Runtime Verification Using a Temporal Description Logic

Franz Baader,[1] Andreas Bauer,[2] and Marcel Lippmann[1]

[1] TU Dresden, {baader,lippmann}@tcs.inf.tu-dresden.de
[2] The Australian National University, baueran@rsise.anu.edu.au

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 behavior 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 behavior, which at any time point 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.
In this paper, we extend the known approaches to LTL runtime verification in two directions. First, instead of *propositional* LTL we use $\mathcal{ALC}$-LTL, which can use axioms of the description logic $\mathcal{ALC}$ instead of propositional variables to describe properties of single states of the system. Second, instead of assuming that the observed system behavior provides us with complete information about the states of the system, we consider the case where states may be described in an incomplete way by $\mathcal{ALC}$-ABoxes.

1 Introduction

Formulae of linear temporal logic (LTL) [11] 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, $\mathsf{turn\_on}$, and $\mathsf{turn\_off}$, 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 $\phi_{tv} := \Box\,(\mathsf{turn\_off} \rightarrow \mathsf{X}(\mathsf{off} \wedge (\mathsf{X}\mathsf{off})\,\mathsf{U}\,\mathsf{turn\_on}))$ says that, whenever the set receives the turn-off signal, it is off at the next time point, and it stays off (i.e., is off also at the next time point) until it receives the turn-on signal.

In model checking [7, 4], one assumes that the system's behavior 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 [8], one does not model all possible behaviors of the system by a transition system. Instead, one observes the actual behavior 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 $u := \{\mathsf{on}, \neg\mathsf{turn\_on}, \mathsf{turn\_off}\}$ and $u' := \{\mathsf{on}, \neg\mathsf{turn\_on}, \mathsf{turn\_off}\}\,\{\mathsf{on}, \neg\mathsf{turn\_on}, \neg\mathsf{turn\_off}\}$ and the formula ϕ_{tv} from above. For the first prefix, the answer is ?, whereas for the second 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 behavior 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 [5]. 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 behavior, 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 generalized 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 [3].[3] From the syntactic point of view, the difference between propositional LTL and

[1] There are also variants of runtime verification that work with only two or even four possible answers [6].

[2] A generalized Büchi automaton for an LTL formula ψ accepts the LTL structures satisfying this formula, viewed as words over an appropriate alphabet [16, 4].

[3] A comparison of $\mathcal{ALC}$-LTL with other temporal DLs [1, 2, 10] is beyond the scope of this introduction. It can be found in [3].

$\mathcal{ALC}$-LTL is that, in the latter, $\mathcal{ALC}$-axioms (i.e., concept and role assertions as well as general concept inclusion axioms formulated in the description logic $\mathcal{ALC}$ [14]) are used in place of propositional letters. From the semantic point of view, $\mathcal{ALC}$-LTL structures are infinite sequences of $\mathcal{ALC}$-interpretations, i.e., first-order relational structures, rather than propositional valuations. In [3], the complexity of the satisfiability problem for $\mathcal{ALC}$-LTL formulae is investigated in detail. In particular, it is shown that this complexity depends on whether rigid concepts and roles (i.e., concepts/roles whose interpretation does not change over time) are available or not. The algorithms for deciding satisfiability of $\mathcal{ALC}$-LTL formulae developed in [3] are not based on generalized Büchi automata. Before we can adapt the monitor construction used for propositional LTL to the case of $\mathcal{ALC}$-LTL, we must first show how Büchi automata for $\mathcal{ALC}$-LTL formulae can be constructed. We will see that this construction becomes more complex in the presence of rigid concepts and roles.

In runtime verification for propositional LTL, one usually assumes that the observed prefix provides one with complete information about the relevant system properties. In the setting of runtime verification for $\mathcal{ALC}$-LTL, this completeness assumption would mean that, for every time point covered by it, the prefix must provide full information about the status of every $\mathcal{ALC}$-axiom occurring in the formula, i.e., it must say whether it is true at that time point or not. If one has only limited access to the system's behavior, this assumption may be too strict. In this paper we show that runtime verification is also possible under the weaker assumption that one has (possibly) incomplete knowledge about the system's behavior at a time point. Technically, this means that we assume that the prefix describing the system's behavior is a finite sequence of ABoxes. Given such an ABox and an axiom occurring in the formula, there are now three possible cases: the axiom may follow from the ABox, its negation may follow from the ABox, or neither of them follows from the ABox. The third case means that we do not know whether in this state of the system the axiom or its negation holds. Thus, in addition to the unknown continuation of the prefix in the future, the use of ABoxes as (possibly) incomplete descriptions of states adds another source of uncertainty, which may cause the monitor to answer with ?.

As a possible application of this kind of monitoring, consider an emergency ward, where the vital parameters of a patient are measured in short intervals (sometimes not longer than 10 minutes), and where additional information about the patient is available from the patient record and added by doctors and nurses. Using concepts defined in a medical ontology like SNOMED CT,[4] a high-level view of the medical status of the patient at a given time point can be given by an ABox. Critical situations, which require the intervention of a doctor, can then be described by an $\mathcal{ALC}$-LTL formula (see [3] for a simple example). As long as the monitor for this formula yields the output ?, we continue with monitoring. If it yields $\top$, we raise an alarm, and if it yields $\bot$ we can shut off this monitor.

In the next section, we introduce the temporal description logic $\mathcal{ALC}$-LTL, and in Section 3 we show how to construct generalized Büchi automata for $\mathcal{ALC}$-

[4] see http://www.ihtsdo.org/our-standards/

LTL formulae. These generalized Büchi automata are then used in Section 4 to construct monitors for $\mathcal{ALC}$-LTL formulae.

2 The temporal DL $\mathcal{ALC}$-LTL

The temporal DL $\mathcal{ALC}$-LTL introduced in [3] combines the basic DL $\mathcal{ALC}$ with linear temporal logic (LTL). First, we recall the relevant definitions for $\mathcal{ALC}$.

Definition 1. *Let N_C, N_R, and N_I respectively be disjoint sets of* concept names*,* role names*, and* individual names*. The set of $\mathcal{ALC}$-*concept descriptions *is the smallest set such that*

- *all concept names are $\mathcal{ALC}$-concept descriptions;*
- *if C, D are $\mathcal{ALC}$-concept descriptions and $r \in N_R$, then $\neg C$, $C \sqcup D$, $C \sqcap D$, $\exists r.C$, and $\forall r.C$ are $\mathcal{ALC}$-concept descriptions.*

A general concept inclusion axiom (GCI) *is of the form $C \sqsubseteq D$, where C, D are $\mathcal{ALC}$-concept descriptions, and an* assertion *is of the form $a : C$ or $(a, b) : r$ where C is an $\mathcal{ALC}$-concept description, r is a role name, and a, b are individual names. We call both GCIs and assertions $\mathcal{ALC}$-*axioms*. A Boolean combination of $\mathcal{ALC}$-axioms is called a* Boolean $\mathcal{ALC}$-knowledge base*, i.e.,*

- *every $\mathcal{ALC}$-axiom is a Boolean $\mathcal{ALC}$-knowledge base;*
- *if $\mathcal{B}_1$ and $\mathcal{B}_2$ are Boolean $\mathcal{ALC}$-knowledge bases, then so are $\mathcal{B}_1 \wedge \mathcal{B}_2$, $\mathcal{B}_1 \vee \mathcal{B}_2$, and $\neg \mathcal{B}_1$.*

*An $\mathcal{ALC}$-*TBox *is a conjunction of GCIs, and an $\mathcal{ALC}$-*ABox *is a conjunction of assertions.*

According to this definition, TBoxes and ABoxes are special kinds of Boolean knowledge bases. However, note that they are often written as sets of axioms rather than as conjunctions of these axioms. The semantics of $\mathcal{ALC}$ is defined through the notion of an interpretation.

Definition 2. *An $\mathcal{ALC}$-*interpretation *is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where the* domain *$\Delta^{\mathcal{I}}$ is a non-empty set, and $\cdot^{\mathcal{I}}$ is a function that assigns to every concept name A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, to every role name r a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to every individual name a an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. This function is extended to $\mathcal{ALC}$-concept descriptions as follows:*

- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}, (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$;
- $(\exists r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid$ *there is a* $y \in \Delta^{\mathcal{I}}$ *with* $(x, y) \in r^{\mathcal{I}}$ *and* $y \in C^{\mathcal{I}}\}$;
- $(\forall r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid$ *for all* $y \in \Delta^{\mathcal{I}}$, $(x, y) \in r^{\mathcal{I}}$ *implies* $y \in C^{\mathcal{I}}\}$.

We say that the interpretation $\mathcal{I}$ satisfies the unique name assumption (UNA) *iff different individual names are interpreted by different elements of the domain. The interpretation $\mathcal{I}$ is a* model *of the $\mathcal{ALC}$-axioms $C \sqsubseteq D$, $a : C$, and $(a, b) : r$ iff it respectively satisfies $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$. The notion of a model is extended to Boolean $\mathcal{ALC}$-knowledge bases as follows:*

– $\mathcal{I}$ *is a model of* $\mathcal{B}_1 \wedge \mathcal{B}_2$ *iff it is a model of* $\mathcal{B}_1$ *and* $\mathcal{B}_2$*;*
– $\mathcal{I}$ *is a model of* $\mathcal{B}_1 \vee \mathcal{B}_2$ *iff it is a model of* $\mathcal{B}_1$ *or* $\mathcal{B}_2$*;*
– $\mathcal{I}$ *is a model of* $\neg\mathcal{B}_1$ *iff it is not a model of* $\mathcal{B}_1$*.*

We say that the Boolean $\mathcal{ALC}$*-knowledge base* $\mathcal{B}$ *is* consistent *iff it has a model. We say that* $\mathcal{B}$ implies *the* $\mathcal{ALC}$*-axiom* α *iff every model of* $\mathcal{B}$ *is a model of* α*.*

Instead of first introducing the propositional temporal logic LTL, we directly define its extension $\mathcal{ALC}$-LTL. The difference to propositional LTL is that $\mathcal{ALC}$-axioms replace propositional letters.

Definition 3. *$\mathcal{ALC}$-LTL formulae are defined by induction:*

– *if* α *is an* $\mathcal{ALC}$*-axiom, then* α *is an* $\mathcal{ALC}$*-LTL formula;*
– *if* ϕ, ψ *are* $\mathcal{ALC}$*-LTL formulae, then so are* $\phi \wedge \psi$*,* $\neg\phi$*,* $\phi \mathsf{U} \psi$*, and* $\mathsf{X}\phi$*.*

As usual, we use $\phi \vee \psi$ as an abbreviation for $\neg(\neg\phi \wedge \neg\psi)$, true as an abbreviation for $(a : A) \vee \neg(a : A)$, $\Diamond\phi$ as an abbreviation for $\mathsf{true}\mathsf{U}\phi$ (*diamond*, which should be read as "some time in the future"), and $\Box\phi$ as an abbreviation for $\neg\Diamond\neg\phi$ (*box*, which should be read as "always in the future"). The semantics of $\mathcal{ALC}$-LTL is based on $\mathcal{ALC}$-LTL structures, which are sequences of $\mathcal{ALC}$-interpretations over the same non-empty domain Δ (constant domain assumption). We assume that every individual name stands for a unique element of Δ (rigid individual names), and we make the unique name assumption.

Definition 4. *An* $\mathcal{ALC}$-LTL structure *is a sequence* $\mathfrak{I} = (\mathcal{I}_i)_{i=0,1,\ldots}$ *of* $\mathcal{ALC}$*-interpretations* $\mathcal{I}_i = (\Delta, \cdot^{\mathcal{I}_i})$ *obeying the UNA (called* worlds*) such that* $a^{\mathcal{I}_i} = a^{\mathcal{I}_j}$ *for all individual names* a *and all* $i, j \in \{0, 1, 2, \ldots\}$*. Given an* $\mathcal{ALC}$*-LTL formula* ϕ*, an* $\mathcal{ALC}$*-LTL structure* $\mathfrak{I} = (\mathcal{I}_i)_{i=0,1,\ldots}$*, and a time point* $i \in \{0, 1, 2, \ldots\}$*, validity of* ϕ *in* $\mathfrak{I}$ *at time* i *(written* $\mathfrak{I}, i \models \phi$*) is defined inductively:*

$$
\begin{array}{ll}
\mathfrak{I}, i \models C \sqsubseteq D & \text{iff } C^{\mathcal{I}_i} \subseteq D^{\mathcal{I}_i} \\
\mathfrak{I}, i \models a : C & \text{iff } a^{\mathcal{I}_i} \in C^{\mathcal{I}_i} \\
\mathfrak{I}, i \models (a, b) : r & \text{iff } (a^{\mathcal{I}_i}, b^{\mathcal{I}_i}) \in r^{\mathcal{I}_i} \\
\mathfrak{I}, i \models \phi \wedge \psi & \text{iff } \mathfrak{I}, i \models \phi \text{ and } \mathfrak{I}, i \models \psi \\
\mathfrak{I}, i \models \neg\phi & \text{iff not } \mathfrak{I}, i \models \phi \\
\mathfrak{I}, i \models \mathsf{X}\phi & \text{iff } \mathfrak{I}, i+1 \models \phi \\
\mathfrak{I}, i \models \phi \mathsf{U} \psi & \text{iff there is } k \geq i \text{ such that } \mathfrak{I}, k \models \psi \\
 & \quad \text{and } \mathfrak{I}, j \models \phi \text{ for all } j, i \leq j < k
\end{array}
$$

As mentioned before, for some concepts and roles it is not desirable that their interpretation changes over time. For example, in a medical application, we may want to assume that the gender and the father of a patient do not change over time, whereas the health status of a patient may of course change. Thus, we will assume that a subset of the set of concept and role names can be designated as being rigid. We will call the elements of this subset *rigid concept names* and *rigid role names*. All other concept and role names are called *flexible*.

Definition 5. *We say that the $\mathcal{ALC}$-LTL structure $\mathfrak{I} = (\mathcal{I}_i)_{i=0,1,\ldots}$* respects rigid names *iff $A^{\mathcal{I}_i} = A^{\mathcal{I}_j}$ and $r^{\mathcal{I}_i} = r^{\mathcal{I}_j}$ holds for all $i, j \in \{0, 1, 2, \ldots\}$, all rigid concept names A, and all rigid role names r.*

The $\mathcal{ALC}$-LTL formula ϕ is satisfiable w.r.t. rigid names *iff there is an $\mathcal{ALC}$-LTL structure $\mathfrak{I}$ respecting rigid names such that $\mathfrak{I}, 0 \models \phi$. It is* satisfiable without rigid names *(or simply* satisfiable*) iff there is an $\mathcal{ALC}$-LTL structure $\mathfrak{I}$ such that $\mathfrak{I}, 0 \models \phi$.*

The $\mathcal{ALC}$-LTL structure $\mathfrak{I}$ is a model of the $\mathcal{ALC}$-LTL formula ϕ (w.r.t. rigid names) iff $\mathfrak{I}, 0 \models \phi$ (and $\mathfrak{I}$ respects rigid names).

In [3], it is shown that satisfiability w.r.t. rigid names in $\mathcal{ALC}$-LTL is 2-ExpTime-complete, whereas satisfiability without rigid names is "only" ExpTime-complete. The decision procedures developed in [3] to show the complexity upper bounds are not based on generalized Büchi automata. In the next section, we show, however, that the ideas underlying these decision procedures can also be used to obtain automata-based decision procedures.

3 Generalized Büchi automata for $\mathcal{ALC}$-LTL formulae

For propositional LTL, the satisfiability problem can be decided by first constructing a generalized Büchi automaton for the given formula, and then testing this automaton for emptiness. Generalized Büchi automata can be used to define ω-languages, i.e., sets of infinite words. For an alphabet Σ, we denote the set of all infinite words over Σ by Σ^ω.

Definition 6. *A* generalized Büchi automaton $\mathcal{G} = (Q, \Sigma, \Delta, Q_0, \mathcal{F})$ *consists of a finite set of states Q, a finite input alphabet Σ, a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, a set $Q_0 \subseteq Q$ of initial states, and a set of sets of final states $\mathcal{F} \subseteq 2^Q$.*

Given an infinite word $w = \sigma_0\sigma_1\sigma_2 \ldots \in \Sigma^\omega$, a run *of $\mathcal{G}$ on w is an infinite word $q_0q_1q_2 \ldots \in Q^\omega$ such that $q_0 \in Q_0$ and $(q_i, \sigma_i, q_{i+1}) \in \Delta$ for all $i \geq 0$. This run is* accepting *if, for every $F \in \mathcal{F}$, there are infinitely many $i \geq 0$ such that $q_i \in F$. The* language accepted by *$\mathcal{G}$ is defined as*

$$L_\omega(\mathcal{G}) := \{w \in \Sigma^\omega \mid \textit{there is an accepting run of } \mathcal{G} \textit{ on } w\}.$$

The emptiness problem *for generalized Büchi automata is the problem of deciding, given a generalized Büchi automaton $\mathcal{G}$, whether $L_\omega(\mathcal{G}) = \emptyset$ or not.*

We use *generalized* Büchi automata rather than normal ones (where $|\mathcal{F}| = 1$) since this allows for a simpler construction of the automaton for a given $\mathcal{ALC}$-LTL formula. It is well-known that a generalized Büchi automaton can be transformed into an equivalent normal one in polynomial time [9, 4]. Together with the fact that the emptiness problem for normal Büchi automata can be solved in polynomial time [15], this yields a polynomial time bound for the complexity of the emptiness problem for generalized Büchi automata.

3.1 The case without rigid names

In principle, given an $\mathcal{ALC}$-LTL formula ϕ, we want to construct a generalized Büchi automaton $\mathcal{G}_\phi$ that accepts exactly the models of ϕ. However, since there are infinitely many $\mathcal{ALC}$-interpretations, we would end up with an infinite alphabet for this automaton. For this reason, we abstract from the specific interpretations, and only consider their $\mathcal{ALC}$-types. We call an $\mathcal{ALC}$-axiom α a *ϕ-axiom* if it occurs in ϕ. A *ϕ-literal* is a ϕ-axiom or the negation of a ϕ-axiom. For example, the formula

$$\phi_{ex} := \mathsf{X}(a : A) \wedge ((A \sqsubseteq B)\,\mathsf{U}\,(a : \neg B)) \tag{1}$$

has $a : A, A \sqsubseteq B, a : \neg B, \neg(a : A), \neg(A \sqsubseteq B), \neg(a : \neg B)$ as its literals. In the following, we assume that an arbitrary (but fixed) $\mathcal{ALC}$-LTL formula ϕ is given.

Definition 7. *The set of ϕ-literals T is an* $\mathcal{ALC}$-type for ϕ *iff the following two properties are satisfied:*

1. *For every ϕ-axiom α we have $\alpha \in T$ iff $\neg\alpha \notin T$.*
2. *The Boolean $\mathcal{ALC}$-knowledge base $\mathcal{B}_T := \bigwedge_{\alpha \in T} \alpha$ is consistent.*

We denote the set of all $\mathcal{ALC}$-types for ϕ with Σ^ϕ.

For example, $\{a : A, A \sqsubseteq B, \neg(a : \neg B)\}$ is an $\mathcal{ALC}$-type for ϕ_{ex} whereas $\{a : A, A \sqsubseteq B, a : \neg B\}$ is not (since it violates the second condition).

Given an $\mathcal{ALC}$-interpretation $\mathcal{I}$, we define its ϕ-type $\tau_\phi(\mathcal{I})$ as the set of all ϕ-literals that $\mathcal{I}$ is a model of. It is easy to see that $\tau_\phi(\mathcal{I})$ is an $\mathcal{ALC}$-type for ϕ. Conversely, given an $\mathcal{ALC}$-type T for ϕ, the model $\mathcal{I}$ of $\mathcal{B}_T$ is such that $\tau_\phi(\mathcal{I}) = T$. The evaluation of ϕ in an $\mathcal{ALC}$-LTL structure only depends on the ϕ-types of the $\mathcal{ALC}$-interpretations in this structure. To be more precise, given an $\mathcal{ALC}$-LTL structure $\mathfrak{I} = (\mathcal{I}_i)_{i=0,1,\ldots}$, its ϕ-type is the following infinite word over Σ^ϕ: $\tau_\phi(\mathfrak{I}) := \tau_\phi(\mathcal{I}_0)\tau_\phi(\mathcal{I}_1)\tau_\phi(\mathcal{I}_2)\ldots$.

If $\mathfrak{I}, \mathfrak{J}$ are two $\mathcal{ALC}$-LTL structures whose ϕ-types coincide, then we have $\mathfrak{I}, i \models \phi$ iff $\mathfrak{J}, i \models \phi$ for all $i \geq 0$. In particular, $\mathfrak{I}$ is a model of ϕ iff $\mathfrak{J}$ is a model of ϕ. Instead of accepting the models of ϕ, the generalized Büchi automaton $\mathcal{G}_\phi$ will accept their ϕ-types.

The states of $\mathcal{G}_\phi$ are types that also take the structure (and not just the axioms) of the $\mathcal{ALC}$-LTL formula ϕ into account. A *sub-literal* of ϕ is a subformula or its negation. For example, the formula ϕ_{ex} in (1) has the sub-literals $\phi_{ex}, \mathsf{X}(a : A), a : A, (A \sqsubseteq B)\,\mathsf{U}\,(a : \neg B), A \sqsubseteq B, a : \neg B, \neg\phi_{ex}, \neg\mathsf{X}(a : A), \neg(a : A), \neg\left((A \sqsubseteq B)\,\mathsf{U}\,(a : \neg B)\right), \neg(A \sqsubseteq B), \neg(a : \neg B)$.

Definition 8. *The set T of sub-literals of ϕ is an* $\mathcal{ALC}$-LTL-type for ϕ *iff the following properties are satisfied:*

1. *For every sub-formula ψ of ϕ we have $\psi \in T$ iff $\neg\psi \notin T$.*
2. *For every sub-formula $\psi_1 \wedge \psi_2$ of ϕ we have $\psi_1 \wedge \psi_2 \in T$ iff $\{\psi_1, \psi_2\} \subseteq T$.*
3. *For every sub-formula $\psi_1\,\mathsf{U}\,\psi_2$ of ϕ we have*

- $\psi_2 \in T \Rightarrow \psi_1 \mathsf{U} \psi_2 \in T$,
- $\psi_1 \mathsf{U} \psi_2 \in T$ *and* $\psi_2 \notin T \Rightarrow \psi_1 \in T$.

4. *The restriction of* T *to its* ϕ*-literals is an* $\mathcal{ALC}$*-type for* ϕ.

We denote the set of all $\mathcal{ALC}$*-LTL-types for* ϕ *by* Q^ϕ.

For example, $\{\phi_{ex}, \mathsf{X}(a : A), a : A, (A \sqsubseteq B)\,\mathsf{U}\,(a : \neg B), A \sqsubseteq B, \neg(a : \neg B)\}$ is an $\mathcal{ALC}$-LTL-type for ϕ_{ex}.

The conditions for until in the definition of an $\mathcal{ALC}$-LTL-type T allow an until-formula $\psi_1 \,\mathsf{U}\, \psi_2 \in T$ to be satisfied either now ($\psi_2 \in T$) or later ($\psi_1 \in T$). The automaton $\mathcal{G}_\phi$ uses the generalized Büchi-acceptance condition to prevent that satisfaction of until formulae is deferred indefinitely. This automaton has the set of all $\mathcal{ALC}$-types for ϕ as its alphabet and the set of all $\mathcal{ALC}$-LTL-types for ϕ as its set of states.

Definition 9. *Given an* $\mathcal{ALC}$*-LTL formula* ϕ*, the corresponding generalized Büchi automaton* $\mathcal{G}_\phi = (Q^\phi, \Sigma^\phi, \Delta^\phi, Q_0^\phi, \mathcal{F}^\phi)$ *is defined as follows:*

- $\Delta^\phi \subseteq Q^\phi \times \Sigma^\phi \times Q^\phi$ *is defined as follows:* $(q, \sigma, q') \in \Delta^\phi$ *iff*
  - σ *is the restriction of* q *to its* ϕ*-literals;*
  - $\mathsf{X}\psi \in q$ *implies* $\psi \in q'$*;*
  - $\psi_1 \mathsf{U} \psi_2 \in q$ *implies that (i)* $\psi_2 \in q$ *or (ii)* $\psi_1 \in q$ *and* $\psi_1 \mathsf{U} \psi_2 \in q'$*;*
- $Q_0^\phi := \{q \in Q^\phi \mid \phi \in q\}$*;*
- $\mathcal{F}^\phi := \{F_{\psi_1 \mathsf{U} \psi_2} \mid \psi_1 \mathsf{U} \psi_2$ *is a sub-formula of* $\phi\}$ *where*

$$F_{\psi_1 \mathsf{U} \psi_2} := \{q \in Q^\phi \mid \psi_1 \mathsf{U} \psi_2 \notin q \text{ or } \psi_2 \in q\}.$$

The following proposition states in which sense this construction of $\mathcal{G}_\phi$ is correct. Its proof is similarly to the one for correctness of the automaton construction for propositional LTL [16, 15].

Proposition 1. *For every infinite word* $w \in (\Sigma^\phi)^\omega$*, we have* $w \in L_\omega(\mathcal{G}_\phi)$ *iff there exists an* $\mathcal{ALC}$*-LTL structure* $\mathfrak{I}$ *such that* $\tau_\phi(\mathfrak{I}) = w$ *and* $\mathfrak{I}, 0 \models \phi$.

As an immediate consequence of this proposition, we obtain that the $\mathcal{ALC}$-LTL formula ϕ is satisfiable iff $L_\omega(\mathcal{G}_\phi) \neq \emptyset$. Thus, we have reduced the satisfiability problem in $\mathcal{ALC}$-LTL (without rigid names) to the emptiness problem for generalized Büchi automata. It remains to analyze the complexity of the decision procedure for satisfiability obtained by this reduction.

The size of the automaton $\mathcal{G}_\phi$ is obviously exponential in ϕ. In addition, this automaton can be computed in exponential time. Indeed, to compute the set Σ^ϕ, we consider all the exponentially many subsets of the set of ϕ-literals. Each such set T has a size that is polynomial in the size of ϕ. The only non-trivial test needed to check whether T is an $\mathcal{ALC}$-type for ϕ is the consistency test for $\mathcal{B}_T$. Since the consistency problem for Boolean $\mathcal{ALC}$-knowledge bases is ExpTime-complete [3], this test can be performed in exponential time. A similar argument can be used to show that Q^ϕ can be computed in exponential time. Obviously,

given the exponentially large sets Σ^ϕ and Q^ϕ, the remaining components of $\mathcal{G}_\phi$ can also be computed in exponential time.

Since the emptiness problem for generalized Büchi automata can be solved in polynomial time, this yields an alternative proof for the fact (originally shown in [3]) that satisfiability of $\mathcal{ALC}$-LTL formulae (without rigid names) can be decided in exponential time.

3.2 The case with rigid names

If rigid concept and role names must be taken into account, Proposition 1 is not sufficient to reduce satisfiability of ϕ w.r.t. rigid names to the emptiness problem for $\mathcal{G}_\phi$. The proposition says that, for any infinite word $T_0T_1T_2\ldots \in L_\omega(\mathcal{G}_\phi)$, there is a model $\mathfrak{I} = (\mathcal{I}_i)_{i=0,1,\ldots}$ of ϕ with $\tau_\phi(\mathcal{I}_i) = T_i$ for $i \geq 0$. However, without additional precautions, there is no guarantee that the $\mathcal{ALC}$-interpretations $\mathcal{I}_i$ interpret the rigid concept and role names in the same way. In order to enforce this, the automaton has to keep track of which $\mathcal{ALC}$-types it has already read, and check the set of these types for consistency w.r.t. rigid names.

Definition 10. *The set $\mathcal{T} = \{T_1, \ldots, T_k\}$ of $\mathcal{ALC}$-types for ϕ is* r-consistent *iff there are $\mathcal{ALC}$-interpretations $\mathcal{I}_1, \ldots, \mathcal{I}_k$ that share the same domain, coincide on the rigid concept and role names, and satisfy $\tau_\phi(\mathcal{I}_i) = T_i$ for $i = 1, \ldots, k$.*

The r-consistency of a set of $\mathcal{ALC}$-types can be decided using the renaming technique for flexible symbols introduced in [3]. Given a set $\mathcal{T} = \{T_1, \ldots, T_k\}$ of $\mathcal{ALC}$-types for ϕ, we introduce renamed variants $A^{(i)}$ and $r^{(i)}$ $(i = 1, \ldots, k)$ for every *flexible* concept name A and every *flexible* role name r. For a ϕ-literal α, its renamed variant $\alpha^{(i)}$ is obtained by replacing the flexible concept and role names occurring in α by the corresponding renamed variants. The following proposition is an easy consequence of the proof of Lemma 10 in [3].

Proposition 2. *Let $\mathcal{T} = \{T_1, \ldots, T_k\}$ be a set of $\mathcal{ALC}$-types for ϕ. Then $\mathcal{T}$ is r-consistent iff the Boolean $\mathcal{ALC}$-knowledge base $\mathcal{B}_\mathcal{T}$ is consistent, where*

$$\mathcal{B}_\mathcal{T} := \bigwedge_{i=1,\ldots,k} \bigwedge_{\alpha \in T_i} \alpha^{(i)}.$$

The set of all r-consistent sets of $\mathcal{ALC}$-types for ϕ will be denoted by $\mathcal{C}_r^\phi$.

The automaton $\widehat{\mathcal{G}}_\phi$ that also takes care of rigid names has tuples (q_1, q_2) as states, where q_1 is a state of $\mathcal{G}_\phi$ and q_2 is an r-consistent set of $\mathcal{ALC}$-types for ϕ. In the first component, $\widehat{\mathcal{G}}_\phi$ works like $\mathcal{G}_\phi$, and in the second it simply collects all the $\mathcal{ALC}$-types it has read. The fact that the set in the second component must be r-consistent ensures that the semantics of rigid names is taken into account.

Definition 11. *For an $\mathcal{ALC}$-LTL formula ϕ with rigid names, the corresponding generalized Büchi automaton $\widehat{\mathcal{G}}_\phi = (\widehat{Q}^\phi, \Sigma^\phi, \widehat{\Delta}^\phi, \widehat{Q}_0^\phi, \widehat{\mathcal{F}}^\phi)$ is defined as follows:*

- $\widehat{Q}^\phi := Q^\phi \times \mathcal{C}_r^\phi$;

– $\widehat{\Delta}^\phi \subseteq \widehat{Q}^\phi \times \Sigma^\phi \times \widehat{Q}^\phi$ *is defined as follows:* $((q_1, q_2), \sigma, (q_1', q_2')) \in \widehat{\Delta}^\phi$ *iff* $(q_1, \sigma, q_1') \in \Delta^\phi$ *and* $q_2' = q_2 \cup \{\sigma\}$*;*
– $\widehat{Q}_0^\phi := \{(q_1, \emptyset) \mid q_1 \in Q_0^\phi\}$*;*
– $\widehat{\mathcal{F}}^\phi := \{F \times \mathcal{C}_r^\phi \mid F \in \mathcal{F}^\phi\}$.

The following proposition states in which sense the construction of $\widehat{\mathcal{G}}_\phi$ is correct. It is an easy consequence of Proposition 1 and the definition of r-consistency.

Proposition 3. *For every infinite word* $w \in (\Sigma^\phi)^\omega$ *we have* $w \in L_\omega(\widehat{\mathcal{G}}_\phi)$ *iff there exists an* $\mathcal{ALC}$*-LTL structure* $\mathfrak{I}$ respecting rigid names *such that* $\tau_\phi(\mathfrak{I}) = w$ *and* $\mathfrak{I}, 0 \models \phi$*.*

As an immediate consequence of this proposition, we obtain that the $\mathcal{ALC}$-LTL formula ϕ is satisfiable w.r.t. rigid names iff $L_\omega(\widehat{\mathcal{G}}_\phi) \neq \emptyset$. Thus, we have reduced the satisfiability problem w.r.t. rigid names in $\mathcal{ALC}$-LTL to the emptiness problem for generalized Büchi automata. However, the complexity of this reduction is higher than for the case of satisfiability without rigid names.

The size of the automaton $\widehat{\mathcal{G}}_\phi$ is double-exponential in the size of ϕ. In fact, the set $\mathcal{C}_r^\phi$ of all r-consistent sets of $\mathcal{ALC}$-types for ϕ may contain double-exponentially many elements since there are exponentially many $\mathcal{ALC}$-types for ϕ. Each element of $\mathcal{C}_r^\phi$ may be of exponential size.

Next, we show that the automaton $\widehat{\mathcal{G}}_\phi$ can be computed in double-exponential time. In addition to computing $\mathcal{G}_\phi$, i.e., the automaton working in the first component of $\widehat{\mathcal{G}}_\phi$, one must also compute the set $\mathcal{C}_r^\phi$. For this, one considers all sets of $\mathcal{ALC}$-types for ϕ. There are double-exponentially many such sets, each of size at most exponential in the size of ϕ. By Proposition 2, testing such a set $\mathcal{T}$ for r-consistency amounts to testing the Boolean $\mathcal{ALC}$-knowledge base $\mathcal{B}_\mathcal{T}$ for consistency. Since the size of $\mathcal{B}_\mathcal{T}$ is exponential in the size of ϕ and the consistency problem for Boolean $\mathcal{ALC}$-knowledge bases is ExpTime-complete, this test can be performed in double-exponential time. Overall, the computation of $\mathcal{C}_r^\phi$ requires double-exponentially many tests each requiring double-exponential time. This shows that $\mathcal{C}_r^\phi$, and thus also $\widehat{\mathcal{G}}_\phi$, can be computed in double-exponential time.

Since the emptiness problem for generalized Büchi automata can be solved in polynomial time, this yields an alternative proof for the fact (originally shown in [3]) that satisfiability w.r.t. rigid names in $\mathcal{ALC}$-LTL can be decided in double-exponential time.

4 The monitor construction

The construction of the monitor is basically identical for the two cases (without rigid names, with rigid names) considered in the previous section since it only depends on the properties of the automata $\mathcal{G}_\phi$ and $\widehat{\mathcal{G}}_\phi$ respectively stated in Proposition 1 and Proposition 3, and not on the actual definitions of these automata. For this reason, we treat only the more complex case with rigid names in detail. However, we distinguish between two cases according to whether the monitor has complete or incomplete knowledge about the current state of the system.

4.1 The case of complete knowledge

In this case, it is assumed that, at every time point, the monitor has complete information about the status of every $\mathcal{ALC}$-axiom occurring in the formula ϕ. To be more precise, assume that $\mathcal{I}_i$ is the $\mathcal{ALC}$-interpretation at time point i. Then the monitor receives its $\mathcal{ALC}$-type $\tau_\phi(\mathcal{I}_i)$ as input at this time point.

Before showing how a monitor for an $\mathcal{ALC}$-LTL formula ϕ can actually be constructed, let us first define how we expect it to behave. As mentioned in the introduction, such a monitor is a deterministic Moore automaton.

Definition 12. *A* deterministic Moore automaton $\mathcal{M} = (S, \Sigma, \delta, s_0, \Gamma, \lambda)$ *consists of a finite set of states S, a finite input alphabet Σ, a transition function $\delta : S \times \Sigma \to S$, an initial state $s_0 \in S$, a finite output alphabet Γ, and an output function $\lambda : S \to \Gamma$.*

The transition function and the output function can be extended to functions $\widehat{\delta} : S \times \Sigma^* \to Q$ and $\widehat{\lambda} : \Sigma^* \to \Gamma$ as follows:

- $\widehat{\delta}(s, \varepsilon) := s$ *where ε denotes the empty word;*
- $\widehat{\delta}(s, u\sigma) := \delta(\widehat{\delta}(s, u), \sigma)$ *where $u \in \Sigma^*$ and $\sigma \in \Sigma$;*

and $\widehat{\lambda}(u) := \lambda(\widehat{\delta}(s_0, u))$ for every $u \in \Sigma^*$.

Given a finite sequence $\tilde{\mathfrak{I}} = \mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_t$ of $\mathcal{ALC}$-interpretations, we say that it *respects rigid names* if these interpretations share the same domain and coincide on the rigid concept and role names. The *ϕ-type of $\tilde{\mathfrak{I}}$* is defined as $\tau_\phi(\tilde{\mathfrak{I}}) := \tau_\phi(\mathcal{I}_0) \ldots \tau_\phi(\mathcal{I}_t)$. We say that the $\mathcal{ALC}$-LTL structure $\mathfrak{J} = (\mathcal{J}_i)_{i=0,1,\ldots}$ *extends* $\tilde{\mathfrak{I}}$ iff $\mathcal{I}_i = \mathcal{J}_i$ for $i = 0, \ldots, t$. In principle, a monitor for ϕ needs to realize the following monitoring function m_ϕ:

$$m_\phi(\tilde{\mathfrak{I}}) := \begin{cases} \top & \text{if } \mathfrak{J}, 0 \models \phi \text{ for all } \mathcal{ALC}\text{-LTL structures } \mathfrak{J} \\ & \text{that respect rigid names and extend } \tilde{\mathfrak{I}}, \\ \bot & \text{if } \mathfrak{J}, 0 \models \neg\phi \text{ for all } \mathcal{ALC}\text{-LTL structures } \mathfrak{J} \\ & \text{that respect rigid names and extend } \tilde{\mathfrak{I}}, \\ ? & \text{otherwise.} \end{cases}$$

Definition 13. *Let ϕ be an $\mathcal{ALC}$-LTL formula. The deterministic Moore automaton $\mathcal{M} = (S, \Sigma^\phi, \delta, s_0, \{\top, \bot, ?\}, \lambda)$ is a* monitor w.r.t. rigid names *for ϕ iff for all finite sequences $\tilde{\mathfrak{I}}$ of $\mathcal{ALC}$-interpretations respecting rigid names we have $\widehat{\lambda}(\tau_\phi(\tilde{\mathfrak{I}})) = m_\phi(\tilde{\mathfrak{I}})$.*

Intuitively, this definition assumes that the system observed by the monitor respects rigid names, i.e., its states are $\mathcal{ALC}$-interpretations over the same domain and these interpretations coincide on the rigid concept and role names. At every time point, the monitor sees the $\mathcal{ALC}$-type of the current interpretation. Thus, if the states that the system successively entered up to time point t were $\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_t$, then the monitor has received the word $\tau_\phi(\mathcal{I}_0) \ldots \tau_\phi(\mathcal{I}_t)$ over Σ^ϕ as input. The monitor now needs to tell (by its output) whether all possible

extensions of the observed behavior satisfy ϕ (output $\top$, which says that the property ϕ will definitely be satisfied by this run of the system) or $\neg\phi$ (output $\bot$, which says that the property ϕ will definitely be violated by this run of the system); if neither is the case, then both satisfaction and violation are still possible, depending on the future behavior of the system. Since we assume that the system respects rigid names, only finite sequences $\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_t$ that respect rigid names can actually be observed, and thus Definition 13 does not formulate any requirements for sequences not satisfying this restriction. Likewise, only $\mathcal{ALC}$-LTL structures respecting rigid names need to be considered as possible extensions in the definition of m_ϕ.

We will show now that a monitor w.r.t. rigid names for ϕ can in principle be obtained by first making the automata $\widehat{\mathcal{G}}_\phi$ and $\widehat{\mathcal{G}}_{\neg\phi}$ (viewed as automata working on *finite* words) deterministic and then building the product automaton of the deterministic automata obtained this way. The output for each state of this product automaton is determined through emptiness tests for generalized Büchi automata derived from $\widehat{\mathcal{G}}_\phi$ and $\widehat{\mathcal{G}}_{\neg\phi}$ by varying the initial states. If q is a state of $\widehat{\mathcal{G}}_\phi$ ($\widehat{\mathcal{G}}_{\neg\phi}$), then $\widehat{\mathcal{G}}^q_\phi$ ($\widehat{\mathcal{G}}^q_{\neg\phi}$) denotes the generalized Büchi automaton obtained from $\widehat{\mathcal{G}}_\phi$ ($\widehat{\mathcal{G}}_{\neg\phi}$) by replacing its set of initial states with the singleton set $\{q\}$. Note that $\widehat{\mathcal{G}}_\phi$ and $\widehat{\mathcal{G}}_{\neg\phi}$ are actually automata over the same alphabet (i.e., $\Sigma^\phi = \Sigma^{\neg\phi}$) since ϕ and $\neg\phi$ obviously contain the same $\mathcal{ALC}$-axioms.

Definition 14. *Let ϕ be an $\mathcal{ALC}$-LTL formula with rigid names, and let $\widehat{\mathcal{G}}_\phi = (\widehat{Q}^\phi, \Sigma^\phi, \widehat{\Delta}^\phi, \widehat{Q}^\phi_0, \widehat{\mathcal{F}}^\phi)$ be the generalized Büchi automaton corresponding to ϕ and $\widehat{\mathcal{G}}_{\neg\phi} = (\widehat{Q}^{\neg\phi}, \Sigma^{\neg\phi}, \widehat{\Delta}^{\neg\phi}, \widehat{Q}^{\neg\phi}_0, \widehat{\mathcal{F}}^{\neg\phi})$ be the generalized Büchi automaton corresponding to $\neg\phi$.*

The deterministic Moore automaton $\mathcal{M}_\phi = (S, \Sigma^\phi, \delta, s_0, \{\top, \bot, ?\}, \lambda)$ is defined as follows:

- $S := 2^{\widehat{Q}^\phi} \times 2^{\widehat{Q}^{\neg\phi}}$
- $s_0 := (\widehat{Q}^\phi_0, \widehat{Q}^{\neg\phi}_0)$
- *For all $(P_1, P_2) \in S$ and $\sigma \in \Sigma^\phi$, we have $\delta((P_1, P_2), \sigma) := (P'_1, P'_2)$, where:*
  - $P'_1 = \bigcup_{q_1 \in P_1} \{q'_1 \in \widehat{Q}^\phi \mid (q_1, \sigma, q'_1) \in \widehat{\Delta}^\phi\}$
  - $P'_2 = \bigcup_{q_2 \in P_2} \{q'_2 \in \widehat{Q}^{\neg\phi} \mid (q_2, \sigma, q'_2) \in \widehat{\Delta}^{\neg\phi}\}$
- $\lambda : Q \to \{\top, \bot, ?\}$ *is defined as*

$$\lambda((P_1, P_2)) := \begin{cases} \top & \text{if } (i)\, L_\omega(\widehat{\mathcal{G}}^{q_2}_{\neg\phi}) = \emptyset \text{ for all } q_2 \in P_2 \text{ and} \\ & \quad (ii)\, L_\omega(\widehat{\mathcal{G}}^{q_1}_{\phi}) \neq \emptyset \text{ for some } q_1 \in P_1 \\ \bot & \text{if } (i)\, L_\omega(\widehat{\mathcal{G}}^{q_1}_{\phi})) = \emptyset \text{ for all } q_1 \in P_1 \text{ and} \\ & \quad (ii)\, L_\omega(\widehat{\mathcal{G}}^{q_2}_{\neg\phi}) \neq \emptyset \text{ for some } q_2 \in P_2 \\ ? & \text{otherwise} \end{cases}$$

Note that the conditions (ii) are necessary to have a unique output for every state of $\mathcal{M}_\phi$, and not just for the ones reachable from s_0. In fact, for the reachable ones, condition (i) implies condition (ii) (see the third and fourth item

in Lemma 1 below). Let $\mathcal{M}_\phi = (S, \Sigma^\phi, \delta, s_0, \{\top, \bot, ?\}, \lambda)$ be the deterministic Moore automaton defined above. Given a state $s \in S$, we define $\widehat{\delta}_1(s)$ to be the first and $\widehat{\delta}_2(s)$ to be the second component of $\widehat{\delta}(s)$. The following lemma easily follows from Proposition 3.

Lemma 1. *Let $\tilde{\mathfrak{I}}$ be a finite sequence of $\mathcal{ALC}$-interpretations respecting rigid names. Then the following equivalences hold:*

- $m_\phi(\tilde{\mathfrak{I}}) \neq \bot$ *iff there exists* $q_1 \in \delta_1(s_0, \tau_\phi(\tilde{\mathfrak{I}}))$ *such that* $L_\omega(\widehat{\mathcal{G}}_\phi^{q_1}) \neq \emptyset$*;*
- $m_\phi(\tilde{\mathfrak{I}}) \neq \top$ *iff there exists* $q_2 \in \delta_2(s_0, \tau_\phi(\tilde{\mathfrak{I}}))$ *such that* $L_\omega(\widehat{\mathcal{G}}_{\neg\phi}^{q_2}) \neq \emptyset$*;*
- $L_\omega(\widehat{\mathcal{G}}_\phi^{q_1}) = \emptyset$ *for all* $q_1 \in \delta_1(s_0, \tau_\phi(\tilde{\mathfrak{I}}))$ *implies that there exists* $q_2 \in \delta_2(s_0, \tau_\phi(\tilde{\mathfrak{I}}))$ *with* $L_\omega(\widehat{\mathcal{G}}_{\neg\phi}^{q_2}) \neq \emptyset$*;*
- $L_\omega(\widehat{\mathcal{G}}_{\neg\phi}^{q_2}) = \emptyset$ *for all* $q_2 \in \delta_2(s_0, \tau_\phi(\tilde{\mathfrak{I}}))$ *implies that there exists* $q_1 \in \delta_1(s_0, \tau_\phi(\tilde{\mathfrak{I}}))$ *such that* $L_\omega(\widehat{\mathcal{G}}_\phi^{q_1}) \neq \emptyset$*.*

The next theorem shows that this construction really yields a monitor according to Definition 13. Its proof is an easy consequence of Lemma 1.

Theorem 1. *The deterministic Moore automaton $\mathcal{M}_\phi$ introduced in Definition 14 is a monitor w.r.t. rigid names for ϕ.*

Since the size of the generalized Büchi automata $\widehat{\mathcal{G}}_\phi$ and $\widehat{\mathcal{G}}_{\neg\phi}$ is double-exponential in the size of ϕ, the size of the monitor $\mathcal{M}_\phi$ is triple-exponential in the size of ϕ, and it is easy to see that $\mathcal{M}_\phi$ can actually be computed in triple-exponential time.

4.2 The case of incomplete knowledge

Instead of presupposing that, at every time point, the monitor has complete information about the status of every $\mathcal{ALC}$-axiom occurring in the formula ϕ, we now assume that the monitor receives incomplete information about the states of the system at different time points in the form of $\mathcal{ALC}$-ABoxes.[5] Given such an ABox $\mathcal{A}$ and an $\mathcal{ALC}$-axiom α occurring in ϕ, there are now three possible cases: (i) $\mathcal{A}$ implies α; (ii) $\mathcal{A}$ implies $\neg\alpha$; (iii) $\mathcal{A}$ implies neither α nor $\neg\alpha$. Under the assumption that all we know about the current state $\mathcal{I}$ of the system is that it is a model of $\mathcal{A}$, then in the third case we do not know whether α or $\neg\alpha$ holds in $\mathcal{I}$. This adds an additional source of uncertainty, which may cause the monitor to answer with ?.

In the following, we thus assume that the input alphabet $\widehat{\Sigma}$ for our monitor consists of all consistent $\mathcal{ALC}$-ABoxes. Formally speaking, a monitor over this alphabet can no longer be a deterministic Moore automaton since we have required the alphabet of such an automaton to be finite. It should be clear, however, that Definition 12 can trivially be extended to cover also the case of an infinite input alphabet. From a practical point of view, this means, of course, that one cannot

[5] Instead of ABoxes we could also use arbitrary Boolean $\mathcal{ALC}$-knowledge bases here.

precompute such an infinite monitor. Instead, one precomputes only the states of the monitor. Given such a state and an input ABox, one then needs to compute the transition (i.e., the successor state in the monitor) on-the-fly.

Before constructing the actual monitor, we formally define how we expect it to behave. Given a (finite) word $\widehat{w}$ over $\widehat{\Sigma}$, i.e., a finite sequence of $\mathcal{ALC}$-ABoxes $\widehat{w} = \mathcal{A}_0, \ldots, \mathcal{A}_t$, the finite sequence $\tilde{\mathfrak{I}} = \mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_t$ of $\mathcal{ALC}$-interpretations is called a *model* of $\widehat{w}$ (written $\tilde{\mathfrak{I}} \models \widehat{w}$) iff $\mathcal{I}_i$ is a model of $\mathcal{A}_i$ for $i = 1, \ldots, t$. If $\tilde{\mathfrak{I}}$ additionally respects rigid names, then we say that it is a *model w.r.t. rigid names*. The monitor w.r.t. partial knowledge for ϕ needs to realize the following monitoring function $\widehat{m}_\phi : \widehat{\Sigma}^* \to \{\top, \bot, ?\}$:

$$\widehat{m}_\phi(\widehat{w}) := \begin{cases} \top & \text{if } m_\phi(\tilde{\mathfrak{I}}) = \top \text{ for all models w.r.t. rigid names of } \widehat{w}, \\ \bot & \text{if } m_\phi(\tilde{\mathfrak{I}}) = \bot \text{ for all models w.r.t. rigid names of } \widehat{w}, \\ ? & \text{otherwise.} \end{cases}$$

Definition 15. *Let ϕ be an $\mathcal{ALC}$-LTL formula. The deterministic Moore automaton $\mathcal{M} = (S, \widehat{\Sigma}, \delta, s_0, \{\top, \bot, ?\}, \lambda)$ is a* monitor w.r.t. rigid names and incomplete knowledge *for ϕ iff for all finite sequences $\widehat{w}$ of $\mathcal{ALC}$-ABoxes, we have $\widehat{\lambda}(\widehat{w}) = \widehat{m}_\phi(\widehat{w})$.*

The monitor w.r.t. rigid names and incomplete knowledge for ϕ constructed in the following is almost identical to the monitor w.r.t. rigid names $\mathcal{M}_\phi$ constructed in the previous subsection. The only difference can be found in the definition of the transition functions. In $\mathcal{M}_\phi$, transitions are of the form $\delta(s, \sigma) = s'$ where σ is an $\mathcal{ALC}$-type for ϕ. In the monitor w.r.t. incomplete knowledge, the input symbol is a consistent $\mathcal{ALC}$-ABox $\mathcal{A}$ instead of an $\mathcal{ALC}$-type. Intuitively, $\mathcal{A}$ stands for all its models since all we know about the current state of the system is that it is a model of $\mathcal{A}$. The monitor must consider all the transitions (in the generalized Büchi automata $\widehat{\mathcal{G}}_\phi$ and $\widehat{\mathcal{G}}_{\neg\phi}$ for ϕ and $\neg\phi$) that can be induced by such models. To be more precise, the transitions in $\widehat{\mathcal{G}}_\phi$ and $\widehat{\mathcal{G}}_{\neg\phi}$ are made with the ϕ-types of these models as input symbols. In the following definition, we use the fact that σ is the ϕ-type of a model of $\mathcal{A}$ iff $\mathcal{A} \wedge \bigwedge_{\alpha \in \sigma} \alpha$ is consistent.

Definition 16. *Let ϕ be an $\mathcal{ALC}$-LTL formula with rigid names, and let $\widehat{\mathcal{G}}_\phi = (\widehat{Q}^\phi, \Sigma^\phi, \widehat{\Delta}^\phi, \widehat{Q}_0^\phi, \widehat{\mathcal{F}}^\phi)$ be the generalized Büchi automaton corresponding to ϕ and $\widehat{\mathcal{G}}_{\neg\phi} = (\widehat{Q}^{\neg\phi}, \Sigma^{\neg\phi}, \widehat{\Delta}^{\neg\phi}, \widehat{Q}_0^{\neg\phi}, \widehat{\mathcal{F}}^{\neg\phi})$ be the generalized Büchi automaton corresponding to $\neg\phi$.*

The deterministic Moore automaton $\mathcal{M}_\phi^{inc} = (S, \widehat{\Sigma}, \delta^{inc}, s_0, \{\top, \bot, ?\}, \lambda)$ is defined as follows:

- *S, s_0, and λ are defined as in Definition 14*
- *For all $(P_1, P_2) \in S$ and $\mathcal{A} \in \widehat{\Sigma}$, we have $\delta^{inc}((P_1, P_2), \sigma) := (P_1', P_2')$, where:*

$$P_1' = \bigcup_{\sigma \in \Sigma^\phi} \bigcup_{q_1 \in P_1} \{q_1' \in \widehat{Q}^\phi \mid (q_1, \sigma, q_1') \in \widehat{\Delta}^\phi \text{ and } \mathcal{A} \wedge \textstyle\bigwedge_{\alpha \in \sigma} \alpha \text{ is consistent}\}$$

$$P_2' = \bigcup_{\sigma \in \Sigma^\phi} \bigcup_{q_2 \in P_2} \{q_2' \in \widehat{Q}^{\neg\phi} \mid (q_2, \sigma, q_2') \in \widehat{\Delta}^{\neg\phi} \text{ and } \mathcal{A} \wedge \textstyle\bigwedge_{\alpha \in \sigma} \alpha \text{ is consistent}\}$$

Lemma 1 from the previous subsection can also be used to show correctness of the monitor construction in the case of incomplete knowledge.

Theorem 2. *The deterministic Moore automaton $\mathcal{M}_\phi^{inc}$ introduced in Definition 16 is a monitor w.r.t. rigid names and incomplete knowledge for ϕ.*

Since the input alphabet $\widehat{\Sigma}$ of $\mathcal{M}_\phi^{inc}$ is infinite, it only makes sense to measure the size of $\mathcal{M}_\phi^{inc}$ in terms of the size of its set of states. This set is identical to the set of states S of $\mathcal{M}_\phi$, and we have already seen that S is of triple-exponential size. Regarding the on-the-fly computation of the transitions in $\mathcal{M}_\phi^{inc}$, for a given input ABox $\mathcal{A}$, one needs to consider the exponentially many $\mathcal{ALC}$-types for ϕ, and, for each such type σ, check the Boolean $\mathcal{ALC}$-knowledge base $\mathcal{A} \wedge \bigwedge_{\alpha \in \sigma} \alpha$ for consistency. This test is exponential in the size of this knowledge base, and thus exponential in the size of ϕ and in the size of $\mathcal{A}$.

5 Conclusion

We have shown that the three-valued approach to runtime verification in propositional LTL [5] can be extended to $\mathcal{ALC}$-LTL and the case where states of the observed system may be described in an incomplete way by $\mathcal{ALC}$-ABoxes. The complexity of the monitor construction is quite high. We have seen that the size of our monitors is triple-exponential in the size of the formula ϕ.[6] However, the size of the formula is usually quite small, whereas the system is monitored over a long period of time. If we assume the size of the formula to be constant (an assumption often made in model checking and runtime verification), then our monitor works in time linear in the length of the observed prefix. Moreover, each input symbol (i.e., $\mathcal{ALC}$-type or consistent ABox) can be processed in constant time.

It should also be noted that the triple-exponential complexity of the monitor construction is a worst-case complexity. Minimization of the intermediate generalized Büchi automata and the monitor may lead to much smaller automata than the ones defined above. We have observed this behavior on several small example formulae. A more thorough empirical evaluation will be part of our future research.

From a worst-case complexity point of view, the large size of the monitor can probably not be avoided. In fact, the complexity lower bounds for the satisfiability problem in $\mathcal{ALC}$-LTL (ExpTime-hard without rigid names and 2-ExpTime-hard with rigid names) imply that our construction of the generalized Büchi automata $\mathcal{G}_\phi$ and $\widehat{\mathcal{G}}_\phi$ is optimal. Regarding complexity lower bounds for the size of the monitor, it is known [13, 12] that, in the case of propositional LTL, the monitor must in general be of size at least exponential in the size of the formula. However, the constructions in the literature [5] actually yield monitors of double-exponential size, i.e., one exponential higher than the size of the generalized Büchi automata for propositional LTL.

[6] For the case without rigid names it is "only" double-exponential since the generalized Büchi automata are then smaller.

References

1. Alessandro Artale and Enrico Franconi. A survey of temporal extensions of description logics. *Ann. of Mathematics and Artificial Intelligence*, 30:171–210, 2000.
2. Alessandro Artale and Enrico Franconi. Temporal description logics. In D. Gabbay, M. Fisher, and L. Vila, editors, *Handbook of Time and Temporal Reasoning in Artificial Intelligence*. The MIT Press, 2001.
3. Franz Baader, Silvio Ghilardi, and Carsten Lutz. LTL over description logic axioms. In Gerhard Brewka and Jérôme Lang, editors, *Proc. of the 11th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 684–694. Morgan Kaufmann, Los Altos, 2008.
4. Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, Cambridge, Massachusetts, 2008.
5. Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In S. Arun-Kumar and N. Garg, editors, *FSTTCS '06: Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 4337 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, December 2006. Springer-Verlag.
6. Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing LTL semantics for runtime verification. *Journal of Logic and Computation*, 2009.
7. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
8. Séverine Colin and Leonardo Mariani. Run-time verification. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*, pages 525–555. Springer, 2004.
9. Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification XV*, pages 3–18, London, UK, 1996. Chapman & Hall, Ltd.
10. Carsten Lutz, Frank Wolter, and Michael Zakharyaschev. Temporal description logics: A survey. In Stéphane Demri and Christian S. Jensen, editors, *Proc. of the 15th Int. Symp. on Temporal Representation and Reasoning (TIME'08)*, pages 3–14. IEEE Computer Society Press, 2008.
11. Amir Pnueli. The temporal logic of programs. In *Proc. of the 18th Annual Symp. on the Foundations of Computer Science (FOCS'77)*, pages 46–57, 1977.
12. Grigore Roşu. On safety properties and their monitoring. Technical Report UIUCDCS-R-2007-2850, Department of Computer Science, University of Illinois at Urbana-Champaign, 2007.
13. Grigore Roşu and Klaus Havelund. Rewriting-based techniques for runtime verification. *Automated Software Engineering*, 12(2):151–197, 2005.
14. Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
15. Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.
16. Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths. In *Proc. of the 24th Annual Symp. on the Foundations of Computer Science (FOCS'83)*, pages 185–194. IEEE Computer Society Press, 1983.