Journal of Applied Logic 12 (2014) 584–613

Contents lists available at ScienceDirect

Journal of Applied Logic

www.elsevier.com/locate/jal

Runtime verification using the temporal description logic $\mathcal{ALC}$-LTL revisited

Franz Baader, Marcel Lippmann

Institute of Theoretical Computer Science, Technische Universität Dresden, 01062 Dresden, Germany

ARTICLE INFO

Article history:
Received 19 March 2014
Accepted 21 August 2014
Available online 6 September 2014

Keywords:
Temporal description logics
Runtime verification
Monitoring

ABSTRACT

Formulae of linear temporal logic (LTL) can be used to specify (wanted or unwanted) properties of a dynamical system. In model checking, the system's behaviour is described by a transition system, and one needs to check whether all possible traces of this transition system satisfy the formula. In runtime verification, one observes the actual system behaviour, which at any point in time yields a finite prefix of a trace. The task is then to check whether all continuations of this prefix to a trace satisfy (violate) the formula. More precisely, one wants to construct a monitor, i.e., a finite automaton that receives the finite prefix as input and then gives the right answer based on the state currently reached.

In this paper, we extend the known approaches to LTL runtime verification in two directions. First, instead of *propositional* LTL we use the more expressive temporal logic $\mathcal{ALC}$-LTL, which can use axioms of the Description Logic (DL) $\mathcal{ALC}$ instead of propositional variables to describe properties of single states of the system. Second, instead of assuming that the observed system behaviour provides us with complete information about the states of the system, we assume that states are described in an incomplete way by $\mathcal{ALC}$-knowledge bases. We show that also in this setting monitors can effectively be constructed. The (double-exponential) size of the constructed monitors is in fact optimal, and not higher than in the propositional case. As an auxiliary result, we show how to construct Büchi automata for $\mathcal{ALC}$-LTL-formulae, which yields alternative proofs for the known upper bounds of deciding satisfiability in $\mathcal{ALC}$-LTL.

1. Introduction

Formulae of linear temporal logic (LTL) [26] can be used to specify (wanted or unwanted) properties of a dynamical system. For example, assume that the system we want to model is a TV set, and consider the properties on, turn_off, and turn_on, which respectively express that the set is on, receives a turn-off signal from the remote control, and receives a turn-on signal from the remote control. The LTL-formula

E-mail addresses: baader@tcs.inf.tu-dresden.de (F. Baader), lippmann@tcs.inf.tu-dresden.de (M. Lippmann).

http://dx.doi.org/10.1016/j.jal.2014.09.001
1570-8683/© 2014 Elsevier B.V. All rights reserved.

$$\phi_{\mathrm{tv}} := \Box\big(\mathsf{turn\_on} \rightarrow \mathsf{X}\big(\mathsf{on} \wedge (\mathsf{X}\mathsf{on})\,\mathsf{U}\,\mathsf{turn\_off}\big)\big)$$

says that, whenever the set receives the turn-on signal, it is on at the next time point, and it stays on (i.e., is on also at the next time point) until it receives the turn-off signal (since we use a "strong until" this signal has to come eventually).

In model checking [7,12], one assumes that the system's behaviour can be described by a transition system. The verification task is then to check whether all possible traces of this transition system satisfy the formula. In contrast, in runtime verification [13], one does not model all possible behaviours of the system by a transition system. Instead, one observes the actual behaviour of the system, which at any time point yields a finite prefix u of a trace. The task is then to check whether all continuations of this prefix to a trace satisfy (violate) the given LTL-formula ϕ. Thus, there are three possible answers[1] to a runtime verification problem (u, ϕ):

- $\top$, if all continuations of u to an infinite trace satisfy ϕ;
- $\bot$, if all continuations of u to an infinite trace do not satisfy ϕ;
- ?, if none of the above holds, i.e., there is a continuation that satisfies ϕ, and one that does not satisfy ϕ.

For example, consider the two prefixes $u := \{\neg\mathsf{on}, \neg\mathsf{turn\_off}, \mathsf{turn\_on}\}$ and $u' := \{\neg\mathsf{on}, \neg\mathsf{turn\_off}, \mathsf{turn\_on}\}$ $\{\neg\mathsf{on}, \neg\mathsf{turn\_off}, \neg\mathsf{turn\_on}\}$ and the formula ϕ_{tv} from our example. For the prefix u, the answer is ?, whereas for u' it is $\bot$. For our specific formula ϕ_{tv}, there is no prefix for which the answer would be $\top$.

It should be noted, however, that runtime verification is not really about solving a single such problem (u, ϕ). In practice, one observes the behaviour of the system over time, which means that the prefix is continuously extended by adding new letters. The runtime verification device should not simply answer the problems $(\varepsilon, \phi), (\sigma_0, \phi), (\sigma_0\sigma_1, \phi), (\sigma_0\sigma_1\sigma_2, \phi), \ldots$ independently of each other. What one is looking for is a monitoring device (called *monitor* in the following) that successively accepts as input the next letter, and then computes the answer to the next runtime verification problem in constant time (where the size of ϕ is assumed to be constant). This can, for example, be achieved as follows [9,11]. For a given LTL-formula ϕ, one constructs a deterministic Moore automaton $\mathcal{M}_\phi$ (i.e., a deterministic finite-state automaton with state output) such that the state reached by processing input u gives as output the answer to the runtime verification problem (u, ϕ). If u is then extended to $u\sigma$ by observing the next letter σ of the actual system behaviour, it is sufficient to perform one transition of $\mathcal{M}_\phi$ in order to get the answer for $(u\sigma, \phi)$. Since $\mathcal{M}_\phi$ depends on ϕ (which is assumed to be constant), but not on u, this kind of monitoring device can answer the runtime verification question for (u, ϕ) in time linear in the length of u. More importantly, the delay between answering the question for u and for $u\sigma$ is constant, i.e., it does not depend on the length of the already processed prefix u. Basically, such a monitor can be constructed from Büchi automata for the formula ϕ and its negation $\neg\phi$.[2]

Using *propositional* LTL for runtime verification presupposes that (the relevant information about) the states of the system can be represented using propositional variables, more precisely conjunctions of propositional literals. If the states actually have a complex internal structure, this assumption is not realistic. In order to allow for a more appropriate description of such complex states, one can use the extension of propositional LTL to $\mathcal{ALC}$-LTL introduced in [4,6].[3] From the syntactic point of view, the difference between propositional LTL and $\mathcal{ALC}$-LTL is that, in the latter, $\mathcal{ALC}$-axioms (i.e., concept and role assertions as well as general concept inclusion axioms (GCIs) formulated in the Description Logic $\mathcal{ALC}$ [28]) are used in place of propositional variables. From the semantic point of view, $\mathcal{ALC}$-LTL structures are infinite sequences

[1] There are also variants of runtime verification for propositional LTL that work with only two or even four possible answers [10].

[2] A Büchi automaton for an LTL-formula ψ accepts the LTL structures satisfying this formula, viewed as infinite words over an appropriate alphabet [7,32].

[3] A comparison of $\mathcal{ALC}$-LTL with other temporal DLs [2,3,25] is beyond the scope of this introduction. It can be found in [4,6].