivalences for rectangular hybrid automata. In: CONCUR. LNCS, vol. 1119, pp. 530–545. Springer (1996), https://doi.org/10.1007/3-540-61604-7_74
58. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? J. Comput. Syst. Sci. **57**(1), 94–124 (1998), https://doi.org/10.1006/jcss.1998.1581
59. Henzinger, T.A., Kupferman, O., Rajamani, S.K.: Fair simulation. Inf. Comput. **173**(1), 64–81 (2002), https://doi.org/10.1006/inco.2001.3085
60. Henzinger, T.A., Qadeer, S., Rajamani, S.K., Tasiran, S.: An assume-guarantee rule for checking simulation. ACM Trans. Program. Lang. Syst. **24**(1), 51–64 (2002), https://doi.org/10.1145/509705.509707
61. Henzinger, T.A., Rajamani, S.K.: Fair bisimulation. In: TACAS. LNCS, vol. 1785, pp. 299–314. Springer (2000), https://doi.org/10.1007/3-540-46419-0_21
62. Hojati, R.: A BDD-Based Environment for Formal Verification of Hardware Systems. Ph.D. thesis, EECS Department, University of California, Berkeley (Jul 1996), http://www2.eecs.berkeley.edu/Pubs/TechRpts/1996/3052.html
63. Hopcroft, J.E.: An n log n algorithm for minimizing states in a finite automaton. In: Theory of Machines and Computations, pp. 189–196. Academic Press (1971), https://doi.org/10.1016/B978-0-12-417750-5.50022-1

64. Jančar, P., Srba, J.: Undecidability of bisimilarity by defender's forcing. J. ACM **55**(1), 1–26 (2008), https://doi.org/10.1145/1326554.1326559
65. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM J. Comput. **22**(6), 1117–1141 (1993), https://doi.org/10.1137/0222067
66. Julius, A.A., D'Innocenzo, A., Benedetto, M.D.D., Pappas, G.J.: Approximate equivalence and synchronization of metric transition systems. Syst. Control. Lett. **58**(2), 94–101 (2009), https://doi.org/10.1016/j.sysconle.2008.09.001
67. Juvekar, S., Piterman, N.: Minimizing generalized Büchi automata. In: CAV. LNCS, vol. 4144, pp. 45–58. Springer (2006), https://doi.org/10.1007/11817963_7
68. Kucera, A., Mayr, R.: Why is simulation harder than bisimulation? In: CONCUR. LNCS, vol. 2421, pp. 594–610. Springer (2002), https://doi.org/10.1007/3-540-45694-5_39
69. Kupferman, O., Vardi, M.Y.: Verification of fair transition systems. Chic. J. Theor. Comput. Sci. **1998** (1998), http://cjtcs.cs.uchicago.edu/articles/1998/2/contents.html
70. Kučera, A., Jančar, P.: Equivalence-checking with infinite-state systems: Techniques and results. In: SOFSEM. LNCS, vol. 2540, pp. 41–73. Springer (2002), https://doi.org/10.1007/3-540-36137-5_3
71. Lanotte, R., Tini, S.: Taylor approximation for hybrid systems. Inf. Comput. **205**(11), 1575–1607 (2007), https://doi.org/10.1016/j.ic.2007.05.004
72. Laroussinie, F., Larsen, K.G., Weise, C.: From timed automata to logic - and back. In: MFCS. LNCS, vol. 969, pp. 529–539. Springer (1995), https://doi.org/10.1007/3-540-60246-1_158
73. Larsen, K.G., Nyman, U., Wasowski, A.: On modal refinement and consistency. In: CONCUR. LNCS, vol. 4703, pp. 105–119. Springer (2007), https://doi.org/10.1007/978-3-540-74407-8_8
74. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Int. J. Softw. Tools Technol. Transf. **1**(1-2), 134–152 (1997), https://doi.org/10.1007/s100090050010
75. Larsen, K.G., Thomsen, B.: A modal process logic. In: LICS. pp. 203–210. IEEE Computer Society (1988), https://doi.org/10.1109/LICS.1988.5119
76. Larsen, K.G., Yi, W.: Time abstracted bisimulation: Implicit specifications and decidability. In: MFPS. LNCS, vol. 802, pp. 160–176. Springer (1993), https://doi.org/10.1007/3-540-58027-1_8
77. Lee, D., Yannakakis, M.: Online minimization of transition systems (extended abstract). In: STOC. pp. 264–274. ACM (1992), https://doi.org/10.1145/129712.129738
78. Lynch, N.A., Tuttle, M.R.: Hierarchical correctness proofs for distributed algorithms. In: PODC. pp. 137–151. ACM (1987), https://doi.org/10.1145/41840.41852
79. Majumdar, R., Zamani, M.: Approximately bisimilar symbolic models for digital control systems. In: CAV. LNCS, vol. 7358, pp. 362–377. Springer (2012), https://doi.org/10.1007/978-3-642-31424-7_28
80. Mayr, R.: Process rewrite systems. Inf. Comput. **156**(1-2), 264–286 (2000), https://doi.org/10.1006/inco.1999.2826
81. Mayr, R., Clemente, L.: Advanced automata minimization. In: POPL. pp. 63–74. ACM (2013), https://doi.org/10.1145/2429069.2429079
82. Mazala, R.: Infinite games. In: Automata, Logics, and Infinite Games: A Guide to Current Research. LNCS, vol. 2500, pp. 23–42. Springer (2001), https://doi.org/10.1007/3-540-36387-4_2
83. Milner, R.: An algebraic definition of simulation between programs. In: IJCAI. pp. 481–489 (1971), http://ijcai.org/Proceedings/71/Papers/044.pdf

84. Moller, F.: Infinite results. In: CONCUR. LNCS, vol. 1119, pp. 195–216. Springer (1996), https://doi.org/10.1007/3-540-61604-7_56
85. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM J. Comput. **16**(6), 973–989 (1987), https://doi.org/10.1137/0216062
86. Park, D.M.R.: Concurrency and automata on infinite sequences. In: Theoretical Computer Science. LNCS, vol. 104, pp. 167–183. Springer (1981), https://doi.org/10.1007/BFb0017309
87. Pola, G., Girard, A., Tabuada, P.: Approximately bisimilar symbolic models for nonlinear control systems. Autom. **44**(10), 2508–2516 (2008), https://doi.org/10.1016/j.automatica.2008.02.021
88. Rauch Henzinger, M., Henzinger, T.A., Kopke, P.W.: Computing simulations on finite and infinite graphs. In: FOCS. pp. 453–462. IEEE Computer Society (1995), https://doi.org/10.1109/SFCS.1995.492576
89. Schewe, S.: Beyond hyper-minimisation—minimising DBAs and DPAs is NP-complete. In: FSTTCS. LIPIcs, vol. 8, pp. 400–411. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2010), https://doi.org/10.4230/LIPIcs.FSTTCS.2010.400
90. Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In: CAV. LNCS, vol. 1855, pp. 248–263. Springer (2000), https://doi.org/10.1007/10722167_21
91. Srba, J.: Roadmap of infinite results. In: Current Trends in Theoretical Computer Science: The Challenge of the New Century, vol. 2, pp. 337–350. World Scientific (2004)
92. Srba, J.: Beyond language equivalence on visibly pushdown automata. Log. Methods Comput. Sci. **5**(1) (2009), http://arxiv.org/abs/0901.2068
93. Stirling, C.: Local model checking games. In: CONCUR. LNCS, vol. 962, pp. 1–11. Springer (1995), https://doi.org/10.1007/3-540-60218-6_1
94. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time. In: STOC. pp. 1–9. ACM (1973), https://doi.org/10.1145/800125.804029
95. Thomas, W.: On the Ehrenfeucht-Fraïssé game in theoretical computer science. In: TAPSOFT. LNCS, vol. 668, pp. 559–568. Springer (1993), https://doi.org/10.1007/3-540-56610-4_89
96. Tiwari, A.: Abstractions for hybrid systems. Formal Methods Syst. Des. **32**(1), 57–83 (2008), https://doi.org/10.1007/s10703-007-0044-3
97. Urabe, N., Hasuo, I.: Fair simulation for nondeterministic and probabilistic Büchi automata: a coalgebraic perspective. Log. Methods Comput. Sci. **13**(3) (2017), https://doi.org/10.23638/LMCS-13(3:20)2017
98. Yi, W.: CCS + time = an interleaving model for real time systems. In: ICALP. LNCS, vol. 510, pp. 217–228. Springer (1991), https://doi.org/10.1007/3-540-54233-7_136