Model Checking Systems and Specifications with Parameterized Atomic Propositions

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. In classical LTL model checking, both the system and the specification are over a finite set of atomic propositions. We present a natural extension of this model, in which the atomic propositions are parameterized by variables ranging over some (possibly infinite) domain. For example, by parameterizing the atomic propositions $send$ and $receive$ by a variable x ranging over possible messages, the specification $\mathsf{G}(send.x \rightarrow \mathsf{F}receive.x)$ specifies that not only each send signal is followed by a receive signal, but also that the content of the received message agrees with the content of the one sent.

Our extended setting consists of *Variable LTL* (VLTL) – a specification formalism that extends LTL with atomic propositions parameterized by variables, and *abstract systems* – systems in which atomic propositions may be parameterized by variables. We study the model-checking problem in this setting. We show that while the general setting is undecidable, some useful special cases are decidable. In particular, for fragments of VLTL that restrict the quantification over the variables, the model checking is PSPACE-complete, and thus is not harder than the LTL model checking problem. The latter result conveys the strength and advantage of our setting.

1 Introduction

In model checking, we verify that a system has a desired behavior by checking that a mathematical model of the system satisfies a formal specification of the desired behavior. Traditionally, the system is modeled by a Kripke structure – a finite-state system whose states are labeled by a finite set of atomic propositions. The specification is a temporal-logic formula over the same set of atomic propositions [4].

The complexity of model checking depends on the sizes of the system and the specification [15,12]. One source for these sizes being huge is a large or even infinite data domain over which the system, and often also the specification, need to be defined. Another source for the large sizes are systems composed of many components, such as underlying processes or communication channels, whose number may not be known in advance. In both cases, desired specifications might be inexpressible and model checking intractable.

In this work we propose a novel approach for model checking systems and specifications that suffer from the size problem described above. We do so by extending both the specification formalism and the system model with atomic propositions that are parameterized by variables ranging over some (possibly infinite) domain.

Let us start with a short description of the specification formalism. We introduce and study the linear temporal logic *Variable LTL* (VLTL, for short). VLTL has the syntax of LTL except that atomic propositions may be parameterized with variables over some finite or infinite domain. The variables can be quantified and assignments to them can be constrained by a set of inequalities. VLTL formulas are interpreted with respect to all assignments to the variables that respect the inequalities. For example, the VLTL formula $\psi = \forall x.\mathsf{G}(send.x \rightarrow \mathsf{F}receive.x)$ states that whenever a message with content d, taken from the domain, is sent, then a message with content d is eventually received. Note that if the domain of messages is infinite or unknown in advance, then ψ does not have an equivalent LTL formula.

In order to see the usefulness of VLTL, consider the LTL specification $\mathsf{G}(req \rightarrow \mathsf{F}grant)$, stating that every request is eventually granted. We may wish to parameterize the req and $grant$ by atomic propositions with the id of the process that invokes the request. In LTL this can only be done by having a new set of atomic propositions $req_1, grant_1, \ldots, req_n, grant_n$, where n is the number of processes. This both blows-up the specification and requires the specifier to know the number of processes in advance. Instead, in VLTL we parameterize the atomic propositions by a variable x ranging over the ids of the processes. Since it is natural to require that the property holds for *every* assignment to x, we quantify it universally, thus obtaining the formula $\psi = \forall x; \mathsf{G}(req.x \rightarrow \mathsf{F}grant.x)$.[1] Note that the negation of ψ quantifies x existentially, thus $\neg\psi = \exists x; \mathsf{F}(req.x \wedge \mathsf{G}\neg grant.x)$. Beyond the use of existential quantification for identifying counterexamples as above, such quantification is useful by itself. For example, the formula $\exists x.\mathsf{GF}\neg idle.x$ states that in each computation, there exists at least one process that is not idle infinitely often.

Next, consider the formula $\theta = \forall x_1; \forall x_2; \mathsf{G}((\neg send.x_2)\,\mathsf{U}send.x_1) \rightarrow ((\neg rec.x_2)\,\mathsf{U}rec.x_1)$, stating that messages are received in the order in which they are sent. To convey this, the values assigned to x_1 and x_2 should be different. This is handled by the set of inequalities that VLTL formulas include. When interpreting a formula, we consider only assignments that respect the set of inequalities. In the example above, the VLTL formula $\langle \theta, x_1 \neq x_2 \rangle$ specifies that θ holds for every assignment in which the variables x_1 and x_2 are assigned different values.

As another example, consider a system with transactions interleaved in a single computation [17]. Each transaction has an id, yet the range of ids is not known in advance. We may wish to refer to the sequence of events associated with one transaction. For instance, when x stands for a transaction id, then $\forall x; \mathsf{G}(req.x \rightarrow \mathsf{F}grant.x)$ states that whenever a request is raised in a transaction, it is eventually granted in the same transaction. Alternatively, we may wish to specify that a request is granted only in a different transaction. In this case we may write $\langle \varphi, x_1 \neq x_2 \rangle$, where $\varphi = \forall x_1; \forall x_2; \mathsf{G}(req.x_1 \rightarrow ((\neg grant.x_1)\,\mathsf{U}grant.x_2))$.

Thus, as demonstrated above, VLTL formulas are able to compactly express specifications over a large, possibly infinite domain, which would otherwise be inexpressible by LTL or lead to extremely large formulas. Moreover, VLTL is able to express

[1] Note that unlike the standard way of augmenting temporal logic with quantification [13,16], here the variables do not range over the set of atomic propositions but rather over the domain of their parameters.

properties for domains whose size is unknown in advance (e.g., number of different messages, number of different channels).

We now turn to describe in detail the way we model systems. We distinguish between *concrete* models, in which atomic propositions are parameterized with values from the domain, and *abstract* models, in which the atomic propositions are parameterized with variables. For instance, in a concrete model, the proposition $send.3$ stands for the atomic proposition $send$ that carries the value 3. In an abstract model, the proposition $send.x$ indicates that $send$ carries the value assigned to x. Assignments to variables are fixed along a computation, except when the system resets their value. More precisely, in every transition of the system, a subset of the variables is reset, meaning that these variables can change their value in the next step of the run[2]. Also, as with VLTL, the model includes a set of inequalities over the variables.

The concrete computations of an abstract system are obtained from system paths by assigning values to the occurrences of the variables in a way that respects the set of inequalities, and is consistent with the resetting along the path. That is, the value assigned to a variable may change only when a transition in which the variable is reset is taken. Note that a path of an abstract system may induce infinitely many different concrete computations of the abstract system, formed by different assignments to the occurrences of the variables. Thus, abstract systems offer a compact and simple representation of infinite-state systems in which the control is finite and the source of infinity is data that may be infinite or unknown in advance.

As a simple example, consider the abstract system in Figure 1. It describes a simple stop-and-wait communication protocol. Once a message x is sent, the system waits for an ack with identical content, confirming the arrival of the message. When this happens, the content of the message is reset and a new cycle starts. If a timeout occurs, the same message is sent. Note that if the message domain is infinite, standard finite systems cannot describe this protocol, as it involves a comparison of the content of the message in different steps of the protocol. More complex communication protocols, in which several send-receive processes can run simulateously (e.g., sliding windows), can be modeled by an abstract system with several variables, one for every process.

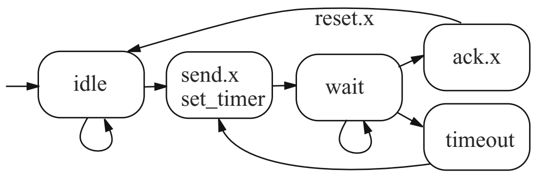

Fig. 1. A simple abstract system for the stop and wait protocol

We study the model-checking problem for VLTL with respect to concrete and abstract systems. We also consider two natural fragments of VLTL: $\forall$-VLTL and $\exists$-VLTL, containing only $\forall$ and $\exists$ quantifiers, respectively.

[2] Despite some similarity in their general description and use of resets, abstract systems have no direct relation to timed automata [22]. In abstract systems, the variables and resets are over domain values rather than clocks ranging over the reals.

We start with VLTL model checking of concrete systems. We show that given a concrete system and a VLTL formula, it is possible to reduce the infinite domain to a finite one, and reduce the problem to LTL model checking. The reduction involves an LTL formula that is exponential in the VLTL formula, and we prove that the problem is indeed EXPSPACE-complete. Hardness in EXPSPACE applies already for the fragment of $\exists$-VLTL. As good news, for the fragment of $\forall$-VLTL, a similar procedure runs in PSPACE, and the problem is PSPACE-complete in the size of the formula, as for LTL. We also consider the model-checking for a single concrete computation and show that it is PSPACE-complete.

We proceed to show that for abstract systems the model-checking problem is in general undecidable. Here too, undecidability applies already for the fragment of $\exists$-VLTL. Our undecidability proof is by a reduction from Post's Correspondence Problem, following the lines of the reduction in [14], showing that the universality problem for register automata is undecidable. On the other hand, for systems with no resetting, model checking is EXPSPACE-complete, and thus is not harder than model checking of concrete systems. Moreover, for the fragment of $\forall$-VLTL, the model-checking problem for abstract systems is PSPACE-complete, and thus is not harder than the LTL model-checking problem. This latter result conveys the strength and advantage of our model: abstract systems are able to accurately describe infinite state systems; $\forall$-VLTL formulas are able to express rich behaviors that are inexpressible by LTL, and may refer to infinitely many values. Yet, surprisingly, model checking in that setting is not harder than that of LTL.

Related Work. Researchers have studied several classes of infinite-state systems, generally differing in the source of infinity. For systems with a finite control and infinite (or very large) data, researchers have developed heuristics such as data abstraction [3,6]. Data abstractions are applied in order to construct a finite model and a simplified specification, and are based on mapping the large data domain into a finite and small number of classes. Our approach, on the other hand, has the abstraction built in the system, and is also part of the specification formalism. Finite control is used also in work on hybrid systems [10] and systems with an unbounded memory (e.g., pushdown systems [7]), where the underlying setting is very different from the one studied here. Closer to our abstract systems are some of the approaches to handle systems having an unbounded number of components (e.g., parameterized systems [8]). The solutions in these cases are tailored for the setting in which the parameter is the number of components, and cannot be applied, say, for parameterizing data.

Several extensions of LTL for handling infinite-state or parameterized systems have been suggested and studied. A general such extension, with a setting similar to that presented in this paper, is *first-order LTL*. First-order temporal logics are used for specifying and reasoning about temporal databases [19]. The work focuses on finding decidable fragments of the logic, its expressive power and axiomatization (c.f., [21]).

In [2,8], the authors studied an extension of temporal logic in which atomic propositions are parameterized with a variable that ranges over the set of processes ids. In [20], the authors study a restricted fragment of first-order temporal logic that is suitable for verifying parameterized systems. Again, these works are tailored for the setting of parameterized systems, and are also restricted to systems in which (almost) all components are identical. VLTL is much richer, already in the context of parameterized systems. For example, it can refer to both sender id and message content. In [5], the authors introduced Constraint LTL, in which atomic propositions may be constraints like $x < y$, and formulas are interpreted over sequences of assignments of variables to values in $\mathbb{N}$ or $\mathbb{Z}$. Thus, like our approach, Constraint LTL enables the specification to capture all assignments to the variable. Unlike our approach, the domain is restricted to numerical values. In [11], the author extends LTL with the freeze quantifier, which is used for storing values from an infinite domain in a register. As discussed in our earlier work [9], the variable-based formalism is cleaner than the register-based one, and it naturally extends traditional LTL. In addition, our extension allows variables in both the system and the specification. Finally, unlike existing work, as well as work on first order LTL, in our setting a parameterized atomic proposition may hold with several different parameter values in the same point in time.

In [9], we introduced VNFAs – nondeterministic finite automata augmented by variables. VNFAs can recognize languages over infinite alphabets. As in VLTL, the idea is simple and is based on labeling some of the transitions of the automaton by variables that range over some infinite domain. In [1], a similar use of variables was studied for finite alphabets, allowing a compact representation of regular expressions. There, the authors considered two semantics; in the "possibility" semantics, the language of an expression over variables is the union of all regular languages obtained by assignments to the variables, and in the "certainty" semantics, the language is the intersection of all such languages. Note that in VLTL, it is possible to mix universal and existential quantification of variables.

2 VLTL: LTL with Variables

Linear temporal logic (LTL) is a formalism for specifying on-going behaviors of reactive systems. We model a finite-state system by a *Kripke structure* $\mathcal{K} = \langle P, S, I, R, L\rangle$, where P is a set of atomic propositions, S is a finite set of states, $I \subseteq S$ is a set of initial states, $R \subseteq S \times S$ is a total transitions relation, and $L : S \to 2^P$ is a labeling function. We then say that $\mathcal{K}$ *is over* P. A *path* in $\mathcal{K}$ is an infinite sequence $s_0, s_1, s_2 \ldots$ of states such that $s_0 \in I$ and $\langle s_i, s_{i+1}\rangle \in R$ for every $i \geq 0$. A *computation* of $\mathcal{K}$ is an infinite word $\pi = \pi_0\pi_1 \ldots$ over 2^P such that there exists a path $s_0, s_1, s_2 \ldots$ in $\mathcal{K}$ with $\pi_i = L(s_i)$ for all $i \geq 0$. We denote by π^i the suffix $\pi_i\pi_{i+1}, \ldots$ of π.

We assume that the reader is familiar with the syntax and semantics of LTL. For a Kripke structure $\mathcal{K}$ and an LTL formula φ, we say that $\mathcal{K}$ satisfies φ, denoted $\mathcal{K} \models \varphi$, if for every computation π of $\mathcal{K}$, it holds that $\pi \models \varphi$. The *model-checking problem* is to decide, given $\mathcal{K}$ and φ, whether $\mathcal{K} \models \varphi$.

The logic VLTL extends LTL by allowing some of the atomic propositions to be parameterized by variables that can take values from a finite or infinite domain. Consider a finite set P of atomic propositions, a finite or infinite domain $\mathcal{D}$, a finite set T of *parameterized atomic propositions*, and a finite set X of *variables*. In VLTL, the propositions in T are parameterized by variables in X that range over $\mathcal{D}$. A VLTL formula also contains guards, in the form of inequalities over X, which restrict the set of possible assignments to the variables.

We now define VLTL formally. An *unrestricted VLTL formula* over P, T, and X is a formula ψ of the form $Q_1x_1; Q_2x_2; \cdots Q_kx_k; \theta$, where $Q_i \in \{\exists, \forall\}$, $x_i \in X$, and θ is an LTL formula over $P \cup T \cup (T \times X)$. Variables that appear in θ and are not quantified are *free*. If θ has no free variables, then ψ is *closed*. We say that a parameterized atomic proposition $a \in T$ is *primitive in* θ if there is an occurrence of a in θ that is not parameterized by a variable. An *inequality set over* X is a set $E \subseteq \{x_i \neq x_j \mid x_i, x_j \in X\}$.

A *VLTL formula* over P, T, and X is a pair $\varphi = \langle \psi, E \rangle$ where ψ is an unrestricted VLTL formula over P, T, and X, and E is an inequality set over X. The notions of closed formulas, and of a primitive parametric atomic proposition are lifted from ψ to φ. That is, φ is closed if ψ is closed, and a is primitive in φ if it is primitive in ψ.

We consider two fragments of VLTL. The logic $\exists$-VLTL is the fragment of VLTL in which $Q_i = \exists$ for all $1 \leq i \leq k$. Dually, $\forall$-VLTL is the fragment of VLTL in which only the $\forall$ quantifier is allowed.

We define the semantics for VLTL with respect to both concrete computations – infinite words over the alphabet $2^{P \cup (T \times \mathcal{D})}$, and abstract computations, defined later in this section. A position in a concrete computation is a letter $\sigma \in 2^{P \cup (T \times \mathcal{D})}$. For $a \in T$, we say that σ satisfies a, denoted $\sigma \models a$, if there exists $d \in \mathcal{D}$ such that $a.d \in \sigma$. Thus, a primitive $a \in T$ is satisfied by assignments in which a holds with at least one value.

We say that a partial function $f : X \to \mathcal{D}$ *respects* E if for every $x_i \neq x_j$ in E, it holds that $f(x_i) \neq f(x_j)$.

Consider a concrete computation π, a VLTL formula $\varphi = \langle \psi, E \rangle$, with $\psi = Q_1x_1; Q_2x_2; \ldots; Q_nx_n; \theta$, and a partial function $f : X \to \mathcal{D}$ that assigns values to all the free variables in ψ and respects E. We use $\pi \models_f \psi$ to denote that π satisfies ψ under the function f. The satisfaction relation is defined as follows.

- If $n = 0$ (that is, $\psi = \theta$ has no quantification and all its variables are free), then the formula ψ_f obtained from ψ by replacing, for every x, every occurrence of $a.x$ with $a.f(x)$, is an LTL formula over $P \cup T \cup (T \times \mathcal{D})$. Then, $\pi \models_f \varphi$ iff f respects E and $\pi \models \psi_f$.
- If $Q_1 = \exists$ (that is, $\psi = \exists x_1; Q_2x_2; \ldots; Q_nx_n; \theta$), then $\pi \models_f \varphi$ iff there exists $d \in \mathcal{D}$ such that $f[x_1 \leftarrow d]$ respects E and $\pi \models_{f[x_1 \leftarrow d]} Q_2x_2; \ldots; Q_nx_n; \theta$.
- If $Q_1 = \forall$ (that is, $\psi = \forall x_1; Q_2x_2; \ldots; Q_nx_n; \theta$), then $\pi \models_f \varphi$ iff for all $d \in \mathcal{D}$ such that $f[x_1 \leftarrow d]$ respects E, we have that $\pi \models_{f[x_1 \leftarrow d]} Q_2x_2; \ldots; Q_nx_n; \theta$.

Example 1. Consider the concrete computation

$$\pi = \{send.1\}\{send.2\}\{rec.2\}\{rec.1\}^\omega$$

and the VLTL formula $\langle \exists x; \theta, \emptyset \rangle$, where $\theta = \mathsf{G}(send.x \to \mathsf{X}rec.x)$. Then for the function $f(x) = 2$, we have that $\theta_f = \mathsf{G}(send.2 \to \mathsf{X}rec.2)$, and since $\pi \models \theta_f$, it holds that $\pi \models \langle \exists x; \theta, \emptyset \rangle$.

Next, consider the VLTL formula $\langle \forall x; \theta, \emptyset \rangle$. Then for the function $g(x) = 1$, we have that $\pi \not\models \theta_g$, and therefore $\pi \not\models \langle \forall x; \theta, \emptyset \rangle$.

Similarly to first order logic, when the formula ψ is closed, the satisfaction of E does not depend on the function f, and we use the notation $\models$.

The extension of the definition to concrete systems (rather than computations) is similar to that of LTL: For a Kripke structure $\mathcal{K}$ over $P \cup (T \times \mathcal{D})$ and a closed VLTL formula $\varphi = \langle \psi, E \rangle$, we say that $\mathcal{K}$ satisfies φ, denoted $\mathcal{K} \models \varphi$, if for every computation π of $\mathcal{K}$, it holds that $\pi \models \langle \psi, E \rangle$.

We now define abstract systems, whose computations may contain infinitely many values. An *abstract system* over P, $T \cup \{reset\}$ and X is a pair $\langle \mathcal{K}, E \rangle$, where $\mathcal{K}$ is a Kripke structure over $P \cup (T \times X)$ with a labeling $L' : R \to 2^{\{reset\} \times X}$ for the transitions, and E is an inequality set over X. Intuitively, a system variable is assigned values throughout a computation of the system. If the computation traverses a transition labeled by $reset.x$, then x can be assigned a new value, that will remain unchanged at least until the next traversal of $reset.x$. Also, if $x_i \neq x_j \in E$, then the values that are assigned to x_i and x_j in a given point in the computation must be different.

An *abstract computation* of $\langle \mathcal{K}, E \rangle$ is a pair $\langle \rho, E \rangle$, where ρ is an infinite word $\rho_0 \rho'_0 \rho_1 \rho'_1 \cdots$ over $P \cup ((T \cup \{reset\}) \times X)$ such that there exists a path $s_0, s_1, \ldots$ in $\mathcal{K}$ such that for every $i \geq 0$, it holds that $L(s_i) = \rho_i$, and $L'(\langle s_i, s_{i+1} \rangle) = \rho'_i$.

A concrete computation is obtained from an abstract computation by assigning values from $\mathcal{D}$ to the variables in X as described next.

Consider an abstract computation $\langle \rho, E \rangle$ over P, $T \cup \{reset\}$ and X, where $\rho = \rho_0 \rho'_0 \rho_1 \rho'_1 \cdots$ and a concrete computation $\pi = \pi_0, \pi_1, \ldots$ over P, T and $\mathcal{D}$. We say that π is a *concretization of* $\langle \rho, E \rangle$ if π is obtained from $\rho_0 \rho_1 \rho_2 \cdots$ by substituting every occurrence of $x \in X$ by a value in $\mathcal{D}$, according to the following rules.

- For every two consecutive occurrences of $reset.x$ in ρ'_i and ρ'_j, the values assigned to occurrences of x in $\rho_{i+1}(x), \rho_{i+2}(x) \ldots \rho_j(x)$ are identical.
- For every $x_i \neq x_j \in E$, for every position ρ_k that contains occurrences of x_i and x_j, these occurrences are assigned different values in π_i.

If π is a concretization of $\langle \rho, E \rangle$, then we say that $\langle \rho, E \rangle$ is an *abstraction* of π. Note that $\langle \rho, E \rangle$ may have infinitely many concretizations.

Notice that for a domain $\mathcal{D}$, an abstract system $\langle \mathcal{K}, E \rangle$ represents a (possibly infinite) concrete system over $P \cup (T \times \mathcal{D})$ whose computations are exactly all concretizations (w.r.t. $\mathcal{D}$) of every abstract computation of $\langle \mathcal{K}, E \rangle$.

Example 2. Consider the abstract system $\langle \mathcal{K}, \emptyset \rangle$, where $\mathcal{K}$, appearing in Figure 2, describes a protocol for two processes, a and b, that use a printer that may print a single job at a time. A token is passed around. The atomic propositions ta and tb are valid when processes a and b hold the token, respectively. The parameterized atomic propositions ra, rb, and p are parameterized by x_1 and x_2, which can get values from the range of file names. The proposition $ra.x_1$ is valid when process a requests to print x_1, and similarly for $rb.x_2$ and process b. Once a file is printed, the variable that maintains the file is reset.

Consider the path $s_1 s_2 (s_3 s_4 s_5 s_6)^\omega$ of $\mathcal{K}$. It induces the abstract computation $\langle \rho, \emptyset \rangle$, where $\rho =$

$$\{ta\}\emptyset\{ta, ra.x_1\}\emptyset(\{ta, ra.x_1, rb.x_2\}\emptyset\{p.x_1, rb.x_2\}\{reset.x_1\}$$
$$\{tb, ra.x_1, rb.x_2\}\emptyset\{p.x_2, ra.x_1\}\{reset.x_2\})^\omega$$

An example for a concretization of $\langle \rho, \emptyset \rangle$ is

$$\{ta\}\{ta, ra.doc1\}\{ta, ra.doc1, rb.data.txt\}\{p.doc1, rb.data.txt\}$$
$$\{tb, ra.doc2, rb.data.txt\}\{p.data.txt, ra.doc2\}\{ta, ra.doc2, rb.vltl.pdf\}\ldots$$

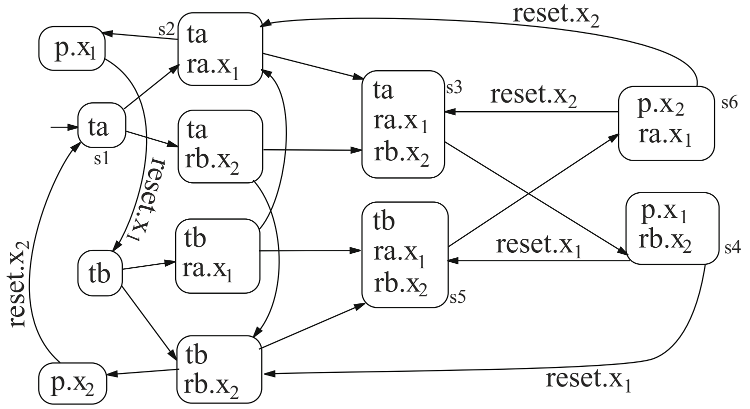

Fig. 2. The abstract system $\mathcal{K}$

We now describe the second semantics of VLTL, for abstract computations. Consider a set X' of variables, an abstract computation $\langle \rho, E' \rangle$ over $P \cup ((T \cup \{reset\}) \times X')$, and a closed VLTL formula $\varphi = \langle \psi, E \rangle$ over $P \cup T \cup (T \times X)$.[3] We say that $\langle \rho, E' \rangle$ satisfies φ, denoted $\langle \rho, E' \rangle \models \varphi$, if for every concretization π of $\langle \rho, E' \rangle$, it holds that $\pi \models \varphi$. Note that ρ and ψ are defined over different sets of variables, that are not related to each other.

Example 3. Consider the abstract system $\langle \mathcal{K}, \emptyset \rangle$ and the abstract computation $\langle \rho, \emptyset \rangle$ of Example 2.

Consider the formula $\varphi = \langle \forall z_1; \forall z_2; \mathsf{G}((ra.z_1 \wedge ta) \rightarrow ((\neg p.z_2) \mathsf{U} p.z_1)), \{z_1 \neq z_2\} \rangle$ over P, T, and $X = \{z_1, z_2\}$. In every concretization π of $\langle \rho, \emptyset \rangle$, whenever process a requests to print a document d and holds the token, then the next job that is printed is d, and no other job d' gets printed before that. This holds for all values d and d' such that $d \neq d'$, and therefore, $\langle \rho, \emptyset \rangle \models \varphi$.

For an abstract system $\langle \mathcal{K}, E' \rangle$ and a closed VLTL formula $\langle \psi, E \rangle$, we say that $\langle \mathcal{K}, E' \rangle$ satisfies $\langle \psi, E \rangle$, denoted $\langle \mathcal{K}, E' \rangle \models \langle \psi, E \rangle$, if for every abstract computation $\langle \rho, E' \rangle$ of $\mathcal{K}$, it holds that $\langle \rho, E' \rangle \models \langle \psi, E \rangle$.

3 Model Checking of Concrete Systems

In this section we present a model-checking algorithm for finite concrete systems and discuss the complexity of the model-checking problem for the fragments of VLTL. We show that the general case is EXPSPACE-complete, but is PSPACE-complete for the fragment of $\forall$-VLTL and for single computations. Thus, the problem for these latter cases is not more complex than LTL model checking.

[3] An abstract computation represents infinitely many concrete computations. For every such computation, a different function may be needed in order to satisfy the formula. Therefore, the definition does not involve a specific function from the variables to the values, and so only closed formulas are considered.

The model-checking procedure we present for concrete systems reduces the model-checking problem for VLTL to the model-checking problem for LTL. The key to this procedure is the observation that different values that do not appear in a given computation behave similarly when assigned to a formula variable. Thus, it is sufficient to consider the finite set of values that do appear in the concrete system $\mathcal{K}$, plus one additional value for every quantifier to represent the values that do not appear in $\mathcal{K}$. This means that in case of a very large, or even infinite, domain, we can check the set of assignments over a finite domain instead, resulting in a finite procedure.

We say that a concrete computation π and a VLTL formula φ *do not distinguish between two values* $d_1, d_2 \in \mathcal{D}$ *with respect to the variable* x *and a partial function* $f : X \to \mathcal{D}$ if $\pi \models_{f[x \leftarrow d_1]} \varphi$ iff $\pi \models_{f[x \leftarrow d_2]} \varphi$.

Lemma 1. *Let π be a concrete computation and let φ be a VLTL formula over P, T and X. Then, for every $x \in X$, every function $f : X \to \mathcal{D}$, and every two values d_1 and d_2 that do not appear in π, it holds that π and φ do not distinguish between d_1 and d_2 with respect to x and f.*

We now use Lemma 1 in order to reduce the VLTL model-checking problem for concrete systems to the LTL model-checking problem. We describe two algorithms. The first algorithm, ModelCheck, gets as input a single computation and a VLTL formula and decides whether the computation satisfies the formula. The second is based on a transformation of a given VLTL formula to an LTL formula such that the system satisfies the VLTL formula iff it satisfies the LTL formula. The idea behind both algorithms is the same – an inductive valuation of the formula, either (the first algorithm) by recursively trying possible assignments to the variables, or (the second algorithm) by encoding the various possible assignments using conjunctions and disjunctions in LTL. In both cases, Lemma 1 implies that the number of calls needed in the recursion or the number of conjuncts or disjuncts in the translation is finite.

Let us describe the first algorithm in more detail. Consider a computation π in the concrete system. Recall that such a system is a finite state system, and therefore π contains finitely many values from $\mathcal{D}$. Let A be the set of values that appear in π. Consider a VLTL formula $\varphi = \langle \psi, E \rangle$ and a partial function $f : X \to \mathcal{D}$ that respects E. Let $B = Image(f)$.

Intuitively, each recursive call of the algorithm evaluates φ with a different assignment to the variables. Lemma 1 enables checking assignments over a finite set of values instead of the entire domain. For every quantifier, a new value is added to this set, initially assigned $A \cup B$. According to Lemma 1, a value that is not in π and is different from every other value that has been added to the set can represent every other such value. Hence, these values are enough to cover the entire domain.[4] For the $\forall$ quantifier, we require that every respecting assignment leads to satisfaction. For the $\exists$ quantifier, we require that at least one respecting assignment leads to satisfaction.

Since different computations of a concrete system may satisfy the formula with different assignments to the same variable, ModelCheck, which checks the entire system against a single assignment, cannot be applied for checking concrete systems instead

[4] Note that we assume that $\mathcal{D}$ is sufficiently large to supply additional new values whenever the algorithm needs them. Our algorithms can be modified to account also for a small $\mathcal{D}$.

of computations. It can, however, be used to model check single paths in PSPACE. We assume, as usual for such a case, that this path is a lasso. Since we can easily reduce the problem of TQBF to model checking of a single path, a matching lower bound follows.

Theorem 1. *Let π be a lasso-shaped concrete computation, let $\varphi = \langle \psi, E\rangle$ be a VLTL formula over $P \cup T \cup (T \times X)$, and let $f : X \to \mathcal{D}$ be a partial function that respects E. Then deciding whether $\pi \models_f \langle \psi, E\rangle$ is PSPACE-complete.*

Proof: For the upper bound, using Lemma 1 we can show that for $B = Image(f)$ and for A, the finite set of values that occur in π, it holds that ModelCheck$(\pi, \langle \psi, E\rangle, f, A \cup B)$ returns *true* iff $\pi \models_f \langle \psi, E\rangle$. This procedure involves repeated runs of LTL model checking for π and a formula of the same size as ψ. Since each such run can be performed in PSPACE (in fact, in polynomial time), the entire procedure is run in PSPACE.

The lower bound is shown by a reduction from TQBF, the problem of deciding whether a closed quantified Boolean formula is valid, which is known to be PSPACE-complete. Given a quantified Boolean formula ψ, consider the single-state system $\mathcal{K}$ labeled $a.d$, where a is a parameterized atomic proposition, and d is some value, and the VLTL formula $\langle \psi', \emptyset\rangle$ obtained form ψ by replacing every variable x in ψ with $a.x$. Then, every truth assignment f to the variables of ψ is mapped to an assignment f' to the variables in ψ', where assigning *true* to x in f is equivalent to assigning d to x in f', and assigning *false* to x in f is equivalent to assigning $d' \neq d$ to x in f'. It can be shown that $f \models \psi$ iff $\mathcal{K} \models_{f'} \langle \psi', \emptyset\rangle$. Since ψ is closed, and therefore does not depend on f, showing this suffices. □

The second algorithm, VLTLtoLTL, translates the VLTL formula into an LTL formula, based on the values and the assignments of the given system $\mathcal{K}$ and function f. As in ModelCheck, a new value is added to a set C' that is maintained by the procedure (initially set to $A \cup B$, where A is the set of values in M, and $B = Image(f)$) for every quantifier in the formula. This time, every $\forall$ quantifier is translated to a conjunction of all of the recursively constructed formulas for these assignments, and every $\exists$ quantifier is translated to a disjunction.

Hence, the formula that is constructed by VLTLtoLTL contains every LTL formula that is checked by ModelCheck, and the conjunctions and disjunctions of VLTLtoLTL match the logical checks that are performed by ModelCheck.

VLTLtoLTL can then be used to model check entire concrete systems (in which case the formula is closed, and the initial function is $\emptyset$). While this leads to an exponentially large formula, this is the best that can be done, as we can show that the model checking problem for concrete systems is EXPSPACE complete, by a reduction from the acceptance problem for EXPSPACE Turing machines.

Theorem 2. *Let $\mathcal{K}$ be a concrete system over $P \cup T \times \mathcal{D}$ and let $\varphi = \langle \psi, E\rangle$ be a closed VLTL formula over $P \cup T \cup (T \times X)$. Then deciding whether $\mathcal{K} \models \varphi$ is EXPSPACE-complete.*

Proof: For the upper bound, using Lemma 1 we can show that for $B = Image(f)$ and for A, the finite set of values that occur in $\mathcal{K}$, it holds that $\mathcal{K} \models_f \langle \psi, E\rangle$ iff $\mathcal{K} \models$ VLTLtoLTL$(\langle \psi, E\rangle, f, A \cup B)$. The run VLTLtoLTL$(\langle \psi, E\rangle, f, A \cup B)$ produces an

LTL formula whose size is exponential in the size of $A \cup B$ and X. Since LTL model checking can be performed in PSPACE, checking $\mathcal{K} \models \mathsf{VLTLtoLTL}(\langle\psi, E\rangle, f, A \cup B)$ can be performed in EXPSPACE. Notice that if φ is closed, model checking is performed by using $\mathsf{VLTLtoLTL}(\langle\psi, E\rangle, \emptyset, A \cup B)$.

The lower bound is shown by a reduction from the acceptance problem for Turing machines that run in EXPSPACE. We sketch the proof. We define an encoding of runs of the Turing machine on a given input. For a Turing machine T and an input $\bar{a}$, we construct a system $\mathcal{K}$ whose computations include all encodings of potential runs of T on $\bar{a}$. We construct a VLTL formula φ that specifies computations that are not encodings of accepting runs of T on $\bar{a}$. Then, there exists an accepting run of T on $\bar{a}$ iff $\mathcal{K} \not\models \varphi$. $\square$

The formula we construct for the lower bound in the proof of Theorem 2 is in $\exists$-VLTL, and so the model-checking problem is EXPSPACE-complete already for this fragment of VLTL. However, a simple variant of $\mathsf{ModelCheck}$ can be used for $\forall$-VLTL. Together with the PSPACE lower bound for LTL model checking, we have the following.

Theorem 3. *The model-checking problem for $\forall$-VLTL and concrete systems is PSPACE-complete.*

4 Model Checking of Abstract Systems

In this section we consider the VLTL model-checking problem for abstract systems. We begin by showing that the problem is undecidable, by proving undecidability for the fragment of $\exists$-VLTL. Then, we show that for certain abstract systems, as well as for $\forall$-VLTL, model checking is not more difficult than for concrete systems.

Theorem 4. *The model-checking problem for $\exists$-VLTL is undecidable.*

Proof: We sketch the proof, which is by reduction from Post's Correspondence Problem (PCP). A similar reduction is shown in [14]. An instance of PCP are two sets $\{u_1, u_2, \ldots u_n\}$ and $\{v_1, v_2, \ldots v_n\}$ of finite words over $\{a, b\}$, and the problem is to decide whether there exists a concatenation $u = u_{i_1} u_{i_2} \cdots u_{i_k}$ of words of the first set that is identical to a concatenation $v = v_{i_1} v_{i_2} \cdots v_{i_k}$ of words of the second set.

We describe an encoding of a correct solution to a PCP instance given an input. For an instance I of PCP, we construct an abstract system $\mathcal{K}$ whose computations include all possible encodings of solutions to I, and an $\exists$-VLTL formula φ that specifies computations that are not legal encodings to a solution to I. Then, we have that I has a solution iff $\mathcal{K} \not\models \varphi$. $\square$

We now show that the VLTL model-checking problem for abstract systems is decidable for certain classes of systems. For the rest of the section, we consider abstract systems and abstract computations over P, $T \cup \{\mathit{reset}\}$, and X', and closed VLTL formulas over P, T, and X. We first introduce some terms.

We say that $\langle\mathcal{K}, E'\rangle$ is *bounded* if there is no occurrence of *reset* in $\mathcal{K}$. This means that the value of the variables does not change throughout a computation of the system. We say that $\langle\mathcal{K}, E'\rangle$ is *strict* if $E' = \{x'_i \neq x'_j | x'_i, x'_j, i \neq j \in X'\}$. Notice that a concrete computation in a bounded and strict system is obtained by an injection to the system variables.

We begin by showing that the model-checking problem for bounded systems is essentially equivalent to the model-checking problem for concrete systems.

Theorem 5. *The model-checking problem for bounded abstract systems is EXPSPACE-complete for VLTL, and PSPACE-complete for* $\forall$*-VLTL.*

Proof: Let $\langle \mathcal{K}, E' \rangle$ be a bounded abstract system, and let $\langle \psi, E \rangle$ be a VLTL formula.

We first prove the upper bounds for the case the system is both bounded and strict. Intuitively, assigning different values to the variables of a bounded and strict system results in a concrete system that satisfies the same set of formulas.

Formally, let $f : X' \to \mathcal{D}$ be an arbitrary injection, and let $\mathcal{K}_f$ be the concrete system that is obtained from $\langle \mathcal{K}, E' \rangle$ by substituting every occurrence of $x \in X'$ with $f(x)$. We can show that $\langle \mathcal{K}, E' \rangle \models \langle \psi, E \rangle$ iff $\mathcal{K}_f \models \langle \psi, E \rangle$.

We now turn to the general case of bounded systems, and reduce it to the case the systems are both bounded and strict. A set of bounded and strict abstract systems is obtained from $\langle \mathcal{K}, E' \rangle$ as follows. Consider the inequality set E'. Every possible setting of it induces a strict and bounded system: for every inequality $x_1 \neq x_2$ that is missing from E', both options of $x_1 \neq x_2$ and $x_1 = x_2$ are checked. For the former, a copy with $x_1 \neq x_2$ in the inequality set is constructed. For the latter, a copy with a new single variable replacing x_1 and x_2 is constructed. Then, we have that $\langle \mathcal{K}, E' \rangle \models \langle \psi, E \rangle$ iff every system in this set satisfies $\langle \psi, E \rangle$.

More specifically, consider a function $h : X' \to Z$ that respects E', where Z is a new set of variables of size $|X'|$. For every such h, an abstract system $\langle \mathcal{K}^h, E'^h \rangle$ is obtained from $\langle \mathcal{K}, E' \rangle$ by substituting every occurrence of $x \in X'$ with $h(x')$, and by setting E' to be the full inequality set. Then for $x_1, x_2 \in X'$, having $h(x_1) \neq h(x_2)$ is equivalent to setting $x_1 \neq x_2$, and $h(x_1) = h(x_2)$ is equivalent to setting $x_1 = x_2$. Every computation of $\langle \mathcal{K}, E' \rangle$ is a computation of $\langle \mathcal{K}^h, E'^h \rangle$ for some h, and vise versa.

Then, we have that $\langle \mathcal{K}, E' \rangle$does not satisfy $\langle \psi, E \rangle$ iff there exists a function h such that $\langle \mathcal{K}^h, E'^h \rangle$ does not satisfy $\langle \psi, E \rangle$. Therefore, the model-checking problem for bounded systems can be solved by guessing an appropriate function h, constructing, in linear time, a single copy of $\langle \mathcal{K}^h, E'^h \rangle$, and checking whether $\langle \mathcal{K}^h, E'^h \rangle \not\models \langle \psi, E \rangle$. This procedure is then performed in PSPACE in the size of the system and the formula.[5]

For the lower bounds, we reduce from the model-checking problem for concrete systems. Given a concrete system $\mathcal{M}$ and a VLTL formula $\langle \psi, E \rangle$, we construct in linear time a bounded (in fact, also strict) abstract system $\langle \mathcal{M}', E' \rangle$ by substituting every value d that occurs in $\mathcal{M}$ by the same unique variable x_d, and by setting E' to be the full inequality set. By a proof similar proof to the upper bound, we can show that $\langle \mathcal{M}', E' \rangle \models \langle \psi, E \rangle$ iff $\mathcal{M} \models \langle \psi, E \rangle$. □

Next, we show that the model-checking problem for abstract systems and $\forall$-VLTL formulas is, surprisingly, not more complex than the model-checking problem for LTL.

[5] For LTL, model checking is PSPACE-hard only in the size of the formula. For a fixed formula, LTL model checking can be performed in NLOGSPACE.

We do this by proving that this problem can be reduced to the model-checking problem for bounded abstract systems.

The following lemma shows that for a given assignment to the formula variables, the values in a concrete computation that are not assigned to any formula variable can be replaced with other such values without affecting the satisfiability.

Lemma 2. *Let $\langle \psi, E \rangle$ be a VLTL formula such that ψ is unquantified, and let $f : X \to \mathcal{D}$ be a function that respects E. Let π and τ be two concretizations of some abstract computation that agree on all values in $Image(f)$. Then $\pi \models_f \langle \psi, E \rangle$ iff $\tau \models_f \langle \psi, E \rangle$.*

We now show that in order to check whether an abstract computation ρ satisfies a $\forall$-VLTL formula φ, it is enough to check that every concretization of ρ that contains a bounded number of values satisfies φ.

Lemma 3. *Let $\langle \rho, E' \rangle$ be an abstract computation, and let $\langle \psi, E \rangle$ be a closed $\forall$-VLTL formula. Let C_ρ be the set of concretizations of $\langle \rho, E' \rangle$ that contains at most $|X'| + |X|$ different values. Then, $\langle \rho, E' \rangle \models \langle \psi, E \rangle$ iff for every $\pi \in C_\rho$, it holds that $\pi \models \langle \psi, E \rangle$.*

Proof: For the first direction, a computation in C_ρ is also a concretization of ρ.

For the second direction, suppose that for every $\tau \in C_\rho$, it holds that $\tau \models \langle \psi, E \rangle$. Assume by way of contradiction that there exists a concretization π of $\langle \rho, E' \rangle$ such that $\pi \not\models \langle \psi, E \rangle$. Since $\psi = \forall x_1; \ldots \forall x_k; \theta$, this means that there exists a function $f : X \to \mathcal{D}$ such that f respects E and $\pi \not\models \theta_f$.

If π contains at most $|X'| + |X|$ different values, then it is also in C_ρ. Therefore, π contains more than $|X'| + |X|$ different values. We show that there exists $\tau \in C_\rho$ such that $\tau \not\models \langle \psi, E \rangle$.

Let $a_1, a_2, \ldots a_k$ be the values that are assigned to the variables of X by f. Let $b_1, b_2, \ldots b_{k'}$, where $k' = |X'|$, be values different from the a values.

Let π' be the concrete computation obtained from ρ by assigning $a_1, a_2, \ldots a_k$ to the same occurrences of variables of X' that are assigned these values in π, and assigning every other occurrence of $x_i \in X'$ the same value b_i.

According to Lemma 2, we have that $\pi' \not\models \theta_f$. By the way we have constructed π', we have that π' is also a concretization of $\langle \rho, E' \rangle$, and that π' contains at most $|X'| + |X|$ different values. Therefore, it is also in C_ρ, a contradiction. □

Finally, we employ Lemma 3 in order to construct a model-checking procedure for $\forall$-VLTL, which runs in polynomial space.

Theorem 6. *The model-checking problem for $\forall$-VLTL and abstract systems is PSPACE-complete.*

Proof: The lower bound follows from the lower bound for LTL model checking.

Let $\langle \mathcal{K}, E' \rangle$ be an abstract system over P, $T \cup \{reset\}$, and X', where $|X'| = k'$, and let $\langle \psi, E \rangle$ be a closed $\forall$-VLTL formula over k variables. Intuitively, we construct from $\langle \mathcal{K}, E' \rangle$ a bounded system that contains exactly all computations of $\langle \mathcal{K}, E' \rangle$ that contain at most $k + k'$ different values.

Let λ be a set of new variables of size $k + k'$. Let $h : \lambda \to X'$ be an onto function. Intuitively, the function h partitions the variables of λ so that each set in the partition replaces a variable in X' in the construction.

Let $\Gamma_h = \{\{\xi_1, \xi_2 \ldots \xi_{k'}\} | \xi_i \in \lambda, h(\xi_i) = x_i, 1 \leq i \leq k'\}$.

For $\Delta \subseteq \Gamma_h$, let $\mathcal{K}_\Delta$ be the bounded system that is obtained from $\mathcal{K}$ as follows. For every set $\Gamma \in \Delta$, let $\mathcal{K}_\Gamma$ be the system obtained from $\mathcal{K}$ by replacing every occurrence of x_i with ξ_i for every $1 \leq i \leq k'$, and by removing every occurrence of $reset$. Then, in $\mathcal{K}_\Gamma$ every variable x in $\mathcal{K}$ is replaced with some variable ξ such that $h(\xi) = x$.

Let R be the set of transitions of $\mathcal{K}$. For every $\langle q, s\rangle \in R$, we add a transition from a copy of q in $\mathcal{K}_\Gamma$ to the copies of s in every $\mathcal{K}_{\Gamma'}$ such that Γ' and Γ agree on all variables in λ to which h assigns variables that are not reset in $\langle q, s\rangle$. Intuitively, a reset of a variable x is in a transition in $\mathcal{K}$ corresponds to switching from one variable in λ representing x to another.

Let $E_\Delta = \{\xi_i \neq \xi_j \, | h(\xi_i) \neq h(\xi_j) \in E'$, and $\xi_i, \xi_j \in \Gamma$ for some $\Gamma \in \Delta\}$. Then E_Δ is the inequality set induced by E' in $\mathcal{K}_\Delta$.

Note that $\mathcal{K}_\Delta$ models a copy of the system in which every occurrence of x_i between two consecutive resets is replaced by some variable in $h^{-1}(x_i)$. Also, for ξ_i and ξ_j such that $h(x_i) \neq h(x_j)$, if ξ_i, ξ_j are not in the same set in Δ, then they do not appear together in the same copy, and therefore are allowed to take the same value, even if $h(\xi_i) \neq h(\xi_j) \in E$.

Then, every abstract computation ρ of $\langle \mathcal{K}, E'\rangle$ is replaced with a set of bounded computations over $k + k'$ variables, whose set of concretizations is exactly the set of concretizations of ρ that contain at most $k + k'$ different values.

According to Lemma 3, we have that $\langle \mathcal{K}, E'\rangle \models \langle \psi, E\rangle$ iff for every h, and for every $\Delta \subseteq \Gamma_h$, it holds that $\langle \mathcal{K}_\Delta, E_\Delta\rangle \models \langle \psi, E\rangle$.

A nondeterministic procedure that runs in polynomial space guesses a function $f : X \to D$ and a function $g : \lambda \to D$, where $D \subset \mathcal{D}$ is some arbitrary set of size $2k + k'$. It follows from the proof of Theorem 5 and from Theorem 3 that a domain of the size of D is sufficient.

Next, in a procedure similar to the automata-theoretic approach to LTL model checking [18], the procedure constructs a violating path in the nondeterministic Büchi automaton $\mathcal{A}_{\neg\theta_f}$ that accepts exactly all computations that violate θ_f, constructs $\mathcal{K}_{\Delta_g}$ on the fly, and guesses a violating path that is accepted by both $\mathcal{A}_{\neg\theta_f}$ and $\mathcal{K}_{\Delta_g}$. While guessing a violating path in $\mathcal{K}_{\Delta_g}$, the procedure must make sure that every state respects E', that is, there exist no ξ_i and ξ_j such that $h(\xi_i) \neq h(\xi_j) \in E'$, and $g(\xi_i) = g(\xi_j)$, and both ξ_i and ξ_j are in the same state along the path. Since the information needed to guess a path in both $\mathcal{A}_{\neg\theta_f}$ and $\mathcal{K}_{\Delta_g}$ is polyomial, we have that the entire procedure can be performed in PSPACE in the size of the formula and of the system. $\square$

5 Conclusions

We presented a simple, general, and natural extension to LTL and Kripke structures. Our extension allows to augment atomic propositions with variables that range over some (possibly infinite) domain. In VLTL, our extension of LTL, the extension enables the specification to refer to the domain values. In abstract systems, our extension of Kripke structures, the extension enables a compact description of infinite and complex concrete systems whose control is finite, and for which the source of infinity is the data.

We studied the model-checking problem in this setting, for both finite concrete systems and for abstract systems. We presented a model-checking procedure for VLTL and concrete systems. We showed that the general problem is EXPSPACE-complete for concrete systems and undecidable for abstract systems. As good news, we showed that even for abstract systems, the model-checking problem for the rich fragment of ∀-VLTL is not only decidable, but is of the same complexity as LTL model checking.

References

1. Barcelo, P., Libkin, L., Reutter, J.: Parameterized regular expressions and their languages. In: FSTTCS (2011)
2. Clarke, E.M., Grumberg, O., Browne, M.C.: Reasoning about Networks with many identical Finite-State Processes. In: PODC 1986 (1986)
3. Clarke, E.M., Grumberg, O., Long, D.E.: Model Checking and Abstraction. In: TOPLAS 1994 (1994)
4. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press (1999)
5. Demri, S., D'Souza, D.: An Automata-Theoretic Approach to Constraint LTL. In: Agrawal, M., Seth, A.K. (eds.) FSTTCS 2002. LNCS, vol. 2556, pp. 121–132. Springer, Heidelberg (2002)
6. Dingel, J., Filkorn, T.: Model Checking for Infinite State Systems Using Data Abstraction, Assumption-Commitment Style Reasoning and Theorem Proving. In: Wolper, P. (ed.) CAV 1995. LNCS, vol. 939, pp. 54–69. Springer, Heidelberg (1995)
7. Esparza, J., Hansel, D., Rossmanith, P., Schwoon, S., Nutzung, D.: Efficient Algorithms for Model Checking Pushdown Systems. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 232–247. Springer, Heidelberg (2000)
8. German, S., Sistla, A.P.: Reasoning about systems with many processes. JACM (1992)
9. Grumberg, O., Kupferman, O., Sheinvald, S.: Variable Automata over Infinite Alphabets. In: Dediu, A.-H., Fernau, H., Martı́n-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 561–572. Springer, Heidelberg (2010)
10. Henzinger, T.A.: Hybrid Automata with Finite Bisimulations. In: Fülöp, Z. (ed.) ICALP 1995. LNCS, vol. 944, pp. 324–335. Springer, Heidelberg (1995)
11. Lazic, R.: Safely freezing LTL. In: FSTSTC (2006)
12. Lichtenstein, O., Pnueli, A.: Checking that finite state concurrent programs satisfy their linear specification. Proc. 12th POPL (1985)
13. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer (1992)
14. Neven, F., Schwentick, T., Vianu, V.: Towards Regular Languages over Infinite Alphabets. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 560–572. Springer, Heidelberg (2001)
15. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logic. JACM (1985)
16. Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for Büchi automata with applications to temporal logic. Theoretical Computer Science (1987)
17. Vardi, M.Y.: Personal communication (2011)
18. Vardi, M.Y., Wolper, P.: Automata-theoretic techniques for modal logics of programs. Journal of Computer and Systems Science (1986)
19. Chomicki, J., Toman, D.: Temporal Logic in Information Systems. In: Logics for Databases and Information Systems, pp. 31–70 (1998)
20. Dixon, C., Fisher, M., Konev, B., Lisitsa, A.: Efficient First-Order Temporal Logic for Infinite-State Systems. CoRR (2007)
21. Hodkinson, I., Wolter, F., Zakharyaschev, M.: Decidable Fragments of First-Order Temporal Logics. Annals of Pure and Applied Logic (1999)
22. Alur, R.: Timed Automata. Theoretical Computer Science, pp. 183–235 (1999)