TECHNISCHE UNIVERSITÄT DRESDEN

Technische Universität Dresden
Institute for Theoretical Computer Science
Chair for Automata Theory

LTCS–Report

Runtime Verification Using a Temporal Description Logic Revisited

Franz Baader Marcel Lippmann

LTCS-Report 14-01

Runtime Verification Using a Temporal Description Logic Revisited[*]

Franz Baader and Marcel Lippmann

Institute of Theoretical Computer Science
Technische Universität Dresden, Germany
`{baader,lippmann}@tcs.inf.tu-dresden.de`

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.

[*]Partially supported by DFG SFB 912 (HAEC) and GRK 1763 (QuantLA).

Contents

1 Introduction

Formulae of linear temporal logic (LTL) [Pnu77] 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, $\mathsf{turn\_off}$, and $\mathsf{turn\_on}$, 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

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

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 [CGP99, BK08], 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 [CM04], 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

[1]There are also variants of runtime verification for propositional LTL that work with only two or even four possible answers [BLS10].

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 [BLS06, BLS11]. 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 [BGL08, BGL12].[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 $\mathcal{ALC}$ [SSS91]) are used in place of propositional variables. 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 [BGL08, BGL12], 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 [BGL08, BGL12] are not based on 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

[2]A Büchi automaton for an LTL-formula ψ accepts the LTL structures satisfying this formula, viewed as infinite words over an appropriate alphabet [WVS83, BK08].

[3]A comparison of $\mathcal{ALC}$-LTL with other temporal DLs [AF00, AF01, LWZ08] is beyond the scope of this introduction. It can be found in [BGL08, BGL12].

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 behaviour, 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 behaviour at a time point. Technically, this means that we assume that the prefix describing the system's behaviour is a finite sequence of of so-called ABoxes (i.e., sets assertions that state that certain individuals belong to certain concepts, or are related to each other via certain roles). 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 ?.

Adding additional background knowledge about the working of the system can reduce the number of such indefinite answers ?. This background knowledge could, for example, be a global TBox, i.e., a finite set of terminological axioms that are known to hold for every state of the system. Rigidity of concepts or roles also constitutes such background knowledge since it tells us that the system does not change the interpretation of these concepts and roles during its run. In the presence of background knowledge, it is convenient to extend the range of answers to a runtime verification problem to a fourth value $\lightning$, which indicates that the prefix seen so far violates the background knowledge. In practice, this value should not be encountered since it is assumed that the observed system actually respects rigid concepts and roles and satisfies the background knowledge. Thus, no finite prefix obtained by observing the system can yield this case. If it does, then the modelling of the properties of the system via background knowledge was incorrect or the sensors that generated the observations of the system were faulty.

As a possible application of monitoring $\mathcal{ALC}$-LTL-formulae with incomplete information and background knowledge, 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 point in time can be given by an ABox. The medical ontology then constitutes the background knowledge. Critical situations, which require the intervention of a doctor, can be described by an $\mathcal{ALC}$-LTL-formula (see [BGL08, BGL12] for a simple example). As long as the monitor for this

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

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.

A preliminary version of this work has previously been published at a conference [BBL09].[5] The results of the present paper improve on this preliminary work in several respects:

- Even in the propositional case, the monitor constructed using Büchi automata is in the worst-case of double-exponential size. We prove the new result that this double-exponential blowup in the construction of the monitor cannot be avoided.[6] This implies that it also cannot be avoided for $\mathcal{ALC}$-LTL.
- We have added the possibility to specify background knowledge to our framework. It should be noted that this knowledge cannot just be added to the formula to be monitored. In fact, it it supposed to hold, and thus it does not make sense to monitor whether it is satisfied or not.
- In [BBL09] we presented different ways of constructing monitors depending on (i) whether the observations of the system are complete or not, and (ii) whether rigid concepts and roles are present or not. The reason was that the worst-case size of the monitors constructed in [BBL09] was dependent on these choices. In the most general case (incomplete observations and rigid concepts and roles), the size of the constructed monitor could actually be triple-exponential in the size of the formula. Here, we only consider this most general case and show that it is always possible to construct a monitor of at most double-exponential size. Given the double-exponential lower bound mentioned above, this shows that our construction is actually optimal.
- The monitor construction (both in the present paper and in [BBL09]) depends on the construction of a Büchi automaton for a given formula. In [BBL09], we have extended one particular such construction from the propositional case to the case of $\mathcal{ALC}$-LTL. The approach used in the present paper is more modular. Basically, instead of constructing such an automaton directly for a given $\mathcal{ALC}$-LTL-formula, we build a propositional abstraction of the formula, and then reuses an arbitrary construction for the propositional case. Thus, any existing efficient tool for transforming an LTL-formula into a Büchi automaton can be used.
- Before building a monitor for an LTL-formula, it makes sense to check whether the monitor will actually be able to give reasonable answers. In

[5]This explains the word “revisited” in the title.

[6]Such a double-exponential lower bound was already claimed in [BLS11], referring to a result of Kupferman and Vardi [KV01]. However, a closer look at the relevant theorem in [KV01] shows that it only yields a lower bound of $2^{2^{\sqrt{n}}}$.

particular, a monitor that always answers ? is clearly useless. This motivates the definition of the notion of monitorability of a formula. A closely related, but simpler looking notion is the one of liveness of a formula. We extend these two notions (which were not considered in [BBL09]) from LTL to $\mathcal{ALC}$-LTL, and determine the complexity of testing liveness and monitorability of $\mathcal{ALC}$-LTL-formulae.

In the next section, we introduce the propositional temporal logic LTL, sketch the connection between LTL-formulae and Büchi automata, and define the monitoring problem for the propositional case. In addition, we prove the double-exponential lower bound for the monitor construction mentioned above, and introduce he notions "liveness" and "monitorability" for the propositional case. In Section 3 we define the the DL $\mathcal{ALC}$ and the temporal DL $\mathcal{ALC}$-LTL, and show how to construct Büchi automata for $\mathcal{ALC}$-LTL-formulae. For didactic reasons, we first present a construction for the case without rigid concepts and roles, and then show how this construction can be extended to the case where rigid concepts and roles are allowed. These Büchi automata are then used in Section 4 to construct monitors for $\mathcal{ALC}$-LTL-formulae. Finally, in Section 5 we use the constructed monitors to determine the complexity of testing liveness and monitorability of $\mathcal{ALC}$-LTL-formulae. Some concluding remarks are made in Section 6.

2 The Propositional Temporal Logic LTL

In this section, we recall the pertinent definitions for propositional linear time logic (LTL) [Pnu77]. In addition to introducing syntax and semantics of LTL, we mention the connection to Büchi automata and define the monitoring problem. LTL extends propositional logics with modal operators that can be used to talk about the past and the future.[7]

Definition 2.1. *Given a finite set $\mathcal{P} = \{p_1, \ldots, p_m\}$ of propositional variables,* LTL-formulae over $\mathcal{P}$ *are defined by induction:*

- *if $p \in \mathcal{P}$, then p is an LTL-formula over $\mathcal{P}$;*
- *if ϕ, ψ are LTL-formulae over $\mathcal{P}$, then so are $\phi \wedge \psi$ (conjunction), $\neg\phi$ (negation), $\mathsf{X}\phi$ (next), $\mathsf{X}^-\phi$ (previous), $\phi \,\mathsf{U}\, \psi$ (until), and $\phi \,\mathsf{S}\, \psi$ (since).*

[7] In the literature, variants of LTL without past operators are often considered. We use LTL with past operators since their presence often makes defining relevant properties easier [LPZ85], but neither makes our approach more complicated nor our algorithms more complex. More information on the connection between LTL with past operators and without can be found in [GPSS80, LMS02]. When giving references for existing results for LTL, we will usually cite papers that do consider LTL *with past* operators.

If the set of propositional variables is clear from the context or irrelevant, we will talk about LTL-formulae rather than LTL-formulae over $\mathcal{P}$.

The semantics of LTL is defined using the natural numbers as discrete linear flow of time. For each point in time (i.e., natural number), the semantic structure, called LTL-structure in the following, determines which of the propositional variables are true at this point.

Definition 2.2. *An* LTL-structure *over* $\mathcal{P} = \{p_1, \ldots, p_m\}$ *is an infinite sequence* $\mathfrak{W} = (w_i)_{i \geq 0}$ *of sets* $w_i \subseteq \mathcal{P}$*, which we call* worlds.

Given an LTL-formula ϕ*, an LTL structure* $\mathfrak{W} = (w_i)_{i \geq 0}$*, and a time point* $i \in \{0, 1, 2, \ldots\}$*,* validity of ϕ in $\mathfrak{W}$ at time i *(written* $\mathfrak{W}, i \models \phi$*) is defined inductively:*

$$
\begin{array}{lll}
\mathfrak{W}, i \models p_j & \textit{iff} & p_j \in w_i \\
\mathfrak{W}, i \models \phi \wedge \psi & \textit{iff} & \mathfrak{W}, i \models \phi \textit{ and } \mathfrak{W}, i \models \psi \\
\mathfrak{W}, i \models \neg\phi & \textit{iff} & \textit{not } \mathfrak{W}, i \models \phi \\
\mathfrak{W}, i \models \mathsf{X}\phi & \textit{iff} & \mathfrak{W}, i+1 \models \phi \\
\mathfrak{W}, i \models \mathsf{X}^{-}\phi & \textit{iff} & i > 0 \textit{ and } \mathfrak{W}, i-1 \models \phi \\
\mathfrak{W}, i \models \phi \,\mathsf{U}\, \psi & \textit{iff} & \textit{there is some } k \geq i \textit{ such that } \mathfrak{W}, k \models \psi \\
 & & \textit{and } \mathfrak{W}, j \models \phi \textit{ for all } j, i \leq j < k \\
\mathfrak{W}, i \models \phi \,\mathsf{S}\, \psi & \textit{iff} & \textit{there is some } k, 0 \leq k \leq i, \textit{ such that } \mathfrak{W}, k \models \psi \\
 & & \textit{and } \mathfrak{W}, j \models \phi \textit{ for all } j, k < j \leq i
\end{array}
$$

If $\mathfrak{W}, 0 \models \phi$*, then we call* $\mathfrak{W}$ *a* model *of* ϕ*. The formula* ϕ *is called* satisfiable *if it has a model.*

It is well-known that the satisfiability problem for LTL is PSpace-complete [SC85]. One possible way of showing this is to translate LTL-formulae into Büchi automata accepting their models. In general, Büchi automata accept ω-words over an alphabet Σ, i.e., infinite sequences of letters $w = \sigma_0\sigma_1\sigma_2\ldots$ with $\sigma_i \in \Sigma$. The set of all ω-words over Σ is denoted by Σ^ω, and a subset L of Σ^ω is called an *ω-language.*

Definition 2.3. *A* (non-deterministic) Büchi automaton $\mathcal{N} = (Q, \Sigma, \Delta, Q_0, F)$ *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 of initial states* $Q_0 \subseteq Q$*, and a set of final states* $F \subseteq Q$*. Such an automaton is called* deterministic *if* Δ *is a partial function, i.e., for every pair* $(q, \sigma) \in Q \times \Sigma$ *there is at most one* $q' \in Q$ *such that* $(q, \sigma, q') \in \Delta$*.*

Given an ω-word $w = \sigma_0\sigma_1\sigma_2\ldots \in \Sigma^\omega$*, a* run *of* $\mathcal{N}$ *on* w *is an ω-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, there are infinitely many* $i \geq 0$ *such that* $q_i \in F$*. The* language accepted by $\mathcal{N}$ *is defined as*

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

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

The emptiness problem for Büchi automata is known to be decidable in polynomial time [VW94].

Given an LTL-formula ϕ over $\mathcal{P} = \{p_1, \dots, p_m\}$, we can view any LTL-structure $\mathfrak{W} = (w_i)_{i \geq 0}$ as an ω-word $w = w_0 w_1 w_2 \ldots \in \Sigma_{\mathcal{P}}^{\omega}$, where the alphabet $\Sigma_{\mathcal{P}}$ consists of all subsets of $\mathcal{P}$. The idea is now to build a Büchi automaton that accepts exactly the models of ϕ.

Definition 2.4. *Let ϕ be an LTL-formula over $\mathcal{P}$ and $\mathcal{N}$ a Büchi automaton using the alphabet $\Sigma_{\mathcal{P}}$. We define*

$$L_\omega(\phi) := \{w_0 w_1 w_2 \ldots \in \Sigma_{\mathcal{P}}^{\omega} \mid \mathfrak{W} = (w_i)_{i \geq 0} \text{ is a model of } \phi\}.$$

and say that $\mathcal{N}$ is a Büchi automaton for ϕ *if $L_\omega(\mathcal{N}) = L_\omega(\phi)$.*

If $\mathcal{N}$ is a Büchi automaton for ϕ, then ϕ is satisfiable iff $L_\omega(\mathcal{N}) \neq \emptyset$. Thus, by constructing a Büchi automaton for ϕ, we can reduce the satisfiability problem in LTL to the emptiness problem for Büchi automata. It is well-known that, given an LTL-formula ϕ, one can compute a Büchi automaton for ϕ in exponential time [WVS83, VW94, GO03]. More precisely, this Büchi automaton has at most exponentially many states in the size of ϕ, but each state can be represented using only polynomial space. If we first compute the exponentially large Büchi automaton for ϕ and then apply the emptiness test for Büchi automata, then we obtain an ExpTime satisfiability procedure. In order to reduce the complexity to PSpace, one must generate the relevant parts of the automaton on-the-fly while performing the emptiness test [SC85, LPZ85]. It can be shown that, in the worst case, an exponential blow-up in the construction of the Büchi automaton for an LTL-formula cannot be avoided. However, there are optimised implementations of the construction that try to keep the number of states as small as possible (see, e.g., [GO01, GO03, GPVW96]).[8] Experiments with these implementations show that an exponential blow-up can frequently be avoided. For example, the tool *LTL2BA*[9] is widely used in practice to generate Büchi automata from propositional LTL-formulae.[10]

Example 2.5. *As an example, consider the LTL-formula $\psi_{ex} := \mathsf{X} p_1 \wedge (p_2 \,\mathsf{U}\, p_3)$. Figure 1 depicts a Büchi automaton for this formula, which was generated by LTL2BA. Note that edges with propositional formulae ϕ as labels are used as*

[8]Some of these algorithms actually generate so-called *generalised* Büchi automata rather than the normal ones we have introduced in Definition 2.3. However, it is well-known that a generalised Büchi automaton can be transformed into an equivalent normal one in polynomial time [GPVW96, BK08].

[9]See `http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/`.

[10]Unfortunately, like most other such tools, LTL2BA does *not* support LTL *with past* operators.

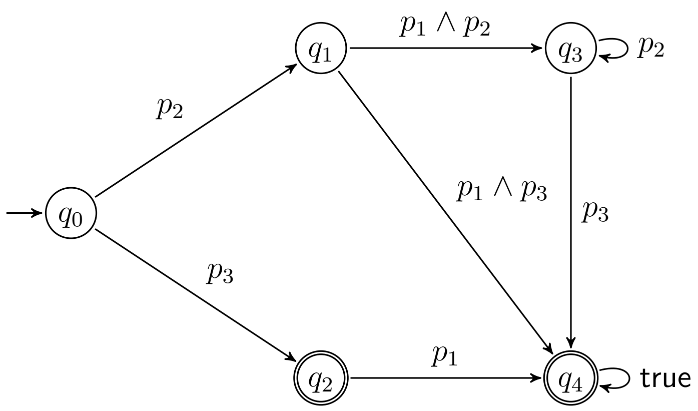

Figure 1: A Büchi automaton for ψ_{ex}

abbreviations for sets of edges labeled with those subsets of $\mathcal{P}$ that represent models of ϕ. For example, the edge with label $p_1 \wedge p_2$ from q_1 to q_3 stands for two edges between these states, one with label $\{p_1, p_2\}$ and one with label $\{p_1, p_2, p_3\}$.

In propositional *runtime verification* [BLS10, BLS11], one observes the actual behaviour of the given system since it started, which at any point in time can be described by a finite word u over $\Sigma_{\mathcal{P}}$. Here $\mathcal{P}$ is a finite set of propositional variables whose truth values at any point in time can be determined by observing the system. Given such a word $u = u_0 u_1 \dots u_t \in \Sigma_{\mathcal{P}}^*$, we say that the LTL-structure $\mathfrak{W} = (w_i)_{i \geq 0}$ *extends* u if $u_i = w_i$ for $i = 0, \dots, t$. In this case we also call $\mathfrak{W} = (w_i)_{i \geq 0}$ an *extension of* u. In principle, a monitor for an LTL-formula ϕ needs to realize the following *monitoring function* $m_\phi : \Sigma_{\mathcal{P}}^* \to \{\top, \bot, ?\}$:

$$m_\phi(u) := \begin{cases} \top & \text{if } \mathfrak{W}, 0 \models \phi \text{ for all LTL-structures } \mathfrak{W} \text{ that extend } u \\ \bot & \text{if } \mathfrak{W}, 0 \models \neg\phi \text{ for all LTL-structures } \mathfrak{W} \text{ that extend } u \\ ? & \text{otherwise.} \end{cases}$$

However, as mentioned in the introduction, this function should not be computed from scratch whenever a new observation $\sigma \in \Sigma_{\mathcal{P}}$ is added. In particular, the time needed for computing the next function value $m_\phi(u\sigma)$ should not depend on the length of the already observed word u. This can be achieved by constructing a deterministic Moore automaton as monitor.

Definition 2.6. *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 $\widehat{\delta} : S \times \Sigma^* \to S$ 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* $u \in \Sigma^*$*.*

Such a deterministic Moore automaton is a monitor for ϕ if its extended output function is the monitoring function for ϕ.

Definition 2.7. *Let $\mathcal{P}$ be a finite set of propositional variables and ϕ an LTL-formula over $\mathcal{P}$. The deterministic Moore automaton $\mathcal{M} = (S, \Sigma_{\mathcal{P}}, \delta, s_0, \{\top, \bot, ?\}, \lambda)$ is a* monitor for ϕ *if* $\widehat{\lambda}(u) = m_\phi(u)$ *holds for all* $u \in \Sigma_{\mathcal{P}}^*$*.*

Given an LTL-formula ϕ over $\mathcal{P}$, a monitor for ϕ can effectively be computed [BLS06, BLS11]. Basically, one computes Büchi automata for ϕ and $\neg\phi$, determinizes them (as finite automata rather than Büchi automata), and then builds the product of the two deterministic automata. The output function is computed using reachability tests in the Büchi automata (see [BLS06, BLS11] and the monitor construction for $\mathcal{ALC}$-LTL in Section 4 below for details). The monitor obtained this way is in the worst-case of double-exponential size and can be computed in double-exponential time.

One can actually show that this double-exponential blowup in the construction of the monitor cannot be avoided. Such a double-exponential lower bound was already claimed in [BLS11], referring to a result of Kupferman and Vardi [KV01]. However, a closer look at Theorem 3.3 in [KV01] shows that it only yields a lower bound of $2^{2^{\sqrt{n}}}$. Fortunately, a more recent result by Kupferman and Rosenberg [KR10] can be used to show a lower bound of 2^{2^n}. Since a proof of this tight lower bound for the monitor construction has, to the best of our knowledge, not yet been published, we give it here for the sake of completeness. This lower bound can also be used to show optimality of our monitor constructions for $\mathcal{ALC}$-LTL.

Kupferman and Rosenberg show (see Theorem 3 in [KR10]) that there exists a sequence $(L_n)_{n\geq 1}$ of ω-languages and a sequence $(\phi_n)_{n\geq 1}$ of LTL-formulae such that the following holds for all $n \geq 1$:

1. the language L_n can be accepted by a deterministic Büchi automaton, but the number of states of any deterministic Büchi automaton accepting L_n is at least 2^{2^n};
2. $L_n = L_\omega(\phi_n)$ and the size of ϕ_n is linear in n.

Using an argument similar to the one employed in [KR10], we can show that the number of states of any monitor for ϕ_n is at least 2^{2^n}. For this purpose, we first recall the definition of the languages L_n from [KR10].

For all $n \geq 1$, we consider the alphabet $\Sigma_n := \{a_1, \ldots, a_n\} \cup \{b_1, \ldots, b_n\} \cup \{\#, \$\}$, and define

$$\begin{array}{rcl} T_n & := & \{a_1, b_1\} \cdot \ldots \cdot \{a_n, b_n\}, \\ S_n & := & \{\#\} \cdot (T_n \cdot \{\#\})^* \cdot \{\$\} \cdot T_n \cdot \{\#\}^\omega, \\ R_n & := & \bigcup_{w \in T_n} \Sigma_n^* \cdot \{\#\} \cdot \{w\} \cdot \{\#\} \cdot \Sigma_n^* \cdot \{\$\} \cdot \Sigma_n^* \cdot \{w\} \cdot \{\#\}^\omega, \\ L_n & := & S_n \cap R_n. \end{array}$$

Thus, T_n consists of the words of length n such that the letter at position i is a_i or b_i. Obviously, there are 2^n such words. The language S_n consists of ω-words that start with a finite sequence of elements of T_n, which are separated by the #-symbol. This sequence is terminated by the \$-symbol, which is followed by exactly one element of T_n. Then comes an infinite sequence of #-symbols. Intersecting the ω-language S_n with the ω-language R_n has the following effect: it ensures that the element w of T_n that follows the \$-symbol has already occurred in the sequence of elements of T_n before the \$-symbol.

The LTL-formulae ϕ_n representing the ω-languages L_n are built over sets of propositional variables with $2n + 2$ elements, i.e., over $\mathcal{P}_n = \{p_1, \ldots, p_{2n+2}\}$. Recall that such a formula defines a language over the alphabet $\Sigma_{\mathcal{P}_n}$, whose letters are the subsets of $\mathcal{P}_n$. Of these exponentially many letters, the language L_n uses only the (linearly many) singleton sets, where

- $\{p_i\}$ represents the letter a_i for $i = 1, \ldots, n$;
- $\{p_{n+i}\}$ represents the letter b_i for $i = 1, \ldots, n$;
- $\{p_{2n+1}\}$ represents the letter # and $\{p_{2n+2}\}$ represents the letter \$.

In order to increase readability, we continue to use the letters from Σ_n rather than these singleton sets in our argument below.

Proposition 2.8. *There is a sequence $(\phi_n)_{n\geq 1}$ of LTL-formulae of size linear in n such that the number of states of any monitor for ϕ_n is at least 2^{2^n}.*

Proof. Let $(\phi_n)_{n\geq 1}$ be the sequence of LTL-formulae constructed in the proof of Theorem 3 of [KR10]. It is shown in that proof that $L_n = L_\omega(\phi_n)$ and that the size of ϕ_n is linear in n.

Now assume that $\mathcal{M}_n = (S_n, \Sigma_n, \delta_n, s_{0,n}, \{\top, \bot, ?\}, \lambda_n)$ is a monitor for ϕ_n with less than 2^{2^n} states. Given a set $\mathcal{T} \subseteq T_n$, we enumerate its elements in lexicographic order (where a_i comes before b_i). Assume that $w_1, \ldots, w_m$ is the enumeration of the elements of $\mathcal{T}$ in this order. Then we define

$$\mathsf{w}(\mathcal{T}) := \#w_1\# \ldots \#w_m\#\$.$$

Let $\mathsf{s}(\mathcal{T})$ be the state reached in $\mathcal{M}_n$ with input $\mathsf{w}(\mathcal{T})$ when starting at the initial state $s_{0,n}$. Since there are 2^{2^n} different subsets of T_n, but less than 2^{2^n} states, there must be two different such subsets $\mathcal{T}, \mathcal{T}'$ such that $\mathsf{s}(\mathcal{T}) = \mathsf{s}(\mathcal{T}')$. Without loss of generality we assume that there is a word $w \in \mathcal{T} \setminus \mathcal{T}'$.[11] Now consider the state s reached from $s_{0,n}$ on input $\mathsf{w}(\mathcal{T})w\#$. Since $\mathsf{s}(\mathcal{T}) = \mathsf{s}(\mathcal{T}')$, this is the same state as the one reached from $s_{0,n}$ on input $\mathsf{w}(\mathcal{T}')w\#$. Since $\mathcal{M}_n$ is a monitor for ϕ_n, this implies that

$$m_{\phi_n}(\mathsf{w}(\mathcal{T})w\#) = \lambda_n(s) = m_{\phi_n}(\mathsf{w}(\mathcal{T}')w\#).$$

This yields a *contradiction* since actually we have

$$m_{\phi_n}(\mathsf{w}(\mathcal{T})w\#) = ? \neq \bot = m_{\phi_n}(\mathsf{w}(\mathcal{T}')w\#).$$

In fact, we can extend $\mathsf{w}(\mathcal{T})w\#$ to an ω-word belonging to L_n (and thus satisfying ϕ_n) by adding an infinite sequence of #-symbols. Any other extension of $\mathsf{w}(\mathcal{T})w\#$ does not belong to L_n. This shows that $m_{\phi_n}(\mathsf{w}(\mathcal{T})w\#) = ?$. The word $\mathsf{w}(\mathcal{T}')w\#$ cannot be extended to an element of L_n since w does not occur in the sequence before the \$-symbol. This shows that $m_{\phi_n}(\mathsf{w}(\mathcal{T}')w\#) = \bot$.

Summing up, we have seen that our assumption that there is a monitor for ϕ_n with less than 2^{2^n} states leads to a contradiction, which shows that any monitor for ϕ_n must have at least 2^{2^n} states. $\square$

Before building a monitor for an LTL-formula ϕ, it makes sense to check whether the monitor will actually be able to give reasonable answers. For example, a monitor that always (i.e., for every finite word) returns the answer ? is clearly useless. Similarly, when running the monitor, it makes sense to check whether, according to what has been seen of the system's behaviour until now (i.e., the finite word read by the monitor until now), it makes sense to continue running the monitor. This leads to the following definition of monitorability [PZ06, FFM09, Bau10].

Definition 2.9. *Let ϕ be an LTL-formula over $\mathcal{P}$ and u a finite word over $\Sigma_{\mathcal{P}}$. We say that ϕ is u-*monitorable *if there is a finite word $v \in \Sigma_{\mathcal{P}}^*$ such that $m_\phi(uv) \neq ?$. Moreover, ϕ is called* monitorable *if it is u-monitorable for all finite words $u \in \Sigma_{\mathcal{P}}^*$.*

Given a monitor $\mathcal{M}$ for ϕ, one can easily decide monitorability through reachability tests in $\mathcal{M}$. Call a state in $\mathcal{M}$ *good* if from it one can reach a state whose output is different from ?. Then

- ϕ is u-monitorable iff the state reached from the initial state with input u is good;
- ϕ is monitorable iff every state reachable from the initial state is good.

[11] The case where $\mathcal{T}' \setminus \mathcal{T}$ is non-empty can be treated symmetrically.

This shows that monitorability can be decided in double-exponential time (in the size of the formula). More precisely, one can obtain an ExpSpace upper bound by constructing the relevant parts of the monitor on the fly while performing the reachability test (this is the same idea underlying the automata-based PSpace satisfiability test for LTL mentioned above). To the best of our knowledge, it is open whether this ExpSpace upper bound is tight. The only known lower bound is PSpace, which can be obtained using a reduction from satisfiability.[12] Interestingly, the same is true for the related, but simpler-looking problem of liveness [AS85].

Definition 2.10. *Let ϕ be an LTL-formula over $\mathcal{P}$. We say that ϕ* expresses a liveness property *if every finite word $u \in \Sigma_{\mathcal{P}}^*$ has an extension to an ω-word that satisfies ϕ.*

Using the monitoring function, liveness of ϕ can thus be expressed as follows: ϕ expresses a liveness property iff $m_\phi(u) \neq \bot$ for all $u \in \Sigma_{\mathcal{P}}^*$. Consequently, given a monitor for ϕ, liveness of ϕ can again be tested by checking reachability in the monitor, which yields an ExpSpace upper bound. Again, it is open whether this upper bound is tight. The only known lower bound is again PSpace, which can again be obtained using a reduction from satisfiability.[13]

3 The Temporal DL $\mathcal{ALC}$-LTL

The temporal Description Logic $\mathcal{ALC}$-LTL introduced in [BGL12] combines LTL with the basic DL $\mathcal{ALC}$. More precisely, $\mathcal{ALC}$-LTL-formulae differ from LTL-formulae in that $\mathcal{ALC}$-axioms replace propositional variables. Before defining $\mathcal{ALC}$-LTL more formally, we recall the relevant definitions for the Description Logic $\mathcal{ALC}$.

3.1 The DL $\mathcal{ALC}$

Definition 3.1. *Let $\mathsf{N_C}$, $\mathsf{N_R}$, and $\mathsf{N_I}$ 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 $r \in \mathsf{N_R}$, then $\neg C$, $C \sqcup D$, $C \sqcap D$, $\exists r.C$, and $\forall r.C$ are $\mathcal{ALC}$-concept descriptions.*

[12]Note that the proof for a PSpace upper bound given in [Bau10] actually does not go through.

[13]Note that the proof for a PSpace upper bound sketched in [UNW01] actually does not go through.

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* $C(a)$ *or* $r(a, b)$ *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* $\mathcal{B}_1, \mathcal{B}_2$ *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 3.2. *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* $a^\mathcal{I} \in \Delta^\mathcal{I}$ *such that the* unique name assumption (UNA) *holds, i.e.,* $a^\mathcal{I} = b^\mathcal{I}$ *implies* $a = b$*. This function is extended to* $\mathcal{ALC}$*-concept descriptions as follows:*

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

The interpretation $\mathcal{I}$ *is a* model *of the* $\mathcal{ALC}$*-axioms* $C \sqsubseteq D$*,* $C(a)$*, and* $r(a, b)$ *if 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* $\mathcal{B}_1 \wedge \mathcal{B}_2$ *if it is a model of* $\mathcal{B}_1$ *and* $\mathcal{B}_2$*;*
- $\mathcal{I}$ *is a model of* $\mathcal{B}_1 \vee \mathcal{B}_2$ *if it is a model of* $\mathcal{B}_1$ *or* $\mathcal{B}_2$*;*
- $\mathcal{I}$ *is a model of* $\neg\mathcal{B}_1$ *if it is not a model of* $\mathcal{B}_1$*.*

We write $\mathcal{I} \models \mathcal{B}$ *to indicate that* $\mathcal{I}$ *is a model of* $\mathcal{B}$*. We say that the Boolean* $\mathcal{ALC}$*-knowledge base* $\mathcal{B}$ *is* consistent *if it has a model. We say that* $\mathcal{B}$ implies *the* $\mathcal{ALC}$*-axiom* α *(written* $\mathcal{B} \models \alpha$*) if every model of* $\mathcal{B}$ *is a model of* α*.*

The consistency problem for Boolean $\mathcal{ALC}$-knowledge bases is known to be ExpTime-complete (see, e.g., Theorem 2.27 in [GKWZ03] or Lemma 6.4 in [BGL12]).

3.2 Syntax and Semantics of $\mathcal{ALC}$-LTL

As mentioned above, the difference between $\mathcal{ALC}$-LTL and propositional LTL is that $\mathcal{ALC}$-axioms replace propositional variables.

Definition 3.3. *$\mathcal{ALC}$-LTL-formulae (with past-operators) are defined inductively:*

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

As usual, we use $\phi \vee \psi$ as an abbreviation for $\neg(\neg\phi \wedge \neg\psi)$, $\phi \rightarrow \psi$ as an abbreviation for $\neg\phi \vee \psi$, true as an abbreviation for $A(a) \vee \neg A(a)$, false as an abbreviation for $\neg\mathsf{true}$, $\Diamond\phi$ 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"). We also use the corresponding abbreviations for the past operators, i.e., $\Diamond^{-}\phi$ as abbreviation for $\mathsf{true}\,\mathsf{S}\,\phi$ and $\Box^{-}\phi$ as abbreviation for $\neg\Diamond^{-}\neg\phi$.

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).

Definition 3.4. *An $\mathcal{ALC}$-LTL-structure is a sequence $\mathfrak{I} = (\mathcal{I}_i)_{i\geq 0}$ of $\mathcal{ALC}$-interpretations $\mathcal{I}_i = (\Delta, \cdot^{\mathcal{I}_i})$ (called* worlds*) such that $a^{\mathcal{I}_i} = a^{\mathcal{I}_j}$ for all $a \in \mathsf{N_I}$ and all $i, j \geq 0$. Given an $\mathcal{ALC}$-LTL-formula ϕ, an $\mathcal{ALC}$-LTL-structure $\mathfrak{I} = (\mathcal{I}_i)_{i\geq 0}$, and a time point $i \geq 0$, validity of ϕ in $\mathfrak{I}$ at time i (written $\mathfrak{I}, i \models \phi$) is defined inductively:*

$$\begin{array}{lll}
\mathfrak{I}, i \models \alpha & \textit{iff} & \mathcal{I}_i \models \alpha \ \textit{ for } \mathcal{ALC}\textit{-axioms } \alpha \\
\mathfrak{I}, i \models \phi \wedge \psi & \textit{iff} & \mathfrak{I}, i \models \phi \textit{ and } \mathfrak{I}, i \models \psi \\
\mathfrak{I}, i \models \neg\phi & \textit{iff} & \textit{not } \mathfrak{I}, i \models \phi \\
\mathfrak{I}, i \models \mathsf{X}\phi & \textit{iff} & \mathfrak{I}, i+1 \models \phi \\
\mathfrak{I}, i \models \mathsf{X}^{-}\phi & \textit{iff} & i > 0 \textit{ and } \mathfrak{I}, i-1 \models \phi \\
\mathfrak{I}, i \models \phi \,\mathsf{U}\, \psi & \textit{iff} & \textit{there is some } k \geq i \textit{ such that } \mathfrak{I}, k \models \psi \\
 & & \textit{and } \mathfrak{I}, j \models \phi \textit{ for all } j, i \leq j < k \\
\mathfrak{I}, i \models \phi \,\mathsf{S}\, \psi & \textit{iff} & \textit{there is some } k, 0 \leq k \leq i, \textit{ such that } \mathfrak{I}, k \models \psi \\
 & & \textit{and } \mathfrak{I}, j \models \phi \textit{ for all } j, k < j \leq i
\end{array}$$

The $\mathcal{ALC}$-LTL-structure $\mathfrak{I}$ is called a model *of the $\mathcal{ALC}$-LTL-formula ϕ if $\mathfrak{I}, 0 \models \phi$. The $\mathcal{ALC}$-LTL-formula ϕ is* satisfiable *if it has a model.*

As mentioned in the introduction, 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. Let $\mathsf{N_{RC}}$ denote the *rigid concept names* and $\mathsf{N_{RR}}$ the *rigid role names* with $\mathsf{N_{RC}} \subseteq \mathsf{N_C}$ and $\mathsf{N_{RR}} \subseteq \mathsf{N_R}$. The concept and role names in $\mathsf{N_C} \setminus \mathsf{N_{RC}}$ and $\mathsf{N_R} \setminus \mathsf{N_{RR}}$ are called *flexible*.

Definition 3.5. *We say that the $\mathcal{ALC}$-LTL-structure $\mathfrak{I} = (\mathcal{I}_i)_{i\geq 0}$* respects rigid names *if $A^{\mathcal{I}_i} = A^{\mathcal{I}_j}$ and $r^{\mathcal{I}_i} = r^{\mathcal{I}_j}$ holds for all $i, j \geq 0$, all $A \in \mathsf{N_{RC}}$, and all $r \in \mathsf{N_{RR}}$. The $\mathcal{ALC}$-LTL-structure $\mathfrak{I}$ is a* model w.r.t. rigid names *of the $\mathcal{ALC}$-LTL-formula ϕ if $\mathfrak{I}, 0 \models \phi$ and $\mathfrak{I}$ respects rigid names. The $\mathcal{ALC}$-LTL-formula ϕ is* satisfiable w.r.t. rigid names *if it has a model w.r.t. rigid names.*

For clarity, if rigidity of names is not required, then we sometimes talk about *satisfiability without rigid names.*

In [BGL12], 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. Additional intermediate cases (such as the case where only concepts, but not roles, are required to be rigid; or where all GCIs occurring in the formula are global, i.e., required to hold at every point in time) are also investigated in [BGL12], but they are not considered in the present paper.

The decision procedures developed in [BGL12] to show the complexity upper bounds for reasoning in $\mathcal{ALC}$-LTL are not based on Büchi automata. In the next two subsections, we show, however, that the ideas underlying these decision procedures can also be used to obtain automata-based decision procedures. The automata constructed below will be the building blocks for our monitors.

3.3 Büchi Automata for the Case Without Rigid Names

In principle, given an $\mathcal{ALC}$-LTL-formula ϕ, we want to construct a Büchi automaton $\mathcal{N}_\phi$ 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 the $\mathcal{ALC}$-axioms occurring in ϕ that they satisfy.

For an $\mathcal{ALC}$-LTL-formula ϕ, we denote with $\mathsf{Ax}(\phi)$ the set of all $\mathcal{ALC}$-axioms occurring in ϕ. Clearly, the cardinality of $\mathsf{Ax}(\phi)$ is bounded by the size of ϕ. For example, the formula

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

contains the $\mathcal{ALC}$-axioms $A(a)$, $A \sqsubseteq B$, and $(\neg B)(a)$, and thus $\mathsf{Ax}(\phi_{\text{ex}}) = \{A(a), A \sqsubseteq B, (\neg B)(a)\}$. For a given $\mathcal{ALC}$-interpretation $\mathcal{I}$, we denote with

$\tau_\phi(\mathcal{I})$ the set of all $\mathcal{ALC}$-axioms in $\mathsf{Ax}(\phi)$ that $\mathcal{I}$ is a model of, i.e.,

$$\tau_\phi(\mathcal{I}) := \{\alpha \in \mathsf{Ax}(\phi) \mid \mathcal{I} \models \alpha\}.$$

Note that $\mathcal{I}$ is a model of the Boolean knowledge base

$$\bigwedge_{\alpha \in \tau_\phi(\mathcal{I})} \alpha \wedge \bigwedge_{\alpha \in \mathsf{Ax}(\phi) \setminus \tau_\phi(\mathcal{I})} \neg\alpha.$$

This motivates the following definition.

Definition 3.6. *Let ϕ be an $\mathcal{ALC}$-LTL-formula. The set of $\mathcal{ALC}$-axioms T is an* $\mathcal{ALC}$-type for ϕ *if the following two properties are satisfied:*

1. *$T \subseteq \mathsf{Ax}(\phi)$,*
2. *the Boolean $\mathcal{ALC}$-knowledge base $\mathcal{B}_T := \bigwedge_{\alpha \in T} \alpha \wedge \bigwedge_{\alpha \in \mathsf{Ax}(\phi) \setminus T} \neg\alpha$ is consistent.*

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

The following lemma is an easy consequence of this definition.

Lemma 3.7. *Let ϕ be an $\mathcal{ALC}$-LTL-formula.*

1. *If $\mathcal{I}$ is an $\mathcal{ALC}$-interpretation, then $\tau_\phi(\mathcal{I})$ is an $\mathcal{ALC}$-type for ϕ.*
2. *If T is an $\mathcal{ALC}$-type for ϕ, then there is an $\mathcal{ALC}$-interpretation $\mathcal{I}$ such that $T = \tau_\phi(\mathcal{I})$ and the domain $\Delta^\mathcal{I}$ of $\mathcal{I}$ is countably infinite.*

We call $\tau_\phi(\mathcal{I})$ the *$\mathcal{ALC}$-type of $\mathcal{I}$*. This notion can also be extended to $\mathcal{ALC}$-LTL-structures. For a given $\mathcal{ALC}$-LTL-structure $\mathfrak{I} = (\mathcal{I}_i)_{i \geq 0}$, we define

$$\tau_\phi(\mathfrak{I}) := \tau_\phi(\mathcal{I}_0)\tau_\phi(\mathcal{I}_1)\tau_\phi(\mathcal{I}_2)\cdots,$$

and call $\tau_\phi(\mathfrak{I})$ the *$\mathcal{ALC}$-type of $\mathfrak{I}$*. Note that the $\mathcal{ALC}$-type of $\mathfrak{I}$ is an ω-word over the alphabet $\mathfrak{T}^\phi$.

Whether a given $\mathcal{ALC}$-LTL-structure is a model of ϕ or not depends only on its $\mathcal{ALC}$-type. This is formally stated in the follwing lemma, which can easily be proved by induction on the structure of ϕ.

Lemma 3.8. *Let ϕ be an $\mathcal{ALC}$-LTL-formula and $\mathfrak{I}, \mathfrak{J}$ $\mathcal{ALC}$-LTL-structures such that $\tau_\phi(\mathfrak{I}) = \tau_\phi(\mathfrak{J})$. Then $\mathfrak{I}$ is a model of ϕ iff $\mathfrak{J}$ is a model of ϕ.*

This lemma justifies considering Büchi automata that receive $\mathcal{ALC}$-types of $\mathcal{ALC}$-LTL-structures as inputs rather than the $\mathcal{ALC}$-LTL-structures themselves.

Definition 3.9. *Let ϕ be an $\mathcal{ALC}$-LTL-formula and $\mathcal{N}$ a Büchi automaton using the alphabet $\mathfrak{T}^{\phi}$. We define*

$$L_\omega(\phi) := \{\tau_\phi(\mathfrak{I}) \mid \mathfrak{I} \text{ is a model of } \phi\},$$

and say that $\mathcal{N}$ is a Büchi automaton for ϕ *if $L_\omega(\mathcal{N}) = L_\omega(\phi)$.*

Instead of constructing such automata directly for $\mathcal{ALC}$-LTL-formulae, we build their propositional abstractions and then reuse the known construction for the propositional case.[14] The propositional abstraction of a given $\mathcal{ALC}$-LTL-formula is constructed by replacing each $\mathcal{ALC}$-axiom occurring in ϕ with a propositional variable such that there is a 1–1 relationship between the $\mathcal{ALC}$-axioms $\alpha_1, \dots, \alpha_m$ occurring in ϕ and the propositional variables $p_1, \dots, p_m$ occurring in its abstraction.

Definition 3.10. *Let ϕ be an $\mathcal{ALC}$-LTL-formula and $\mathcal{P}_\phi$ a finite set of propositional variables such that there is a bijection $\pi : \mathsf{Ax}(\phi) \to \mathcal{P}_\phi$.*

1. *The LTL-formula ϕ^π is obtained from ϕ by replacing every occurrence of an $\mathcal{ALC}$-axiom α in ϕ by its π-image $\pi(\alpha)$. We call ϕ^π the* propositional abstraction *of ϕ w.r.t. π.*

2. *Given an $\mathcal{ALC}$-LTL-structure $\mathfrak{I} = (\mathcal{I}_i)_{i\geq 0}$, its* propositional abstraction *w.r.t. π is the LTL-structure $\mathfrak{I}^\pi = (w_i)_{i\geq 0}$ with $w_i = \{\pi(\alpha) \mid \alpha \in \tau_\phi(\mathcal{I}_i)\}$ for all $i \geq 0$.*

As an example, consider the formula $\phi_{\text{ex}} = \mathsf{X}(A(a)) \wedge ((A \sqsubseteq B)\,\mathsf{U}\,((\neg B)(a)))$, and let $\pi : \mathsf{Ax}(\psi_{\text{ex}}) \to \{p_1, p_2, p_3\}$ be the bijection that maps $A(a)$ to p_1, $A \sqsubseteq B$ to p_2, and $(\neg B)(a)$ to p_3. Then the formula $\psi_{\text{ex}} = \mathsf{X}p_1 \wedge (p_2\,\mathsf{U}\,p_3)$ of Example 2.5 is the propositional abstraction of ϕ_{ex} w.r.t. π.

In the following, we assume that ϕ is an $\mathcal{ALC}$-LTL-formula and $\pi : \mathsf{Ax}(\phi) \to \mathcal{P}_\phi$ a bijection. The next lemma states that an $\mathcal{ALC}$-LTL-structure is a model of ϕ iff its abstraction is a model of the abstraction of ϕ. It can easily be proved by induction on the structure of $\mathcal{ALC}$-LTL-formulae.

Lemma 3.11. *Let $\mathfrak{I}$ be an $\mathcal{ALC}$-LTL-structure. Then, $\mathfrak{I}$ is a model of ϕ iff $\mathfrak{I}^\pi$ is a model of ϕ^π.*

The only-if direction of this lemma yields that satisfiability of ϕ implies satisfiability of ϕ^π. However, the if direction does not yield the converse of this implication.

[14]One could also define the Büchi automaton directly as done in [BBL09], but the approach developed below is more modular and also easier to implement. In fact, the approach does not depend on a specific algorithm for generating Büchi automata from propositional LTL-formulae. Thus, any existing efficient tool for transforming an LTL-formula into a Büchi automaton can be used.

In fact, ϕ^π may turn out to be satisfiable even though the original $\mathcal{ALC}$-LTL-formula ϕ is not. The reason is that there may exist LTL-structures that are not propositional abstractions of $\mathcal{ALC}$-LTL-structures. In our example, the LTL-structure $\mathfrak{W} = (w_i)_{i\geq 0}$ with $w_i = \{p_1, p_2, p_3\}$ for all $i \geq 0$ is a model of $\psi_{\text{ex}} = \phi^\pi_{\text{ex}}$, but there is no $\mathcal{ALC}$-LTL-structures $\mathfrak{J}$ such that $\mathfrak{J}^\pi = \mathfrak{W}$. In fact, every world of this structure would need to satisfy the three axioms in $\mathsf{Ax}(\psi_{\text{ex}})$ simultaneously, which is clearly not possible.

Thus, if we assume that $\mathcal{N}_{\phi^\pi}$ is a Büchi automaton for the propositional abstraction of ϕ, this automaton may accept ω-words that do not correspond to abstractions of $\mathcal{ALC}$-LTL-structures. This problem can be addressed by requiring that the letters occurring in such a word correspond to abstractions of $\mathcal{ALC}$-types for ϕ.

Lemma 3.12. *For every ω-word $w = w_0w_1w_2\ldots \in \Sigma^\omega_{\mathcal{P}_\phi}$, the following two statements are equivalent:*

1. *There is a model $\mathfrak{J}$ of ϕ with $\mathfrak{J}^\pi = (w_i)_{i\geq 0}$.*
2. *$w \in L_\omega(\mathcal{N}_{\phi^\pi})$ and for all letters w_i occurring in w, we have that*

$$T_{w_i} := \{\pi^{-1}(p) \mid p \in w_i\}$$

is an $\mathcal{ALC}$-type for ϕ, i.e., $T_{w_i} \in \mathfrak{T}^\phi$.

Proof. ($1 \implies 2$) Assume that there exists an $\mathcal{ALC}$-LTL structure $\mathfrak{J} = (\mathcal{I}_i)_{i\geq 0}$ with $\mathfrak{J}^\pi = (w_i)_{i\geq 0}$ and $\mathfrak{J}, 0 \models \phi$. By Lemma 3.11, we have $\mathfrak{J}^\pi, 0 \models \phi^\pi$, and thus $w \in L_\omega(\mathcal{N}_{\phi^\pi})$. By the definition of the propositional abstraction of $\mathfrak{J}$, we have $T_{w_i} = \tau_\phi(\mathcal{I}_i)$ for $i \geq 1$, and by Lemma 3.7 the sets $\tau_\phi(\mathcal{I}_i)$ are $\mathcal{ALC}$-types.

($2 \implies 1$) Assume that $w = w_0w_1w_2\ldots \in L_\omega(\mathcal{N}_{\phi^\pi})$ and, for all $i \geq 0$, we have that T_{w_i} is an $\mathcal{ALC}$-type for ϕ. The first fact implies that $(w_i)_{i\geq 0}$ is a model of ϕ^π. By Lemma 3.7, the second fact yields that there are $\mathcal{ALC}$-interpretations $\mathcal{I}_i$ with countably infinite domains such that $T_{w_i} = \tau_\phi(\mathcal{I}_i)$. Since the domains of the interpretations $\mathcal{I}_i$ have the same cardinality, we can assume without loss of generality that they have the same domain. In addition, since these interpretations obey the UNA, we can also assume that they interpret the individual names in the same way. Consequently, if we define $\mathfrak{J} := (\mathcal{I}_i)_{i\geq 0}$, then $\mathfrak{J}$ is an $\mathcal{ALC}$-LTL-structure and $\mathfrak{J}^\pi = (w_i)_{i\geq 0}$. Since the propositional abstraction of $\mathfrak{J}$ is a model of ϕ^π, Lemma 3.11 yields that $\mathfrak{J}$ is a model of ϕ. □

This lemma shows how a Büchi automaton for the $\mathcal{ALC}$-LTL-formula ϕ can be constructed from a Büchi automaton for the propositional abstraction of ϕ. Basically, all transitions labeled with letters not corresponding to $\mathcal{ALC}$-types must be removed.

Theorem 3.13. *Let ϕ be an $\mathcal{ALC}$-LTL-formula and $\pi : \mathsf{Ax}(\phi) \to \mathcal{P}_\phi$ a bijection. If $\mathcal{N}_{\phi^\pi} = (Q, \Sigma_{\mathcal{P}_\phi}, \Delta, Q_0, F)$ is a Büchi automaton for the propositional abstraction ϕ^π of ϕ, then the following is a Büchi automaton for ϕ: $\mathcal{N}_\phi := (Q, \mathfrak{T}^\phi, \Delta', Q_0, F)$ where*

$$\Delta' := \{(q, T_\sigma, q') \mid (q, \sigma, q') \in \Delta \text{ and } T_\sigma \in \mathfrak{T}^\phi\}.$$

Proof. We must show that $L_\omega(\mathcal{N}_\phi) = L_\omega(\phi)$.

First, assume that $T = T_0T_1T_2 \ldots \in L_\omega(\mathcal{N}_\phi)$. Since the alphabet of $\mathcal{N}_\phi$ is $\mathfrak{T}^\phi$, we know that $T_i \in \mathfrak{T}^\phi$. In addition, the definition of $\mathcal{N}_\phi$ implies that there is a word $w = w_0w_1w_2 \ldots \in L_\omega(\mathcal{N}_{\phi^\pi})$ such that $T_i = T_{w_i}$. Thus, Lemma 3.12 yields the existence of a model $\mathfrak{I}$ of ϕ with $\mathfrak{I}^\pi = (w_i)_{i \geq 0}$. Consequently, $\tau_\phi(\mathfrak{I}) = T_0T_1T_2 \ldots \in L_\omega(\phi)$.

Conversely, assume that $T = T_0T_1T_2 \ldots \in L_\omega(\phi)$. Then there is a model $\mathfrak{I}$ of ϕ with $\tau_\phi(\mathfrak{I}) = T_0T_1T_2 \ldots$. By Lemma 3.7, the letters T_i are $\mathcal{ALC}$-types for ϕ, i.e., $T_i \in \mathfrak{T}^\phi$. Let $\mathfrak{I}^\pi = (w_i)_{i \geq 0}$ be the propositional abstraction of $\mathfrak{I}$. By Lemma 3.11, $\mathfrak{I}^\pi$ is a model of ϕ^π, and thus the ω-word $w = w_0w_1w_2 \ldots$ is accepted by $\mathcal{N}_{\phi^\pi}$. Since $T_i = T_{w_i}$ and the letters T_i belong to $\mathfrak{T}^\phi$, this implies that $T = T_0T_1T_2 \ldots$ is accepted by $\mathcal{N}_\phi$. □

As an immediate consequence of this theorem, the satisfiability problem in $\mathcal{ALC}$-LTL (without rigid names) can be reduced to the emptiness problem for Büchi automata.

Corollary 3.14. *The $\mathcal{ALC}$-LTL-formula ϕ is satisfiable iff $L_\omega(\mathcal{N}_\phi) \neq \emptyset$.*

It remains to analyse the complexity of the decision procedure for satisfiability obtained by this reduction.

The size of the automaton $\mathcal{N}_\phi$ is obviously exponential in the size of ϕ as $\mathcal{N}_{\phi^\pi}$ is exponential in the size of ϕ^π and the size of ϕ^π is linearly bounded by the size of ϕ. In addition, the automaton $\mathcal{N}_\phi$ can be computed in exponential time. As mentioned above, a Büchi automaton for a propositional LTL-formula can be computed in time exponential in the size of the formula. Thus, we can compute $\mathcal{N}_{\phi^\pi}$ in exponential time. To obtain $\mathcal{N}_\phi$ from $\mathcal{N}_{\phi^\pi}$, we basically have to remove all transitions labeled with a letter σ such that $T_\sigma \notin \mathfrak{T}^\phi$. For the remaining transitions, we then simply replace σ with T_σ. In order to test whether T_σ belongs to $\mathfrak{T}^\phi$, we need to test the Boolean $\mathcal{ALC}$-knowledge base B_{T_σ} for consistency. As mentioned before, this can be done in exponential time in the size of this knowledge base. Since there are exponentially many letters σ, but the size of each knowledge base B_{T_σ} is linearly bounded by the size of ϕ, we need to perform exponentially many exponential-time consistency tests, which yields an overall exponential-time complexity for the computation of $\mathcal{N}_\phi$.

Since the emptiness problem for Büchi automata can be solved in polynomial time [VW94], this yields an alternative proof for the fact (originally shown in [BGL12] for the case without past operators) that satisfiability of $\mathcal{ALC}$-LTL-formulae (without rigid names) can be decided in exponential time.

3.4 The Case With Rigid Names

In this case, we are only interested in the $\mathcal{ALC}$-types of models that respect rigid names. Thus, we need to modify the notion of a Büchi automaton for a formula accordingly.

Definition 3.15. *Let ϕ be an $\mathcal{ALC}$-LTL-formula and $\mathcal{N}$ a Büchi automaton using the alphabet $\mathfrak{T}^{\phi}$. We define*

$$L^r_\omega(\phi) := \{\tau_\phi(\mathfrak{I}) \mid \mathfrak{I} \textit{ is a model of } \phi \textit{ respecting rigid names}\},$$

and say that $\mathcal{N}$ is a Büchi automaton for ϕ respecting rigid names *if $L_\omega(\mathcal{N}) = L^r_\omega(\phi)$.*

For the case without rigid names, Lemma 3.12 characterises under which conditions an ω-word over the alphabet $\Sigma_{\mathcal{P}_\phi}$ is the propositional abstraction of a model of the $\mathcal{ALC}$-LTL-formula ϕ. If rigid concepts and roles occur in ϕ, then this characterization is clearly not sufficient since rigidity of their interpretation is not enforced.

As an example, consider the formula $\phi_{\text{ex}} = \mathsf{X}(A(a)) \wedge ((A \sqsubseteq B)\, \mathsf{U}\, ((\neg B)(a)))$ from our previous example and its propositional abstraction $\phi^\pi = \psi_{\text{ex}} = \mathsf{X}p_1 \wedge (p_2\, \mathsf{U}\, p_3)$. The LTL-structure $\mathfrak{W} = (w_i)_{i\geq 0}$ with $w_0 = \{p_2\}$ and $w_i = \{p_1, p_3\}$ for all $i \geq 1$ satisfies 2. of Lemma 3.12, where $\mathcal{N}_{\phi^\pi}$ is the automaton depicted in Figure 1. However, an $\mathcal{ALC}$-LTL-structure that has $\mathfrak{W}$ as its propositional abstraction does not interpret A as a rigid concept: a does not belong to A at time point 0, but belongs to it at all later points in time.

In order to obtain a characterization analogous to the one in Lemma 3.12 also for the case of rigid names, we must add an additional condition.

Definition 3.16. *The set $\mathfrak{T} = \{T_1, \ldots, T_k\}$ of $\mathcal{ALC}$-types for ϕ is called* r-consistent *if there are $\mathcal{ALC}$-interpretations $\mathcal{I}_1, \ldots, \mathcal{I}_k$ that share the same domain, coincide on the individual names and the rigid concept and role names, and satisfy $\tau_\phi(\mathcal{I}_i) = T_i$ for all i, $1 \leq i \leq k$. The set of all r-consistent sets of $\mathcal{ALC}$-types for ϕ is denoted by $\mathcal{C}^\phi_r$.*

Note that any subset of an r-consistent set of $\mathcal{ALC}$-types for ϕ is again r-consistent. In particular, the empty set is always r-consistent. The r-consistency of a set of $\mathcal{ALC}$-types can be decided using the renaming technique for flexible symbols introduced in [BGL12]. Given a set $\mathfrak{T} = \{T_1, \ldots, T_k\}$ of $\mathcal{ALC}$-types for ϕ, we introduce renamed variants $A^{(i)}$ and $r^{(i)}$ $(1 \leq i \leq k)$ for every *flexible* concept name A and every *flexible* role name r. For an $\mathcal{ALC}$-axiom α of ϕ, 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 lemma is an easy consequence of the proof of Lemma 4.3 in [BGL12].

Lemma 3.17. *Let $\mathfrak{T} = \{T_1, \ldots, T_k\}$ be a set of $\mathcal{ALC}$-types for ϕ. Then $\mathfrak{T}$ is r-consistent iff the Boolean $\mathcal{ALC}$-knowledge base $\mathcal{B}_{\mathfrak{T}}$ is consistent, where*

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

The next lemma characterises the additional condition the automaton that respects rigid names has to take care of.

Lemma 3.18. *For every ω-word $w = w_0w_1w_2\ldots \in \Sigma_{\mathcal{P}_\phi}^{\omega}$, the following two statements are equivalent:*

1. *There is a model $\mathfrak{I}$ of ϕ* respecting rigid names *with $\mathfrak{I}^\pi = (w_i)_{i\geq 0}$.*
2. *$w \in L_\omega(\mathcal{N}_{\phi^\pi})$ and the set*

$$\mathfrak{T}_w := \{T_{w_i} \mid i \geq 0\}$$

is an r-consistent set of $\mathcal{ALC}$-types for ϕ.

Proof. ($1 \implies 2$) Assume that there exists an $\mathcal{ALC}$-LTL structure $\mathfrak{I} = (\mathcal{I}_i)_{i\geq 0}$ respecting rigid names such that $\mathfrak{I}^\pi = (w_i)_{i\geq 0}$ and $\mathfrak{I}, 0 \models \phi$. By Lemma 3.12, $w \in L_\omega(\mathcal{N}_{\phi^\pi})$ and for all letters w_i occurring in w, we have that T_{w_i} is an $\mathcal{ALC}$-type for ϕ. Consequently, $\mathfrak{T}_w = \{T_{w_i} \mid i \geq 0\}$ is a set of $\mathcal{ALC}$-types. Since there are only finitely many $\mathcal{ALC}$-types for ϕ_i, $\mathfrak{T}_w$ is a finite set. Let $\mathfrak{T}_w = \{T_1, \ldots, T_k\}$ and let i_j for $1 \leq j \leq k$ be such that $T_i = T_{w_{i_j}}$. Then $\mathcal{I}_{i_1}, \ldots, \mathcal{I}_{i_k}$ are $\mathcal{ALC}$-interpretations such that $\tau_\phi(\mathcal{I}_{i_j}) = T_j$ for all $j, 1 \leq j \leq k$. Since $\mathfrak{I} = (\mathcal{I}_i)_{i\geq 0}$ is an $\mathcal{ALC}$-LTL structure respecting rigid names, the interpretations $\mathcal{I}_{i_j}$ ($j = 1, \ldots, k$) have the same domain and coincide on the individual names and the rigid concept and role names.

($2 \implies 1$) Assume that $w = w_0w_1w_2\ldots \in L_\omega(\mathcal{N}_{\phi^\pi})$ and that $\mathfrak{T}_w = \{T_{w_i} \mid i \geq 0\}$ is an r-consistent set of $\mathcal{ALC}$-types for ϕ. The first fact implies that $(w_i)_{i\geq 0}$ is a model of ϕ^π. Let $\mathfrak{T}_w = \{T_1, \ldots, T_k\}$, where $T_1, \ldots, T_k$ are the finitely many, pairwise distinct $\mathcal{ALC}$-types for ϕ occurring in $\mathfrak{T}_w$. The r-consistency of this set yields $\mathcal{ALC}$-interpretations $\mathcal{I}_1, \ldots, \mathcal{I}_k$ that share the same domain, coincide on the individual names and the rigid concept and role names, and satisfy $\tau_\phi(\mathcal{I}_j) = T_j$ for all $j, 1 \leq j \leq k$. Since $\{T_{w_i} \mid i \geq 0\} = \mathfrak{T}_w = \{T_1, \ldots, T_k\}$, there is a mapping $\mu : \{0, 1, 2, \ldots\} \to \{1, \ldots, k\}$ such that $T_{w_i} = T_{\mu(i)}$ for all $i \geq 0$. We define the sequence of $\mathcal{ALC}$-interpretations $\mathfrak{J} = (\mathcal{J}_i)_{i\geq 0}$ by setting $\mathcal{J}_i := \mathcal{I}_{\mu(i)}$. Since the $\mathcal{ALC}$-interpretations $\mathcal{I}_1, \dots, \mathcal{I}_k$ share the same domain and coincide on the individual names, this sequence is indeed an $\mathcal{ALC}$-LTL structure. This structure respects rigid names since the $\mathcal{ALC}$-interpretations $\mathcal{I}_1, \dots, \mathcal{I}_k$ coincide on the rigid concept and role names. Since $\mathfrak{J}^\pi = (w_i)_{i \geq 0}$, the propositional abstraction of $\mathfrak{J}$ is a model of ϕ^π, and thus Lemma 3.11 yields that $\mathfrak{J}$ is a model of ϕ. $\square$

A Büchi automaton that respects rigid names thus needs to check the r-consistency of the set of $\mathcal{ALC}$-types it has seen within a run. This is realized by using tuples (q_1, q_2) as states, where q_1 is a state of the automaton $\mathcal{N}_\phi$ introduced in Theorem 3.13, and q_2 is an r-consistent set of $\mathcal{ALC}$-types for ϕ.

Theorem 3.19. *Let ϕ be an $\mathcal{ALC}$-LTL-formula and $\pi : \mathsf{Ax}(\phi) \to \mathcal{P}_\phi$ a bijection. If $\mathcal{N}_{\phi^\pi} = (Q, \Sigma_{\mathcal{P}_\phi}, \Delta, Q_0, F)$ is a Büchi automaton for the propositional abstraction ϕ^π of ϕ, then the following is a Büchi automaton for ϕ with respect to rigid names: $\mathcal{N}_\phi^r := (Q \times \mathcal{C}_r^\phi, \mathfrak{T}^\phi, \Delta', Q_0 \times \{\emptyset\}, F \times \mathcal{C}_r^\phi)$ where*

$$\Delta' := \{((q_1, q_2), T_\sigma, (q_1', q_2')) \mid (q_1, \sigma, q_1') \in \Delta, T_\sigma \in \mathfrak{T}^\phi, \text{ and } q_2' = q_2 \cup \{T_\sigma\} \in \mathcal{C}_r^\phi\}.$$

Proof. We must show that $L_\omega(\mathcal{N}_\phi^r) = L_\omega^r(\phi)$.

First, assume that $T = T_0T_1T_2 \ldots \in L_\omega^r(\mathcal{N}_\phi)$, and let $(q_1^{(0)}, q_2^{(0)})(q_1^{(1)}, q_2^{(1)})\,(q_1^{(2)}, q_2^{(2)}) \ldots$ be an accepting run of $\mathcal{N}_\phi^r$ on T. It is easy to see that the projection $q_1^{(0)}q_1^{(1)}q_1^{(2)} \ldots$ of this run to the first component is an accepting run of $\mathcal{N}_\phi$ on T. We have seen in the proof of Theorem 3.13 that this implies the existence of a word $w = w_0w_1w_2 \ldots \in L_\omega(\mathcal{N}_{\phi^\pi})$ such that $T_i = T_{w_i}$. In addition, the letters T_i are $\mathcal{ALC}$-types. It remains to show that the set $\mathfrak{T}_w = \{T_{w_i} \mid i \geq 0\}$ is an r-consistent set of $\mathcal{ALC}$-types. In fact, once this is shown, Lemma 3.18 yields a model $\mathfrak{J}$ of ϕ respecting rigid names with $\mathfrak{J}^\pi = (w_i)_{i \geq 0}$. Consequently, $\tau_\phi(\mathfrak{J}) = T_0T_1T_2 \ldots \in L_\omega^r(\phi)$. To see that $\mathfrak{T}_w$ is r-consistent, we note that the second components $q_2^{(j)}$ of the states in the run are r-consistent sets of $\mathcal{ALC}$-types satisfying $q_2^{(j)} = \{T_{w_0}, \ldots, T_{w_{j-1}}\}$. Since there are only finitely many $\mathcal{ALC}$-types, there is an index $k \geq 1$ such that $q_2^{(k)} = \{T_{w_i} \mid i \geq 0\}$, and thus this set is r-consistent.

Conversely, assume that $T = T_0T_1T_2 \ldots \in L_\omega^r(\phi)$. Then there is a model $\mathfrak{J}$ of ϕ respecting rigid names with $\tau_\phi(\mathfrak{J}) = T_0T_1T_2 \ldots$. The letters T_i are $\mathcal{ALC}$-types by Lemma 3.7, and since $\mathfrak{J}$ respects rigid names, the set $\{T_i \mid i \geq 0\}$ is r-consistent (see the argument used in the proof of $(1 \Longrightarrow 2)$ of Lemma 3.18). Since $L_\omega^r(\phi) \subseteq L_\omega(\phi)$, we have $T = T_0T_1T_2 \ldots \in L_\omega(\phi)$, and thus Theorem 3.13 implies the existence of an accepting run $q_1^{(0)}q_1^{(1)}q_1^{(2)} \ldots$ of $\mathcal{N}_\phi$ on T. If we define $q_2^{(j)} := \{T_0, \ldots, T_{j-1}\}$ for all $j \geq 0$, then these sets are r-consistent since they are subsets of the r-consistent set $\{T_i \mid i \geq 0\}$. Consequently, $(q_1^{(0)}, q_2^{(0)})(q_1^{(1)}, q_2^{(1)})\,(q_1^{(2)}, q_2^{(2)}) \ldots$ is an accepting run of $\mathcal{N}_\phi^r$ on T. $\square$

As an immediate consequence of this theorem, the satisfiability problem in $\mathcal{ALC}$-LTL with rigid names can also be reduced to the emptiness problem for Büchi automata.

Corollary 3.20. *The $\mathcal{ALC}$-LTL-formula ϕ is satisfiable w.r.t. rigid names iff $L_\omega(\mathcal{N}_\phi^r) \neq \emptyset$.*

The complexity of the decision procedure for satisfiability obtained by this reduction is, however, higher than the complexity of the decision procedure for the case without rigid names.

The size of the automaton $\mathcal{N}_\phi^r$ is double exponential in the size of ϕ. This is due to the fact that the set $\mathcal{C}_\mathrm{r}^\phi$ of all r-consistent sets of $\mathcal{ALC}$-types for ϕ may contain double exponentially many elements since these sets are subsets of the exponentially large set of all $\mathcal{ALC}$-types for ϕ. Each element of $\mathcal{C}_\mathrm{r}^\phi$ may be of exponential size.

Next, we show that the automaton $\mathcal{N}_\phi^r$ can be computed in double exponential time. In addition to computing $\mathcal{N}_\phi$, i.e., the automaton constituting the first component of $\mathcal{N}_\phi^r$, we must also compute the set $\mathcal{C}_\mathrm{r}^\phi$. For this, we consider 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 Lemma 3.17, 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 $\mathcal{C}_\mathrm{r}^\phi$ requires double exponentially many tests, each requiring doubly exponential time. This shows that $\mathcal{C}_\mathrm{r}^\phi$, and thus also the automaton $\mathcal{N}_\phi^r$, can be computed in double exponential time.

Since the emptiness problem for Büchi automata can be solved in polynomial time [VW94], this yields an alternative proof for the fact (originally shown in [BGL12] for the case without past operators) that satisfiability w.r.t. rigid names in $\mathcal{ALC}$-LTL can be decided in double exponential time.

4 Monitoring $\mathcal{ALC}$-LTL-formulae

In this section, we extend existing definitions and results for runtime verification from propositional LTL to $\mathcal{ALC}$-LTL. We restrict the attention to the case with rigid names since the complexity of the monitor construction for this more general case is actually the same (double exponential) as for the case without rigid names. Thus, it does not make sense to treat the restricted case separately. In addition to considering a more expressive logic, our notion of monitoring extends the one for propositional logic in two directions.

On the one hand, we do not assume that the monitor has complete knowledge about the states of the system. In the propositional case, as introduced in Section 2, at each point in time the monitor knows which of the propositional variables are true at this point and which are not. In our setting, $\mathcal{ALC}$-axioms take the place of propositional variables, but we do not assume that we have complete knowledge about their truth value. For some of the relevant axioms, we may know that they are true, for others that they are false, but it also may be the case that we have no information regarding the truth status of a certain axiom.

On the other hand, we take background knowledge about the working of the system into account. This background knowledge could, for example, be a global TBox, i.e., a finite set of GCIs that are known to hold for every state of the system. In this case, the formula describing the background knowledge is of the form $\psi = \Box\mathcal{T}$, where $\mathcal{T}$ is a TBox. The presence of background knowledge enables the monitor to give more often definite answers (i.e., $\top$ or $\bot$) rather than the answer ?.

4.1 Basic Definitions

In the following, we extend the notion of a monitoring function and a monitor, as introduced in Section 2 for propositional LTL-formulae, to the case of $\mathcal{ALC}$-LTL-formulae ϕ. We assume that the background knowledge is described by an additional $\mathcal{ALC}$-LTL-formula ψ and that the monitor receives the information about the current state of the system in the form of a Boolean knowledge base $\mathcal{K}$ that provides (partial) information about the truth values of certain axioms from a fixed finite set of axioms. Without loss of generality, we can also assume that the axioms occurring in ψ and $\mathcal{K}$ also occur in ϕ. This assumption is made throughout this section without explicitly mentioning it.

To simplify the subsequent definitions, we introduce the following notation. A *literal of* ϕ is either an axiom $\alpha \in \mathsf{Ax}(\phi)$ (*positive* literal) or the negation $\neg\alpha$ of an axiom $\alpha \in \mathsf{Ax}(\phi)$ (*negative* literal).

Definition 4.1. *A finite conjunction $\mathcal{K} = L_1 \wedge \ldots \wedge L_m$ of literals of ϕ is a* partial $\mathcal{ALC}$-type for ϕ *if this Boolean $\mathcal{ALC}$-knowledge base is consistent. We denote the set of all partial $\mathcal{ALC}$-types for ϕ with $\mathfrak{P}^{\phi}$.*

Note that, up to equivalence, there are at most exponentially many partial $\mathcal{ALC}$-types for ϕ in the size of ϕ. Given an $\mathcal{ALC}$-type T for ϕ, the corresponding Boolean knowledge base

$$\mathcal{B}_T = \bigwedge_{\alpha \in T} \alpha \wedge \bigwedge_{\alpha \in \mathsf{Ax}(\phi)\setminus T} \neg\alpha$$

is a partial $\mathcal{ALC}$-type for ϕ. In such a knowledge base $\mathcal{B}_T$, every axiom of ϕ occurs either positively or negatively. For an arbitrary partial $\mathcal{ALC}$-type for ϕ, this need not be the case. Some axioms of ϕ may not occur at all in $\mathcal{K}$.

Definition 4.2. *Let ϕ, ψ be $\mathcal{ALC}$-LTL-formulae and $\mathfrak{K} = \mathcal{K}_0\mathcal{K}_1 \dots \mathcal{K}_t$ a* finite *sequence of partial $\mathcal{ALC}$-types for ϕ.*

1. *We say that the $\mathcal{ALC}$-LTL-structure $\mathfrak{J} = (\mathcal{J}_i)_{i\geq 0}$* extends $\mathfrak{K}$ w.r.t. ψ *if $\mathfrak{J}, 0 \models \psi$ and $\mathcal{J}_i \models \mathcal{K}_i$ for all i, $0 \leq i \leq t$.*
2. *We write $\mathfrak{K}, \psi \mathrel{|\!\approx_\exists} \phi$ if there is an $\mathcal{ALC}$-LTL-structure $\mathfrak{J}$ respecting rigid names that extends $\mathfrak{K}$ w.r.t. ψ and satisfies $\mathfrak{J}, 0 \models \phi$. If this is not the case, we write $\mathfrak{K}, \psi \mathrel{|\!\not\approx_\exists} \phi$.*
3. *We write $\mathfrak{K}, \psi \mathrel{|\!\approx_\forall} \phi$ if all $\mathcal{ALC}$-LTL-structures $\mathfrak{J}$ extending $\mathfrak{K}$ w.r.t. ψ and respecting rigid names are such that $\mathfrak{J}, 0 \models \phi$. If this is not the case, we write $\mathfrak{K}, \psi \mathrel{|\!\not\approx_\forall} \phi$.*

The notions introduced in 2. and 3. of this definition are dual to each other in the following sense:

$$\mathfrak{K}, \psi \mathrel{|\!\approx_\exists} \phi \text{ iff } \mathfrak{K}, \psi \mathrel{|\!\not\approx_\forall} \neg\phi \quad \text{and} \quad \mathfrak{K}, \psi \mathrel{|\!\approx_\forall} \phi \text{ iff } \mathfrak{K}, \psi \mathrel{|\!\not\approx_\exists} \neg\phi.$$

We assume that our system actually respects rigid names and satisfies the background knowledge ψ in the sense that any run of the system corresponds to an $\mathcal{ALC}$-LTL-structure respecting rigid names that is a model of ψ. Thus, if the monitor receives information about the partial types of a finite prefix of such a run, this finite sequence of partial types can actually be extended to an $\mathcal{ALC}$-LTL-structure satisfying ψ and respecting rigid names. In this case, the following lemma holds.

Lemma 4.3. *Let ϕ, ψ be $\mathcal{ALC}$-LTL-formulae and $\mathfrak{K}$ a finite sequence of partial $\mathcal{ALC}$-types for ϕ such that there is an $\mathcal{ALC}$-LTL-structure extending $\mathfrak{K}$ w.r.t. ψ and respecting rigid names. Then $\mathfrak{K}, \psi \mathrel{|\!\approx_\forall} \phi$ and $\mathfrak{K}, \psi \mathrel{|\!\approx_\forall} \neg\phi$ cannot both be true.*

Proof. Let $\mathfrak{J}$ be an $\mathcal{ALC}$-LTL-structure respecting rigid names and extending $\mathfrak{K}$ w.r.t. ψ. Then we have $\mathfrak{J}, 0 \models \phi$ or $\mathfrak{J}, 0 \models \neg\phi$. In the first case, $\mathfrak{K}, \psi \mathrel{|\!\not\approx_\forall} \neg\phi$ and in the second, $\mathfrak{K}, \psi \mathrel{|\!\not\approx_\forall} \phi$. □

The monitoring function receives as input a finite sequence of partial $\mathcal{ALC}$-types for ϕ, i.e., a finite word over the alphabet $\mathfrak{P}^\phi$.

Definition 4.4. *Let ϕ, ψ be $\mathcal{ALC}$-LTL-formulae. The* monitoring function for ϕ w.r.t. ψ *is the function $m_{\phi,\psi} : (\mathfrak{P}^\phi)^* \to \{\top, \bot, ?, \lightning\}$ with*

$$m_{\phi,\psi}(\mathfrak{K}) := \begin{cases} \top & \text{if } \mathfrak{K}, \psi \mathrel{|\!\approx_\forall} \phi \text{ and } \mathfrak{K}, \psi \mathrel{|\!\not\approx_\forall} \neg\phi \\ \bot & \text{if } \mathfrak{K}, \psi \mathrel{|\!\not\approx_\forall} \phi \text{ and } \mathfrak{K}, \psi \mathrel{|\!\approx_\forall} \neg\phi \\ ? & \text{if } \mathfrak{K}, \psi \mathrel{|\!\not\approx_\forall} \phi \text{ and } \mathfrak{K}, \psi \mathrel{|\!\not\approx_\forall} \neg\phi \\ \lightning & \text{if } \mathfrak{K}, \psi \mathrel{|\!\approx_\forall} \phi \text{ and } \mathfrak{K}, \psi \mathrel{|\!\approx_\forall} \neg\phi. \end{cases}$$

Compared to the definition in the propositional setting, we have added a fourth possible output for the monitoring function in order to have a well-defined value also for sequences $\mathfrak{K} \in (\mathfrak{P}^\phi)^*$ that have no extension respecting rigid names w.r.t. ψ. In fact, if there is no $\mathcal{ALC}$-LTL-structure respecting rigid names and extending $\mathfrak{K}$ w.r.t. ψ, then we have both $\mathfrak{K}, \psi \models\!\!\approx_\forall \phi$ and $\mathfrak{K}, \psi \models\!\!\approx_\forall \neg\phi$. In this case, the monitoring function yields the value $\lightning$. In practice, this value should not be encountered since we assume that the observed system actually respects rigid names and satisfies the background knowledge ψ. Thus, no finite sequence of partial types obtained by observing the system can yield this case.[15] The monitoring function returns the value $\top$ if there is at least one extension of $\mathfrak{K}$ w.r.t. ψ that respects rigid names (expressed by $\mathfrak{K}, \psi \not\models\!\!\approx_\forall \neg\phi$), and all such extensions satisfy ϕ (expressed by $\mathfrak{K}, \psi \models\!\!\approx_\forall \phi$). Similarly, it returns the value $\perp$ if there is at least one extension of $\mathfrak{K}$ w.r.t. ψ that respects rigid names, and all such extension satisfy $\neg\phi$. Finally, it returns he value ? if there is an extension of $\mathfrak{K}$ w.r.t. ψ that respects rigid names and satisfies ϕ, and there is another extension of $\mathfrak{K}$ w.r.t. ψ that respects rigid names and satisfies $\neg\phi$.

We are interested in constructing a monitor that realizes the monitoring function defined above. As in the propositional case, this monitor is a deterministic Moore automaton whose output function is equal to the monitoring function.

Definition 4.5. *Let ϕ, ψ be $\mathcal{ALC}$-LTL-formulae. The deterministic Moore automaton $\mathcal{M} = (S, \mathfrak{P}^\phi, \delta, s_0, \{\top, \perp, ?, \lightning\}, \lambda)$ is a* monitor for ϕ w.r.t. ψ *if $\widehat{\lambda}(\mathfrak{K}) = m_{\phi,\psi}(\mathfrak{K})$ holds for all $\mathfrak{K} \in (\mathfrak{P}^\phi)^*$.*

4.2 An auxiliary deterministic automaton

Before we construct the monitor, we define a *deterministic* finite automaton that accepts exactly those sequences of partial types $\mathfrak{K} \in (\mathfrak{P}^\phi)^*$ such that $\mathfrak{K}, \psi \models\!\!\approx_\forall \phi$. We know that requiring $\mathfrak{K}, \psi \models\!\!\approx_\forall \phi$ is the same as requiring $\mathfrak{K}, \psi \not\models\!\!\approx_\exists \neg\phi$. Thus, the automaton needs to accept all words $\mathfrak{K} \in (\mathfrak{P}^\phi)^*$ that have no extension w.r.t. ψ and respecting rigid names that satisfies $\neg\phi$. To construct this automaton, we take the Büchi automaton $\mathcal{N}^r_{\neg\phi\wedge\psi}$ for $\neg\phi \wedge \psi$ and make it deterministic by applying an appropriate modification of the powerset construction to the first components of the states of $\mathcal{N}^r_{\neg\phi\wedge\psi}$. The second component of a state of $\mathcal{N}^r_{\neg\phi\wedge\psi}$ collects the $\mathcal{ALC}$-types encountered on the path leading to this state, which enables the automaton to check whether this collection of $\mathcal{ALC}$-types is r-consistent. Instead, our deterministic automaton collects the partial $\mathcal{ALC}$-types encountered on a path, and checks whether this set is related in an appropriate way to an r-consistent set of $\mathcal{ALC}$-types.

Before we can define this relation, we need to introduce some notation. Given a

[15]If it does, then the modelling of the properties of the system using ψ and the rigididy of symbols was incorrect, or the sensors that generated the sequence $\mathfrak{K}$ were faulty.

partial type $\mathcal{K} = L_1 \wedge \ldots \wedge L_m$, we define $\mathsf{Pos}(\mathcal{K}) = \{L_i \mid 1 \leq i \leq m, L_i \text{ is positive}\}$ and $\mathsf{Neg}(\mathcal{K}) = \{\alpha_i \mid 1 \leq i \leq m, L_i = \neg\alpha_i \text{ is negative}\}$. Given an $\mathcal{ALC}$-type T for ϕ and a partial $\mathcal{ALC}$-type $\mathcal{K}$ for ϕ, we define

$$\mathcal{K} <^\phi T \text{ iff } \mathsf{Pos}(\mathcal{K}) \subseteq T \text{ and } \mathsf{Neg}(\mathcal{K}) \cap T = \emptyset.$$

If $T = \tau_\phi(\mathcal{I})$ for an $\mathcal{ALC}$-interpretation $\mathcal{I}$, then we obviously have $\mathcal{I} \models \mathcal{K}$ iff $\mathcal{K} <^\phi T$. We now lift the relation $<^\phi$ from (partial) types to sets of (partial) types.

Definition 4.6. *Let $\mathfrak{T}$ be a set of $\mathcal{ALC}$-types for ϕ and $\mathfrak{P}$ a set of partial $\mathcal{ALC}$-types for ϕ. We say that $\mathfrak{T}$* realizes *$\mathfrak{P}$ and write $\mathfrak{P} \prec^\phi \mathfrak{T}$ if the following property is satisfied: for every $\mathcal{K} \in \mathfrak{P}$, there is a $T \in \mathfrak{T}$ such that $\mathcal{K} <^\phi T$.*

This relation can be used to characterize r-consistency of a set of partial $\mathcal{ALC}$-types for ϕ.

Definition 4.7. *The set $\mathfrak{P} = \{\mathcal{K}_1, \ldots, \mathcal{K}_k\}$ of partial $\mathcal{ALC}$-types for ϕ is called* r-consistent *if there are $\mathcal{ALC}$-interpretations $\mathcal{I}_1, \ldots, \mathcal{I}_k$ that share the same domain, coincide on the individual names and the rigid concept and role names, and satisfy $\mathcal{I}_i \models \mathcal{K}_i$ for all i, $1 \leq i \leq k$. The set of all r-consistent sets of partial $\mathcal{ALC}$-types for ϕ is denoted by $\widehat{\mathcal{C}}_r^\phi$*

Lemma 4.8. *The set $\mathfrak{P}$ of partial $\mathcal{ALC}$-types for ϕ is r-consistent iff there is an r-consistent set $\mathfrak{T}$ of $\mathcal{ALC}$-types for ϕ such that $\mathfrak{P} \prec^\phi \mathfrak{T}$.*

Proof. First, assume that $\mathfrak{P} = \{\mathcal{K}_1, \ldots, \mathcal{K}_k\}$ is r-consistent. Then there are $\mathcal{ALC}$-interpretations $\mathcal{I}_1, \ldots, \mathcal{I}_k$ that share the same domain, coincide on the individual names and the rigid concept and role names, and satisfy $\mathcal{I}_i \models \mathcal{K}_i$ for all i, $1 \leq i \leq k$. If we define $\mathfrak{T} := \{\tau_\phi(\mathcal{I}_1), \ldots, \tau_\phi(\mathcal{I}_k)\}$, then this set of $\mathcal{ALC}$-types is obviously r-consistent and we have $\mathcal{K}_i <^\phi \tau_\phi(\mathcal{I}_i)$ for all $i, 1 \leq i \leq k$. This shows $\mathfrak{P} \prec^\phi \mathfrak{T}$.

Conversely, let $\mathfrak{P} = \{\mathcal{K}_1, \ldots, \mathcal{K}_k\}$ and assume that $\mathfrak{T} = \{T_1, \ldots, T_m\}$ is an r-consistent set of $\mathcal{ALC}$-types for ϕ such that $\mathfrak{P} \prec^\phi \mathfrak{T}$. Then there are $\mathcal{ALC}$-interpretations $\mathcal{I}_1, \ldots, \mathcal{I}_m$ that share the same domain, coincide on the individual names and the rigid concept and role names, and satisfy $T_i = \tau_\phi(\mathcal{I}_i)$ for all i, $1 \leq i \leq m$. In addition, for every $j, 1 \leq j \leq k$, there is an index $i_j, 1 \leq i_j \leq m$, such that $\mathcal{K}_j <^\phi T_{i_j}$. The $\mathcal{ALC}$-interpretations $\mathcal{I}_{i_1}, \ldots, \mathcal{I}_{i_k}$ share the same domain, coincide on the individual names and the rigid concept and role names, and satisfy $\mathcal{I}_{i_j} \models \mathcal{K}_j$ for all $j, 1 \leq j \leq k$. This shows that $\mathfrak{P}$ is r-consistent. □

We are now ready to define a deterministic automaton that accepts exactly those sequences of partial types $\mathfrak{K} \in (\mathfrak{P}^\phi)^*$ such that $\mathfrak{K}, \psi \not\approx_\forall \phi$. But first, for the sake of completeness, let us recall the definition of a deterministic finite automaton.

Definition 4.9. *A* deterministic finite automaton *$\mathcal{D} = (S, \Sigma, \delta, s_0, E)$ 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 $s_0 \in S$, and a set of final states $E \subseteq S$.*

The transition function can be extended to a function $\widehat{\delta} : S \times \Sigma^* \to S$ as follows:

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

The language accepted by $\mathcal{D}$ *is defined as by*

$$L(\mathcal{D}) := \{u \in \Sigma^* \mid \widehat{\delta}(s_0, u) \in E\}.$$

As mentioned above, the deterministic automaton to be defined is based on the Büchi automaton $\mathcal{N}^r_{\neg\phi\wedge\psi}$ for the $\mathcal{ALC}$-LTL-formula $\neg\phi \wedge \psi$. Recall that, according to our assumption, all the $\mathcal{ALC}$-axioms occurring in ψ already occur in ϕ. Thus, the alphabet of this automaton is actually $\mathfrak{T}^\phi$ and the second components of the states are r-consistent sets of $\mathcal{ALC}$-types for ϕ. Given a state $(q, \mathfrak{T})$ of $\mathcal{N}^r_{\neg\phi\wedge\psi}$, we denote the Büchi automaton obtained from this automaton by replacing the set of initial states with $\{(q, \mathfrak{T})\}$ by $\mathcal{N}^r_{\neg\phi\wedge\psi}(q, \mathfrak{T})$.

Definition 4.10. *Let* ϕ, ψ *be* $\mathcal{ALC}$*-LTL-formulae and let* $\mathcal{N}^r_{\neg\phi\wedge\psi} = (Q \times \mathcal{C}^\phi_r, \mathfrak{T}^\phi, \Delta, Q_0 \times \{\emptyset\}, F \times \mathcal{C}^\phi_r)$ *be the Büchi-automaton for* $\neg\phi \wedge \psi$*, as introduced in Theorem 3.19. The deterministic finite automaton* $\mathcal{D}^r_{\phi,\psi} := (S, \mathfrak{P}^\phi, \delta, s_0, E)$ *is defined as follows:*

- $S := 2^Q \times \widehat{\mathcal{C}}^\phi_r$;
- $s_0 := (Q_0, \emptyset)$;
- $\delta : S \times \mathfrak{P}^\phi \to S$ *is defined as follows:*
  - *if* $\mathfrak{P} \cup \{\mathcal{K}\} \notin \widehat{\mathcal{C}}^\phi_r$*, then* $\delta((P, \mathfrak{P}), \mathcal{K}) := (\emptyset, \mathfrak{P})$;
  - *if* $\mathfrak{P} \cup \{\mathcal{K}\} \in \widehat{\mathcal{C}}^\phi_r$*, then* $\delta((P, \mathfrak{P}), \mathcal{K}) := (P', \mathfrak{P} \cup \{\mathcal{K}\})$ *where*

$$P' := \bigcup_{q \in P} \Big\{ q' \in Q \mid \begin{array}{l} \text{there is } ((q, \mathfrak{T}), T, (q', \mathfrak{T} \cup \{T\})) \in \Delta \\ \text{such that } \mathcal{K} <^\phi T,\ \mathfrak{P} \prec^\phi \mathfrak{T}, \text{ and} \\ L_\omega(\mathcal{N}^r_{\neg\phi\wedge\psi}(q', \mathfrak{T} \cup \{T\})) \neq \emptyset \Big\}; \end{array}$$

- *the set of final states is defined as* $E := \{\emptyset\} \times \widehat{\mathcal{C}}^\phi_r$.

Final states are those where the first component is the empty set. Note that these states reproduce themselves: states for which the first component is empty have only successor states for which this is again the case. There are two possible reasons for reaching such a state with letter $\mathcal{K}$ from a state $(P, \mathfrak{P})$ whose first component P is non-empty. Either the set $\mathfrak{P} \cup \{\mathcal{K}\}$ is not r-consistent, or there are no states $q' \in Q$ satisfying the conditions in the definition of P'.

Lemma 4.11. *For all sequences of partial types* $\mathfrak{K} \in (\mathfrak{P}^\phi)^*$ *the following are equivalent:*

1. $\mathfrak{K}, \psi \mathrel{|\!\!\!\approx_{\forall}} \phi$,

2. $\mathfrak{K} \in L(\mathcal{D}^r_{\phi,\psi})$.

Proof. First, assume to the contrary that $\mathfrak{K} = \mathcal{K}_0\mathcal{K}_1 \dots \mathcal{K}_t \in L(\mathcal{D}^r_{\phi,\psi})$, but $\mathfrak{K}, \psi \not\mathrel{|\!\!\!\approx_{\forall}} \phi$. Then we have $\mathfrak{K}, \psi \mathrel{|\!\!\!\approx_{\exists}} \neg\phi$, i.e., there is an $\mathcal{ALC}$-LTL-structure $\mathfrak{I} = (\mathcal{I}_i)_{i\geq 0}$ that extends $\mathfrak{K}$ w.r.t. ψ, respects rigid names, and is a model of $\neg\phi$. This means that $\mathfrak{I}$ is a model of $\neg\phi \wedge \psi$ respecting rigid names, and $\mathcal{I}_i \models \mathcal{K}_i$ for every i, $0 \leq i \leq t$. Thus, $\tau_\phi(\mathfrak{I}) \in L^r_\omega(\neg\phi \wedge \psi)$, and since $\mathcal{N}^r_{\neg\phi\wedge\psi}$ is a Büchi automaton for $\neg\phi \wedge \psi$ respecting rigid names, we have $\tau_\phi(\mathfrak{I}) \in L_\omega(\mathcal{N}^r_{\neg\phi\wedge\psi})$. This means that there is an accepting run $(q_0, \mathfrak{T}_0)(q_1, \mathfrak{T}_1) \dots$ of $\mathcal{N}^r_{\neg\phi\wedge\psi}$ on $\tau_\phi(\mathfrak{I})$. In particular, this yields $L_\omega(\mathcal{N}^r_{\neg\phi\wedge\psi}(q_i, \mathfrak{T}_i)) \neq \emptyset$ for all $i \geq 0$.

Moreover, we have by the construction of $\mathcal{N}^r_{\neg\phi\wedge\psi}$ that $\mathfrak{T}_i = \{\tau_\phi(\mathcal{I}_j) \mid 0 \leq j < i\}$ for every $i \geq 0$. We define $\mathfrak{P}_i := \{\mathcal{K}_j \mid 0 \leq j < i\}$ for every i, $0 \leq i \leq t+1$. Note that we have $\mathcal{K}_i <^\phi \tau_\phi(\mathcal{I}_i)$ for every i, $0 \leq i \leq t+1$. Hence, $\mathfrak{P}_i \prec^\phi \mathfrak{T}_i$ holds for every i, $0 \leq i \leq t+1$. By the definition of $\mathcal{N}^r_{\neg\phi\wedge\psi}$, the sets $\mathfrak{T}_i$ are r-consistent, and thus Lemma 4.8 yields that $\mathfrak{P}_i$ is r-consistent for every i, $0 \leq i \leq t+1$. Thus, we have $\widehat{\delta}(s_0, \mathcal{K}_0 \dots \mathcal{K}_i) = (P_{i+1}, \mathfrak{P}_{i+1})$ with $q_{i+1} \in P_{i+1}$ for every i, $0 \leq i \leq t$. In particular, $\widehat{\delta}(s_0, \mathfrak{K}) = (P_{t+1}, \mathfrak{P}_{t+1})$ with $q_{t+1} \in P_{t+1}$, which shows that $P_{t+1} \neq \emptyset$. Consequently, we have $(P_{t+1}, \mathfrak{P}_{t+1}) \notin E$, which is a contradiction to our assumption that $\mathfrak{K} \in L(\mathcal{D}^r_{\phi,\psi})$.

Conversely, assume to the contrary that $\mathfrak{K} = \mathcal{K}_0\mathcal{K}_1 \dots \mathcal{K}_t \notin L(\mathcal{D}^r_{\phi,\psi})$ and $\mathfrak{K}, \psi \mathrel{|\!\!\!\approx_{\forall}} \phi$, i.e., every $\mathcal{ALC}$-LTL-structure $\mathfrak{I} = (\mathcal{I}_i)_{i\geq 0}$ that extends $\mathfrak{K}$ w.r.t. ψ and respects rigid names is a model of ϕ. The first assumption implies that $\widehat{\delta}(s_0, \mathfrak{K}) \notin E$, i.e., $\widehat{\delta}(s_0, \mathfrak{K}) = (P_{t+1}, \mathfrak{P}_{t+1}) \in 2^Q \times \widehat{\mathcal{C}}^\phi_r$ with $P_{t+1} \neq \emptyset$. This yields intermediate states $(P_i, \mathfrak{P}_i) \in 2^Q \times \widehat{\mathcal{C}}^\phi_r$ $(0 \leq i \leq t)$ such that $P_0 = Q_0, \mathfrak{P}_0 = \emptyset$ and $\widehat{\delta}(s_0, \mathcal{K}_0 \dots \mathcal{K}_i) = (P_{i+1}, \mathfrak{P}_{i+1}) \in 2^Q \times \widehat{\mathcal{C}}^\phi_r$ with $\mathfrak{P}_{i+1} = \mathfrak{P}_i \cup \{\mathcal{K}_i\}$ and $P_{i+1} \neq \emptyset$ for every i, $0 \leq i \leq t$. Moreover, we have that there are, for every i, $0 \leq i \leq t$, a state $q_i \in P_i$, a type $T_i \in \mathfrak{T}^\phi$, and an r-consistent set of types $\mathfrak{T}_i \in \mathcal{C}^\phi_r$ such that $((q_i, \mathfrak{T}_i), T_i, (q_{i+1}, \mathfrak{T}_{i+1})) \in \Delta$, $\mathfrak{T}_{i+1} = \mathfrak{T}_i \cup \{T_i\}$, $\mathcal{K}_i <^\phi T_i$, $\mathfrak{P}_i \prec^\phi \mathfrak{T}_i$, and $L_\omega(\mathcal{N}^r_{\neg\phi\wedge\psi}(q_{i+1}, \mathfrak{T}_{i+1})) \neq \emptyset$. Note that $q_0 \in Q_0$ since $q_0 \in P_0$ and $P_0 = Q_0$.

We define $\mathfrak{T}'_i := \{T_j \mid 0 \leq j < i\}$ for every i, $0 \leq i \leq t+1$. Obviously, we then have $\mathfrak{T}'_i \subseteq \mathfrak{T}_i$ for every i, $0 \leq i \leq t+1$. Since every subset of an r-consistent set of types is again r-consistent, this shows $\mathfrak{T}'_i \in \mathcal{C}^\phi_r$ for every i, $0 \leq i \leq t+1$. Moreover, since $\mathfrak{P}_i = \{\mathcal{K}_j \mid 0 \leq j < i\}$ for every i, $0 \leq i \leq t+1$, the fact that $\mathcal{K}_i <^\phi T_i$ for every i, $0 \leq i \leq t+1$, implies $\mathfrak{P}_i \prec^\phi \mathfrak{T}'_i$ for every i, $0 \leq i \leq t+1$. In addition, we have $((q_i, \mathfrak{T}'_i), T_i, (q_{i+1}, \mathfrak{T}'_{i+1})) \in \Delta$ for every i, $0 \leq i \leq t$.

Since $L_\omega(\mathcal{N}^r_{\neg\phi\wedge\psi}(q_{t+1}, \mathfrak{T}_{t+1})) \neq \emptyset$, there is an ω-word $T \in (\mathfrak{T}^\phi)^\omega$ such that there is an accepting run of $\mathcal{N}^r_{\neg\phi\wedge\psi}(q_{t+1}, \mathfrak{T}_{t+1})$ on T. Using similar arguments as above, we can transform this run into an accepting run of $\mathcal{N}^r_{\neg\phi\wedge\psi}(q, \mathfrak{T}'_{t+1})$ on T. Hence, $T \in L_\omega(\mathcal{N}^r_{\neg\phi\wedge\psi}(q, \mathfrak{T}'_{t+1}))$. Overall, we obtain that the ω-word $T_0T_1 \dots T_t \cdot T$ is in $L_\omega(\mathcal{N}^r_{\neg\phi\wedge\psi})$. Since $\mathcal{N}^r_{\neg\phi\wedge\psi}$ is a Büchi automaton for $\neg\phi \wedge \psi$ respecting rigid names, this shows that there exists an $\mathcal{ALC}$-LTL-structure $\mathfrak{I} = (\mathcal{I}_i)_{i\geq 0}$ respecting rigid names such that $\tau_\phi(\mathfrak{I}) = T_0 T_1 \dots T_t \cdot T$ and $\mathfrak{I}$ is a model of $\neg\phi \wedge \psi$. For every i, $0 \leq i \leq t$, we have $\mathcal{K}_i <^\phi \tau_\phi(\mathcal{I}_i)$ since $\tau_\phi(\mathcal{I}_i) = T_i$. This yields $\mathcal{I}_i \models \mathcal{K}_i$ for every i, $0 \leq i \leq t$. Since $\mathfrak{I}$ is a model of ψ, we obtain that $\mathfrak{I}$ extends $\mathfrak{K}$ w.r.t. ψ. Hence, there is an $\mathcal{ALC}$-LTL-structure, namely $\mathfrak{I}$, that extends $\mathfrak{K}$ w.r.t. ψ, respects rigid names, and is a model of $\neg\phi$, which contradicts our assumption that $\mathfrak{K}, \psi \models_\forall \phi$. $\square$

It remains to analyze the complexity of the construction of the deterministic finite automaton $\mathcal{D}^r_{\phi,\psi}$. The size of $\mathcal{D}^r_{\phi,\psi}$ is double-exponential in the size of ϕ and ψ. This is due to the fact that the size of Q may be exponential and the fact that the set $\widehat{\mathcal{C}}^\phi_r$ of all r-consistent partial types for ϕ may contain double-exponentially many elements since these sets are subsets of the exponentially large set $\mathfrak{P}^\phi$ of all partial types for ϕ. Each element of $\widehat{\mathcal{C}}^\phi_r$ may be of exponential size.

Next, we show that $\mathcal{D}^r_{\phi,\psi}$ can be constructed in double-exponential time. In addition to constructing the Büchi automaton $\mathcal{N}^r_{\neg\phi\wedge\psi}$, we must also compute the set $\widehat{\mathcal{C}}^\phi_r$. As shown in Section 3.4, the Büchi automaton $\mathcal{N}^r_{\neg\phi\wedge\psi}$, and thus also the set $\mathcal{C}^\phi_r$, can be constructed in time double-exponential in the size of ϕ and ψ. To compute $\widehat{\mathcal{C}}^\phi_r$, we use Lemma 4.8, which yields

$$\widehat{\mathcal{C}}^\phi_r = \left\{\mathfrak{P} \subseteq \mathfrak{P}^\phi \mid \mathfrak{P} \prec^\phi \mathfrak{T} \text{ for some } \mathfrak{T} \in \mathcal{C}^\phi_r\right\}.$$

We consider all sets of partial types for ϕ. There are double-exponentially many such sets, each of size at most exponential in the size of ϕ. For each such set $\mathfrak{P} = \{\mathcal{K}_1, \dots, \mathcal{K}_k\}$, we need to check whether there is a set $\mathfrak{T} = \{T_1, \dots, T_m\} \in \mathcal{C}^\phi_r$ such that $\mathfrak{P} \prec^\phi \mathfrak{T}$. Since $\mathcal{C}^\phi_r$ is of double-exponential size, there are at most double-exponentially many such tests for each $\mathfrak{P}$. The test $\mathfrak{P} \prec^\phi \mathfrak{T}$ itself amounts to checking, for each $\mathcal{K}_i$, $1 \leq i \leq k$, whether there is a T_j, $1 \leq j \leq m$, such that $\mathsf{Pos}(\mathcal{K}_i) \subseteq T_j$ and $\mathsf{Neg}(\mathcal{K}_i) \cap T_j = \emptyset$, which can be done in exponential time since both k and m are at most exponential in the size of ϕ. Overall, we can thus compute $\widehat{\mathcal{C}}^\phi_r$ is double-exponential time. Using these arguments, the fact that $\mathcal{N}^r_{\neg\phi\wedge\psi}$ can be constructed in double-exponential time, and the fact that the emptiness problem for Büchi automata can be solved in time polynomial in the size of the Büchi automaton [VW94], it is easy to see that the transition function δ and the set of final states E can be computed in double-exponential time. Overall, we have shown that $\mathcal{D}^r_{\phi,\psi}$ can be constructed in time double-exponential in the size of ϕ and ψ.

4.3 The Monitor Construction

Given the construction of the deterministic automaton of the previous subsection, it no a simple exercise to construct the monitor for ϕ w.r.t. ψ. Such a monitor is obtained by first constructing the auxiliary deterministic finite automata $\mathcal{D}^r_{\phi,\psi}$

and $\mathcal{D}^r_{\neg\phi,\psi}$, and then building the product of these two automata. The output of the monitor is determined by the final states of the auxiliary automata.

Theorem 4.12. *Let ϕ and ψ be $\mathcal{ALC}$-LTL-formulae. If $\mathcal{D}^r_{\phi,\psi} = (S, \mathfrak{P}^\phi, \delta, s_0, E)$ and $\mathcal{D}^r_{\neg\phi,\psi} = (S', \mathfrak{P}^\phi, \delta', s_0', E')$ are the deterministic finite automata introduced in Section 4.2, then $\mathcal{M}_{\phi,\psi} := (S \times S', \mathfrak{P}^\phi, \delta'', (s_0, s_0'), \{\top, \bot, ?, \lightning\}, \lambda)$ with $\delta''((s, s'), \mathcal{K}) := (\delta(s, \mathcal{K}), \delta'(s', \mathcal{K}))$ and*

$$\lambda((s,s')) := \begin{cases} \top & \text{if } s \in E \text{ and } s' \notin E' \\ \bot & \text{if } s \notin E \text{ and } s' \in E' \\ ? & \text{if } s \notin E \text{ and } s' \notin E' \\ \lightning & \text{if } s \in E \text{ and } s' \in E' \end{cases}$$

is a monitor for ϕ w.r.t. ψ.

Proof. The theorem is an immediate consequence of the following facts:

- $\widehat{\delta''}((s_0, s_0'), \mathfrak{K}) = (\widehat{\delta}(s_0, \mathfrak{K}), \widehat{\delta'}(s_0', \mathfrak{K}))$;
- $\widehat{\delta}(s_0, \mathfrak{K}) \in E$ iff $\mathfrak{K}, \psi \mathrel{|\!\!\approx_\forall} \phi$ (by Lemma 4.11);
- $\widehat{\delta'}(s_0', \mathfrak{K}) \in E'$ iff $\mathfrak{K}, \psi \mathrel{|\!\!\approx_\forall} \neg\phi$ (by Lemma 4.11);

and the definition of the monitoring function (Definition 4.4). □

It remains to analyze the complexity of the construction. As shown in Section 4.2, the size of the auxiliary deterministic finite automata $\mathcal{D}^r_{\phi,\psi}$ and $\mathcal{D}^r_{\neg\phi,\psi}$ is double-exponential in the size of ϕ and ψ. Furthermore, they can be constructed in double-exponential time. Hence, the size of $\mathcal{M}_{\phi,\psi}$ is also double-exponential in the size of ϕ and ψ, and it can be constructed in double-exponential time.

This double-exponential blow-up in the construction of the monitor cannot be avoided, since Proposition 2.8 yields that such a blow-up is unavoidable even for propositional LTL.

5 The Complexity of Liveness and Monitorability in $\mathcal{ALC}$-LTL

In this section, we extend the definitions and results about liveness and monitorability from propositional LTL to $\mathcal{ALC}$-LTL.

5.1 Deciding Liveness

First, we extend the notion of liveness from propositional LTL (see Definition 2.10) to $\mathcal{ALC}$-LTL and the presence of background knowledge.

Definition 5.1. *Let ϕ and ψ be $\mathcal{ALC}$-LTL-formulae. We say that ϕ* expresses a liveness property w.r.t. ψ *if, for every finite sequence of partial types $\mathfrak{K} \in (\mathfrak{P}^{\phi})^*$ that has an extension w.r.t. ψ respecting rigid names, we have $\mathfrak{K}, \psi \mathrel{|\!\!\approx_\exists} \phi$.*

Note that, in this definition, we restrict ourselves to finite sequences of partial types that have an extension w.r.t. ψ respecting rigid names. In fact, these are the sequences that we expect to see in practice since we assume that the system satisfies ψ and respects rigid names.

As in the propositional case, liveness of ϕ w.r.t. ψ can be expressed using the monitoring function.

Lemma 5.2. *Let ϕ and ψ be $\mathcal{ALC}$-LTL-formulae. Then ϕ expresses a liveness property w.r.t. ψ iff $m_{\phi,\psi}(\mathfrak{K}) \neq \bot$, for every $\mathfrak{K} \in (\mathfrak{P}^{\phi})^*$.*

Proof. First, assume that ϕ expresses a liveness property w.r.t. ψ, and consider $\mathfrak{K} \in (\mathfrak{P}^{\phi})^*$. If $\mathfrak{K}$ does not have an extension w.r.t. ψ respecting rigid names, then $m_{\phi,\psi}(\mathfrak{K}) = \lightning \neq \bot$. Otherwise, the fact that ϕ expresses a liveness property w.r.t. ψ implies that $\mathfrak{K}, \psi \mathrel{|\!\!\approx_\exists} \phi$. Consequently, we have $\mathfrak{K}, \psi \mathrel{|\!\!\not\approx_\forall} \neg\phi$, which yields $m_{\phi,\psi}(\mathfrak{K}) \neq \bot$.

Second, assume that $m_{\phi,\psi}(\mathfrak{K}) \neq \bot$, for every $\mathfrak{K} \in (\mathfrak{P}^{\phi})^*$. Consider a finite sequence of partial types $\mathfrak{K} \in (\mathfrak{P}^{\phi})^*$ that has an extension w.r.t. ψ respecting rigid names. The existence of this extension implies that $m_{\phi,\psi}(\mathfrak{K}) \neq \lightning$. Thus, we know that $m_{\phi,\psi}(\mathfrak{K}) \in \{\top, ?\}$. In both cases, $\mathfrak{K}, \psi \mathrel{|\!\!\not\approx_\forall} \neg\phi$ holds, which is equivalent, to $\mathfrak{K}, \psi \mathrel{|\!\!\approx_\exists} \phi$. □

Consequently, given a monitor for ϕ w.r.t. ψ, liveness of ϕ w.r.t. ψ can be tested by checking reachability in the monitor, which yields an upper bound of 2-ExpTime. The lower bound can be obtained by a reduction of unsatisfiability in $\mathcal{ALC}$-LTL.

Theorem 5.3. *The problem of deciding whether an $\mathcal{ALC}$-LTL-formula ϕ expresses a liveness property w.r.t. an $\mathcal{ALC}$-LTL-formula ψ is 2-*ExpTime*-complete.*

Proof. Regarding the upper bound, Lemma 5.2 implies that ϕ expresses a liveness property w.r.t. ψ iff in the monitor $\mathcal{M}_{\phi,\psi}$ no state with output $\bot$ is reachable from the initial state. Since this monitor is of double-exponential size and reachability can be decided in linear time in the size of the automaton, this yields the required 2-ExpTime upper bound.

To show the 2-ExpTime lower bound, we reduce the 2-ExpTime-complete problem of unsatisfiability in $\mathcal{ALC}$-LTL w.r.t. rigid names [BGL12] to the liveness problem. To this purpose, consider an $\mathcal{ALC}$-LTL-formula ψ. We prove that ψ is unsatisfiable w.r.t. rigid names iff the $\mathcal{ALC}$-LTL-formula $\phi := \mathsf{false}$ expresses a liveness property w.r.t. ψ.

In fact, if ψ is unsatisfiable w.r.t. rigid names, then there is *no* sequence $\mathfrak{K} \in (\mathfrak{P}^\phi)^*$ such that $\mathfrak{K}$ has extension w.r.t. ψ respecting rigid names. Consequently, the condition in the definition of liveness quantifies over the empty set of sequences, and is thus trivially true.

Conversely, if ψ is satisfiable, then there is some $\mathfrak{K} \in (\mathfrak{P}^\phi)^*$ (e.g. the empty sequence) that has an extension w.r.t. ψ respecting rigid names. But then $\mathfrak{K}, \psi \not\models\!\!\approx_\exists \phi$ since ϕ is unsatisfiable. Hence, ϕ does not express a liveness property w.r.t. ψ. $\square$

Note that our 2-ExpTime-hardness proof strongly depends on the presence of background knowledge. Without background knowledge (i.e., in the case where $\psi = \mathsf{true}$), we can only show an ExpTime-hardness result by a reduction of satisfiability in $\mathcal{ALC}$-LTL without rigid names.

Proposition 5.4. *The problem of deciding whether an $\mathcal{ALC}$-LTL-formula ϕ expresses a liveness property w.r.t. the $\mathcal{ALC}$-LTL-formula* true *is* ExpTime*-hard.*

Proof. Consider an $\mathcal{ALC}$-LTL-formula ϕ that does not contain rigid names. We prove that ϕ is satisfiable iff $\Diamond\phi$ expresses a liveness property w.r.t. true. Since satisfiability in $\mathcal{ALC}$-LTL without rigid names is ExpTime-complete [BGL12], this shows ExpTime-hardness of liveness w.r.t. true.

If ϕ is unsatisfiable, then obviously $\Diamond\phi$ is unsatisfiable as well, and thus no sequence of partial types can be extended to a model of $\Diamond\phi$. In addition, there is a sequence of partial types (e.g., the empty sequence) that can be extended to a model of true. Consequently, $\Diamond\phi$ does not express a liveness property w.r.t. true.

Conversely, assume that ϕ is satisfiable, and let $\mathfrak{K} = \mathcal{K}_0\mathcal{K}_1 \dots \mathcal{K}_{t-1} \in (\mathfrak{P}^\phi)^*$ be a sequence of partial types. Satisfiability of ϕ yields a model $\mathfrak{I} = (\mathcal{I}_i)_{i\geq 0}$ of ϕ. In addition, since partial types are by definition consistent, there are $\mathcal{ALC}$-interpretations $\mathcal{I}'_i$ $(i = 0, 1, \dots, t-1)$ such that $\mathcal{I}'_i \models \mathcal{K}_i$. It is easy to see that the $\mathcal{ALC}$-LTL-structure

$$\mathfrak{J} = (\mathcal{J}_i)_{i\geq 0} \text{ with } \mathcal{J}_i = \mathcal{I}'_i \ (i = 0, 1, \dots, t-1) \text{ and } \mathcal{J}_{i+t} = \mathcal{I}_i \ (i \geq 0)$$

is a model of $\Diamond\phi$ that extends $\mathfrak{K}$ w.r.t. true, i.e., $\mathfrak{K}, \mathsf{true} \models\!\!\approx_\exists \phi$. This shows that $\Diamond\phi$ expresses a liveness property w.r.t. true. $\square$

Unfortunately, the proof of this proposition does not go through in the presence of rigid names. In fact, the $\mathcal{ALC}$-LTL-structure $\mathfrak{J}$ constructed there need not satisfy rigid names.

5.2 Deciding Monitorability

We first extend the notion of monitorability from propositional LTL (see Definition 2.9) to $\mathcal{ALC}$-LTL and the presence of background knowledge.

Definition 5.5. *Let ϕ and ψ be $\mathcal{ALC}$-LTL-formulae, and let $\mathfrak{K} \in (\mathfrak{P}^{\phi})^*$. We say that ϕ is* $\mathfrak{K}$-monitorable w.r.t. ψ *if there is a finite word $\mathfrak{K}' \in (\mathfrak{P}^{\phi})^*$ such that $m_{\phi,\psi}(\mathfrak{K} \cdot \mathfrak{K}') \in \{\top, \bot\}$. Moreover, we call ϕ* monitorable w.r.t. ψ *if it is $\mathfrak{K}$-monitorable for every finite sequence of partial types $\mathfrak{K} \in (\mathfrak{P}^{\phi})^*$ that has an extension w.r.t. ψ respecting rigid names.*

Monitorability can thus be expressed using the monitoring function as follows: ϕ is monitorable w.r.t. ψ iff for every finite sequence of partial types $\mathfrak{K} \in (\mathfrak{P}^{\phi})^*$ with $m_{\phi,\psi}(\mathfrak{K}) \neq \lightning$ there exists a finite sequence of partial types $\mathfrak{K}' \in (\mathfrak{P}^{\phi})^*$ satisfying $m_{\phi,\psi}(\mathfrak{K} \cdot \mathfrak{K}') \in \{\top, \bot\}$. This can again be checked using reachability tests in the monitor.

Lemma 5.6. *The problem of deciding monitorability of an $\mathcal{ALC}$-LTL-formula ϕ w.r.t. an $\mathcal{ALC}$-LTL-formula ψ is decidable in double-exponential time*

Proof. To decide monitorability of ϕ w.r.t. ψ, we construct the monitor $\mathcal{M}_{\phi,\psi}$. In this monitor, we compute all the state with output different from $\lightning$ that reachable from the initial state. For each of these states we then check whether a state with output $\top$ or $\bot$ is reachable. If this is the case, then ϕ is monitorable w.r.t. ψ. Otherwise, i.e., if there is a state reachable from the initial state such that every state reachable from it has output ? or $\lightning$, then ϕ is not monitorable w.r.t. ψ.

Since the monitor can be constructed in double-exponential time and each of the double-exponentially many reachability test requires at most double-exponential time, this yields a 2-ExpTime procedure for deciding monitorability. □

To show the matching lower bound, we again reduce the 2-ExpTime-complete unsatisfiability problem in $\mathcal{ALC}$-LTL w.r.t. rigid names. For monitoring, such a reduction is possible even for the case without background knowledge.

Lemma 5.7. *The problem of deciding monitorability of an $\mathcal{ALC}$-LTL-formula ϕ w.r.t.* `true` *is 2-ExpTime-hard.*

Proof. Note that 2-ExpTime-hardness for the satisfiability problem in $\mathcal{ALC}$-LTL holds already for $\mathcal{ALC}$-LTL-formulae without past operators [BGL12]. Thus, let ψ be an $\mathcal{ALC}$-LTL-formula without past operators. We define the $\mathcal{ALC}$-LTL-formula ϕ as $\phi := \Diamond\psi \wedge \Box\Diamond A(a)$ where the (flexible) concept name A and the individual name a do not occur in ψ. We prove that ψ is unsatisfiable iff ϕ is monitorable w.r.t. `true`.

If ψ is unsatisfiable w.r.t. rigid names, then $\phi \equiv \Diamond\mathsf{false} \wedge \Box\Diamond A(a) \equiv \mathsf{false} \wedge \Box\Diamond A(a) \equiv \mathsf{false}$, i.e., ϕ is also unsatisfiable. Take now any $\mathfrak{K} \in (\mathfrak{P}^\phi)^*$ that has an extension w.r.t. true respecting rigid names. Since ϕ is unsatisfiable, we have $\mathfrak{K}, \mathsf{true} \models_{\forall}^{\approx} \neg\phi$. Thus, Lemma 4.3 yields $\mathfrak{K}, \mathsf{true} \not\models_{\forall}^{\approx} \phi$, which shows that $m_{\phi,\mathsf{true}}(\mathfrak{K}) = \bot$. Consequently, ϕ is $\mathfrak{K}$-monitorable w.r.t. true (take $\mathfrak{K}'$ to be the empty word). Since $\mathfrak{K}$ was an arbitrary element of $(\mathfrak{P}^\phi)^*$ that has an extension w.r.t. true respecting rigid names, this shows that ϕ is monitorable w.r.t. true.

Conversely, if ψ is satisfiable w.r.t. rigid names, then there is a model $\mathfrak{I} = (\mathcal{I}_i)_{i\geq 0}$ of ψ that respects rigid names. We define

$$\mathcal{K}_i := \bigwedge_{\alpha \in \tau_\psi(\mathcal{I}_i)} \alpha \wedge \bigwedge_{\alpha \in \mathsf{Ax}(\psi)\setminus\tau_\psi(\mathcal{I}_i)} \neg\alpha$$

for every $i \geq 0$. Obviously, $\mathcal{I}_i$ is a model of $\mathcal{K}_i$, and thus $\mathcal{K}_i$ is a partial type, i.e., $\mathcal{K}_i \in \mathfrak{P}^\phi$ for every $i \geq 0$. Since there are only finitely many partial types, there are finitely many partial types $\mathcal{K}'_1, \dots, \mathcal{K}'_k$ such that $\{\mathcal{K}'_1, \dots, \mathcal{K}'_k\} = \{\mathcal{K}_i \mid i \geq 0\}$. Then there is a surjective function $\nu : \{i \mid i \geq 0\} \to \{1, \dots, k\}$ such that $\mathcal{K}_i = \mathcal{K}'_{\nu(i)}$.

To show that ϕ is not monitorable w.r.t. true, we consider the finite sequence of partial types $\mathfrak{K} := \mathcal{K}'_1 \dots \mathcal{K}'_k$. Since the function ν is surjective, any partial type $\mathcal{K}'_i$ in this sequence has at least one of the interpretation $\mathcal{I}_j$ as model, and since $\mathfrak{I} = (\mathcal{I}_i)_{i\geq 0}$ respects rigid names, these models of $\mathcal{K}'_1, \dots, \mathcal{K}'_k$ share the same domain and coincide on the individual names and the rigid concept and role names. Consequently, the set $\{\mathcal{K}'_1, \dots, \mathcal{K}'_k\}$ is r-consistent. Obviously, this implies that $\mathfrak{K}$ has an extension w.r.t. true that respects rigid names.

To disprove monitorability of ϕ w.r.t. true, it is thus sufficient to show that ϕ is not $\mathfrak{K}$-monitorable w.r.t. true. For this purpose, we take any finite sequence $\mathfrak{K}' = \mathcal{K}''_1 \dots \mathcal{K}''_m \in (\mathfrak{P}^\phi)^*$ and show that $m_{\phi,\mathsf{true}}(\mathfrak{K} \cdot \mathfrak{K}') \notin \{\top, \bot\}$.

If $\mathfrak{K} \cdot \mathfrak{K}'$ does not have an extension w.r.t. true (which can happen due to the presence of rigid names), then $\mathfrak{K} \cdot \mathfrak{K}', \mathsf{true} \models_{\forall}^{\approx} \phi$ and $\mathfrak{K} \cdot \mathfrak{K}', \mathsf{true} \models_{\forall}^{\approx} \neg\phi$, and thus $m_{\phi,\mathsf{true}}(\mathfrak{K} \cdot \mathfrak{K}') = \lightning \notin \{\top, \bot\}$ as required.

Otherwise, let $\mathfrak{J} = (\mathcal{J}_i)_{i\geq 0}$ be an extension of $\mathfrak{K} \cdot \mathfrak{K}'$ w.r.t. true respecting rigid names. We define a new $\mathcal{ALC}$-LTL-structure $\mathfrak{J}' := (\mathcal{J}'_i)_{i\geq 0}$ with

$$\mathcal{J}'_i := \begin{cases} \mathcal{J}_i & \text{if } 0 \leq i \leq k+m-1; \text{ and} \\ \mathcal{J}_{\nu(i-k-m)} & \text{otherwise.} \end{cases}$$

By definition, $\mathfrak{J}'$ coincides with with $\mathfrak{J}$ on the first $k+m$ structures, which shows that it extends $\mathfrak{K} \cdot \mathfrak{K}'$ w.r.t. true. In addition, it respects rigid names since it consists of interpretations occurring in $\mathfrak{J}$, and $\mathfrak{J}$ respects rigid names. Moreover, since every $\mathcal{K}'_i$, $1 \leq i \leq k$, contains complete information about the axioms in ψ, we have that

$$\tau_\psi(\mathfrak{J}) = \tau_\psi(\mathcal{J}'_{k+m})\tau_\psi(\mathcal{J}'_{k+m+1}) \dots.$$

By Lemma 3.8, his shows that $(\mathcal{J}'_{k+m+i})_{ß\geq 0}$ is a model of ψ. Since ψ does not contain past operators, This implies that $\mathfrak{J}'$ is a model of $\Diamond\psi$. Since ψ does not contain the concept name A and the individual name a, this is independent on how A and a are interpreted. In addition, since A is flexible, changing its interpretation does not change the fact that rigid names are respected.

Let now $\mathfrak{J}_A$ and $\mathfrak{J}_{\neg A}$ be $\mathcal{ALC}$-LTL-structures such that

1. $\mathfrak{J}_A$ and $\mathfrak{J}_{\neg A}$ coincide for all points in time with $\mathfrak{J}'$ on the interpretation domain as well as on the interpretation of all individual names, role names, and concept names different from A,
2. $\mathfrak{J}_A$ and $\mathfrak{J}_{\neg A}$ coincide with $\mathfrak{J}'$ for all points in time up to $k+m-1$ also on the interpretation of A,
3. In $\mathfrak{J}_A$, the interpretation of A consists of the individual interpreting a at all points in time strictly after $k+m-1$,
4. In $\mathfrak{J}_{\neg A}$, the interpretation of A is empty at all points in time strictly after $k+m-1$.

Obviously, both $\mathfrak{J}_A$ and $\mathfrak{J}_{\neg A}$ are models of $\Diamond\psi$ respecting rigid names and they extend $\mathfrak{K}\cdot\mathfrak{K}'$. However, only $\mathfrak{J}_A$ is also a model of $\Box\Diamond A(a)$. Thus, $\mathfrak{J}_A$ is an extension of $\mathfrak{K}\cdot\mathfrak{K}'$ w.r.t. true respecting rigid names that satisfies ϕ, and $\mathfrak{J}_{\neg A}$ is an extension of $\mathfrak{K}\cdot\mathfrak{K}'$ w.r.t. true respecting rigid names that satisfies $\neg\phi$. This shows that we have $\mathfrak{K}\cdot\mathfrak{K}', \mathsf{true} \not\approx_\forall \phi$ and $\mathfrak{K}\cdot\mathfrak{K}', \mathsf{true} \not\approx_\forall \neg\phi$. Consequently, $m_{\phi,\mathsf{true}}(\mathfrak{K}\cdot\mathfrak{K}') = ? \notin \{\top,\bot\}$, which finishes the proof that ϕ is not monitorable w.r.t. true. □

Putting the two previous lemmas together, we obtain the following theorem.

Theorem 5.8. *The problem of deciding monitorability of an $\mathcal{ALC}$-LTL-formula ϕ w.r.t. an $\mathcal{ALC}$-LTL-formula ψ is 2-*ExpTime*-complete. The lower bound of 2-*ExpTime* already holds for the special case where $\phi = \mathsf{true}$.*

6 Conclusion

We have shown that known approaches to runtime verification in propositional LTL [BLS06] 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, and some known properties of the system can be specified as background knowledge. The complexity of the monitor construction is quite high (double exponential), but this blowup already occurs in the propositional case and we have shown that (in the worst case) it cannot be avoided. It should also be notes that 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. The double exponential complexity of the monitor construction is a worst-case complexity. Minimization of the intermediate Büchi automata and the monitor may lead to much smaller automata than the ones defined above. We have observed this behaviour on several example formulae for which we constructed monitors.

One important topic for future research are extensions towards probabilistic reasoning. Probabilities may come into our monitoring framework for a variety of reasons. For example, sensors used to observe the system and generate the ABoxes may be erroneous with some probability. Another interesting topic is the integration of numerical sensor values. These values could, e.g., be represented using Description Logics with so-called concrete domains [Lut03], which may, however, cause computational problems even without a temporal component.

References

[AF00] Alessandro Artale and Enrico Franconi. A survey of temporal extensions of description logics. *Annals of Mathematics and Artificial Intelligence*, 30(1–4):171–210, 2000.

[AF01] Alessandro Artale and Enrico Franconi. Temporal description logics. In Michael Fisher, Dov Gabbay, and Lluís Vila, editors, *Handbook of Temporal Reasoning in Artificial Intelligence.* 2001.

[AS85] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[Bau10] Andreas Bauer. Monitorability of ω-regular languages. Computing Research Repository (CoRR) abs/1006.3638, Association for Computing Machinery (ACM), 2010. http://arxiv.org/abs/1006.3638.

[BBL09] Franz Baader, Andreas Bauer, and Marcel Lippmann. Runtime verification using a temporal description logic. In Silvio Ghilardi and Roberto Sebastiani, editors, *Proceedings of the 7th International Symposium on Frontiers of Combining Systems (FroCoS 2009)*, volume 5749 of *Lecture Notes in Computer Science*, pages 149–164, Trento, Italy, September 2009. Springer-Verlag.

[BGL08] Franz Baader, Silvio Ghilardi, and Carsten Lutz. LTL over description logic axioms. In Gerhard Brewka and Jérôme Lang, editors, *Proceedings of the 11th International Conference on Principles of Knowledge*

Representation and Reasoning (KR 2008), pages 684–694, Los Altos, 2008. Morgan Kaufmann.

[BGL12] Franz Baader, Silvio Ghilardi, and Carsten Lutz. LTL over description logic axioms. *ACM Transactions on Computational Logic*, 13(3), 2012. Extended version of [BGL08].

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* The MIT Press, Cambridge, Massachusetts, USA, 2008.

[BLS06] Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In S. Arun-Kumar and Naveen Garg, editors, *Proceedings of the 26th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2006)*, volume 4337 of *Lecture Notes in Computer Science*, pages 260–272. Springer-Verlag, 2006.

[BLS10] Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing LTL semantics for runtime verification. *Journal of Logic and Computation*, 20(3):651–674, 2010.

[BLS11] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology*, 20(4):14, 2011.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking.* The MIT Press, 1999.

[CM04] 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, Advanced Lectures*, volume 3472 of *Lecture Notes in Computer Science.* Springer-Verlag, 2004.

[FFM09] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. Runtime verification of safety-progress properties. In Saddek Bensalem and Doron Peled, editors, *Selected Papers of the 9th International Workshop on Runtime Verification (RV 2009)*, volume 5779 of *Lecture Notes in Computer Science*, pages 40–59. Springer-Verlag, 2009.

[GKWZ03] Dov Gabbay, Agnes Kurusz, Frank Wolter, and M. Zakharyaschev. *Many-dimensional Modal Logics: Theory and Applications.* Elsevier, 2003.

[GO01] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the 13th International Conference on Computer Aided*

Verification (CAV 2001), volume 2102 of *Lecture Notes in Computer Science*, pages 53–65, Paris, France, 2001. Springer-Verlag.

[GO03] Paul Gastin and Denis Oddoux. LTL with past and two-way very-weak alternating automata. In Branislav Rovan and Peter Vojtás, editors, *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS 2003)*, volume 2747 of *Lecture Notes in Computer Science*, pages 439–448, Bratislava, Slovakia, 2003. Springer-Verlag.

[GPSS80] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *Proceedings of the 7th ACM Symposium on Principles of Programming Languages*, pages 163–173, 1980.

[GPVW96] Rob Gerth, Doron A. Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In Piotr Dembinski and Marek Sredniawa, editors, *Proceedings of the 15th IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification XV*, pages 3–18, London, UK, 1996. Chapman & Hall, Ltd.

[KR10] Orna Kupferman and Adin Rosenberg. The blowup in translating LTL to deterministic automata. In Ron van der Meyden and Jan-Georg Smaus, editors, *Revised Selected and Invited Papers of the 6th International Workshop on Model Checking and Artificial Intelligence MoChArt 2010*, volume 6572 of *Lecture Notes in Computer Science*, pages 85–94. Springer-Verlag, 2010.

[KV01] Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[LMS02] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Temporal logic with forgettable past. In *Proc. of the 17th Annual IEEE Symp. on Logic in Computer Science (LICS'02)*, pages 383–392. IEEE Computer Society Press, 2002.

[LPZ85] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The glory of the past. In Rohit Parikh, editor, *Proceedings of the Conference on Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer, 1985.

[Lut03] Carsten Lutz. Description logics with concrete domains—a survey. In Philippe Balbiani, Nobu-Yuki Suzuki, Frank Wolter, and Michael Zakharyaschev, editors, *Advances in Modal Logics Volume 4*, pages 265–296. King's College Publications, 2003.

[LWZ08] Carsten Lutz, Frank Wolter, and Michael Zakharyaschev. Temporal description logics: A survey. In Stéphane Demri and Christian S. Jensen, editors, *Proceedings of the 15th International Symposium on Temporal Representation and Reasoning (TIME 2008)*, pages 3–14. IEEE Computer Society Press, 2008.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pages 46–57, Providence, Rhode Island, USA, 1977. IEEE Computer Society Press.

[PZ06] Amir Pnueli and Aleksandr Zaks. PSL model checking and run-time verification via testers. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *Proceedings of the 14th International Symposium on Formal Methods (FM 2006)*, volume 4085 of *Lecture Notes in Computer Science*, pages 573–586, Hamilton, Canada, 2006. Springer-Verlag.

[SC85] A. Prasad Sistla and Edmund M. Clarke. The complexity of proposional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[SSS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[UNW01] Ulrich Ultes-Nitsche and Pierre Wolper. Checking properties within fairness and behavior abstractions. Computing Research Repository (CoRR) cs.LO/0101017, Association for Computing Machinery (ACM), 2001. http://arxiv.org/abs/cs.LO/0101017.

[VW94] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 155(1):1–37, 1994.

[WVS83] Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS 1983)*, pages 185–194. IEEE Computer Society Press, 1983.