

Technische Universität Dresden
Fakultät Informatik
Institut für theoretische Informatik
Professur für Automatentheorie

Diplomarbeit

Runtime Verification Using Temporal Description Logics

Marcel Lippmann

Studiengang: Informatik

Matrikelnummer: 3125591

26. März 2009

Betreuer und
verantwortlicher Hochschullehrer: Prof. Dr.-Ing. Franz Baader

To Francisca

Acknowledgements

First of all, I would like to thank my supervisor—Professor Franz Baader. Without his guidance and useful pointers this thesis would not have been possible. I am thankful for his patience and his helpful answers to my questions as well as his remarks.

I would like to thank also the teachers at Technische Universität Dresden for their motivating lectures on theoretical computer science—notably Professor Horst Reichel, Professor Franz Baader and Professor Christel Baier. Their lectures equipped me with the essential knowledge that was needed to finish this thesis.

I also appreciated useful hints concerning \LaTeX , given by Patrick Bahr, Stephan Krauß and Andreas Rümpel. I thank Michael Köhler for his careful proof-reading and hints on improving the English text.

Last, I would like to thank my parents, my family and my girlfriend Francisca for their constant support and motivation.

Contents

Acknowledgements	iii
1 Introduction and motivation	1
1.1 The main idea of runtime verification	1
1.2 Related work and contribution of this work	2
1.3 Overview	3
2 Preliminaries	4
2.1 Automata models	4
2.1.1 Finite-state machines with output	5
2.1.2 Non-deterministic Büchi automata	6
2.2 Temporal description logics	9
2.2.1 Description logics	9
2.2.2 Temporal logics	14
2.2.3 The combination: temporal description logics	17
2.3 Formal verification of linear-time properties	19
2.3.1 Explicit-state model checking for LTL	20
2.3.2 Runtime verification for LTL	26
3 Runtime verification for \mathcal{ALC}-LTL without rigid names	31
3.1 \mathcal{ALC} -LTL with and without rigid names	31
3.2 Defining the monitor for an \mathcal{ALC} -LTL formula without rigid names	32
3.2.1 The generalised non-deterministic Büchi automaton for an \mathcal{ALC} -LTL formula without rigid names	32
3.2.2 The monitor construction for \mathcal{ALC} -LTL without rigid names	38
3.3 The complexity for the monitor construction for \mathcal{ALC} -LTL without rigid names	43
4 Runtime verification for \mathcal{ALC}-LTL with respect to rigid names	46
4.1 Defining the monitor for an \mathcal{ALC} -LTL formula with rigid names	46
4.1.1 The generalised non-deterministic Büchi automaton for an \mathcal{ALC} -LTL formula with rigid names	47

4.1.2	The monitor construction for \mathcal{ALC} -LTL without rigid names . . .	51
4.2	The complexity for the monitor construction for \mathcal{ALC} -LTL with rigid names	53
5	Runtime verification for \mathcal{ALC}-LTL with rigid names w. r. t. incomplete knowledge	55
5.1	Defining the monitor for an \mathcal{ALC} -LTL formula with rigid names that respects incomplete knowledge	55
5.2	The complexity for the monitor construction for \mathcal{ALC} -LTL with rigid names with respect to incomplete knowledge	62
5.3	An example of the monitor construction	63
5.3.1	The \mathcal{ALC} -LTL formula for the specification	64
5.3.2	The generalised non-deterministic Büchi automata for the \mathcal{ALC} -LTL formulas φ and $\neg\varphi$	65
5.3.3	The monitor for the \mathcal{ALC} -LTL formula φ	69
6	Conclusions and further work	73
6.1	Conclusions of this work	73
6.2	Possible further work	74

1 Introduction and motivation

Daily life in the 21st century depends on computers, i. e. complex hardware and software systems to be more precise. The question whether such a system works as it should, i. e. whether such a system satisfies its specification, is becoming more and more crucial. This gets even more important if one deals with safety-critical systems, such as aviation systems or software systems for power plants. If the specification is not satisfied, one has to cope with negative consequences, such as accidents or disasters. Almost the same applies to commercially used systems where faults in the design and implementation of a system can yield heavy financial losses.

One can examine such complex systems by means of testing [13] and simulation. Even though these techniques are very important, they do not suffice, for they only reveal faults in the implementation but fail to prove that the system works correctly. In the context of such important systems, one is interested in *verifying* that the system satisfies its specification.

There are several ways for verifying complex systems. One method is model checking [6, 3], i. e. a full automatic model-based verification technique. This method can be applied only if one can formalise the relevant system behaviour appropriately. This includes that one needs to know how the system will perform in future. Another approach is called runtime verification [7], which is the technique we will consider here.

1.1 The main idea of runtime verification

Runtime verification works as follows. One formalises the specification for a system that one would verify—or at least a specific property that could be used for verification. This is usually done using a temporal logic, such as LTL [14]. Chapter 2 contains a subsection that deals with temporal logics and introduces LTL in more detail; see Subsection 2.2.2.

Given this formalisation of the specification, one constructs a so-called monitor out of the formalisation of the specification. This automatically generated monitor and the system that is to be verified are executed in parallel. Now, the monitor alarms the user or system designer if the specification is violated.

Obviously, the monitor does not depend on the system but only on the specification. The input of the monitor is the observed behaviour of the system. Hence, runtime verification is a very powerful tool for monitoring black-box systems and grey-box systems. For more information about black-box systems and grey-box systems see [8, 13].

Runtime verification can be used in the testing phase of a system [5] in order to check the behaviour of the system executed with typical inputs. Monitors cannot only be used during elaboration of a system, but also thereafter—whilst the system is running productively. Thus, one can find faults in the system implementation whilst the system is working. If such faults cannot be corrected when the system is running, a system shut-down could be forced in order to prevent accidents or other severe consequences.

1.2 Related work and contribution of this work

Various approaches for runtime verification for LTL have already been proposed. In [4] there is introduced a three-valued output of the monitor. If the monitor outputs 1, then the specification is satisfied in all possible future points of time and vice versa. The truth value 0 is output if and only if the specification is violated in all possible futures. The output ?—inconclusive—is generated if and only if there are futures that meet the specification and there are also futures that violate the specification, i. e. it is not known how the system will behave. We adapt this approach for the temporal description logic *ACC-LTL* [2] in this work.

It makes sense to consider runtime verification using temporal description logics, since there are applications in which the specification of the system not only has a temporal aspect but also relates to description logics. This holds true for some ontology-based applications, e. g. in the medical field. It can be necessary to monitor the patient's vital functions. Using concepts which are defined in some medical ontology enables medical staff to obtain a high-level view of the patient's medical status. Runtime verification can be used to check whether an intervention by a doctor or an adaptation of the patient's medication is necessary.

Sadly, it is often the case that the observation of the system behaviour is not detailed enough. If the observation yields incomplete knowledge over the internal state of the system at some points of time, then we reason over incomplete knowledge. In this work also this case is considered. A monitor for \mathcal{ALC} -LTL w.r.t. incomplete knowledge is defined and its correctness is proved.

1.3 Overview

This work is organised as follows. Chapter 2 introduces some basic notions and definitions. We start introducing some automata models that are used for the monitor construction later on. Then basic notions of description logics and temporal logics are given in order to introduce the combination of both—namely, temporal description logics. The chapter concludes with introducing two verification techniques for LTL: runtime verification and explicit-state model checking. Chapter 2 equips the reader with necessary knowledge in order to understand the contents of later chapters.

The next chapter, Chapter 3, deals with the easiest case, namely, runtime verification for \mathcal{ALC} -LTL without rigid names. First, the motivation for the notion of rigid names is given. This is followed by the monitor construction. We conclude with an analysis of the complexity.

Chapter 4 contains a further development of the approach in Chapter 3. It is shown how to treat rigid names. This chapter also concludes with an analysis of the complexity.

In Chapter 5, we go the extra mile and consider runtime verification with respect to rigid names and incomplete knowledge. After an analysis of the complexity, we will give an example of the monitor construction for an \mathcal{ALC} -LTL formula with rigid names w.r.t. incomplete knowledge.

Chapter 6 deals with conclusions and possible further work.

2 Preliminaries

In order to set a basis, this chapter presents some preliminaries and basic definitions that are used later on.

We begin by considering automata models which we will need later on for the construction of a monitor for specific temporal description logics. We will take a look at finite-state machines with output and generalised non-deterministic Büchi automata. Both models are needed for monitoring systems with respect to a specification given in terms of a formula of a temporal description logic.

After that, we will introduce description logics in general and temporal description logics in particular and we will define basic notions of the logical formalism that we use for runtime verification. Among these, we consider the description logic \mathcal{ALC} and the temporal logic LTL . Then we will take a look at the combination of both—namely the temporal description logic $\mathcal{ALC-LTL}$.

Finally, two techniques for formal verification will be introduced. We will consider model checking for LTL and runtime verification for LTL . This will set a basis to understand the monitor construction for specific temporal description logics.

2.1 Automata models

In this section, we are going to consider automata models that are needed later on for the construction of a monitor for an $\mathcal{ALC-LTL}$ formula.

We start by introducing finite-state machines with output. After this, we will consider two types of non-deterministic Büchi automata and we will show that they are equivalent, i. e. both types of automata can be transferred into each other.

2.1.1 Finite-state machines with output

Finite-state machines with output are also known as Moore machines. These automata work on finite words (over an input alphabet) and output a symbol of the output alphabet at every state.

Definition 2.1. A finite-state machine with output (or finite-state machine for short) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, \Gamma, \lambda)$ where:

- Q is the set of states.
- Σ is the input alphabet.
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function.
- $q_0 \in Q$ is the initial state.
- Γ denotes the finite output alphabet.
- $\lambda : Q \rightarrow \Gamma$ denotes the output function.

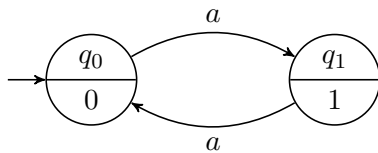
We denote the language that is accepted by \mathcal{A} by $L(\mathcal{A})$.

Finite-state machines work as indicated in the following example.

Example 2.2. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, \Gamma, \lambda)$ be a finite-state machine with:

- $Q := \{q_0, q_1\}$
- $\Sigma := \{a\}$
- $\Gamma := \{0, 1\}$
- $\delta : Q \times \Sigma \rightarrow Q : (q_0, a) \mapsto q_1, (q_1, a) \mapsto q_0$
- $\lambda : Q \rightarrow \Gamma : q_i \mapsto i$ for $i \in \Gamma$

For visualisation, we display the transition function as a digraph:



This finite-state machine outputs at every state whether the just consumed symbol of the input word has an even or an odd index.

The transition function of a finite-state machine works on symbols of an input alphabet. We will now generalise this function to a function working on words.

Definition 2.3. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, \Gamma, \lambda)$ be a finite-state machine. We denote by $\hat{\delta}$ the following function. $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ is defined inductively:

- $\hat{\delta}(q, \varepsilon) := q$
- $\hat{\delta}(q, \sigma w) := \hat{\delta}(\delta(q, \sigma), w)$ for $\sigma \in \Sigma$ and $w \in \Sigma^*$

2.1.2 Non-deterministic Büchi automata

For the construction of the monitor later on we also need an automata model that works on infinite words. We introduce Büchi automata here.

Definition 2.4. A non-deterministic Büchi automaton \mathcal{N} is defined as a tuple $\mathcal{N} = (Q, \Sigma, \Delta, Q_0, F)$ with:

- a set of states Q ,
- an 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 $F \subseteq Q$.

\mathcal{N} accepts a word $w = \sigma_0 \sigma_1 \dots \in \Sigma^\omega$ iff there is a path q_0, q_1, \dots with $q_0 \in Q_0$ and $(q_i, \sigma_i, q_{i+1}) \in \Delta$ for all $i \in \mathbb{N}$ and the path contains infinitely many final states.

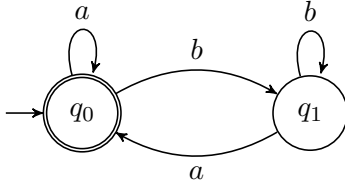
We denote the language that is accepted by \mathcal{N} by $L(\mathcal{N})$.

The next example illustrates this formalism.

Example 2.5. Let $\mathcal{N} = (Q, \Sigma, \Delta, Q_0, F)$ be a non-deterministic Büchi automaton with:

- $Q := \{q_0, q_1\}$
- $\Sigma := \{a, b\}$
- $\Delta := \{(q_0, a, q_0), (q_0, b, q_1), (q_1, a, q_0), (q_1, b, q_1)\}$
- $Q_0 := \{q_0\}$
- $F := \{q_0\}$

As usual, we display the transition relation as a digraph:



This automaton accepts all words $w \in \Sigma^\omega$ that infinitely often contain the symbol a .

The next definition generalises the notion of a non-deterministic Büchi automaton.

Definition 2.6. A generalised non-deterministic Büchi automaton \mathcal{G} is a tuple $\mathcal{G} = (Q, \Sigma, \Delta, Q_0, \mathcal{F})$ where every component is defined as in the case of non-deterministic Büchi automata except for \mathcal{F} that is defined as a set of sets of final states, i. e. $\mathcal{F} \subseteq 2^Q$ and thus $\mathcal{F} = \{F_0, F_1, \dots, F_k\}$ for some $k \in \mathbb{N}$.

The acceptance condition for \mathcal{G} is the following. The automaton accepts a word $w \in \Sigma^\omega$ iff there is a path q_0, q_1, \dots with $q_0 \in Q_0$ and $(q_i, \sigma_i, q_{i+1}) \in \Delta$ for all $i \in \mathbb{N}$ and the path passes through at least one state of each set of final states F_i for $0 \leq i \leq k$ infinitely often.

We denote the language that is accepted by \mathcal{G} by $L(\mathcal{G})$.

The next definition introduces a notation that is used to change the set of initial states of (generalised) non-deterministic Büchi automata.

Definition 2.7. For the generalised non-deterministic Büchi automaton \mathcal{G} with $\mathcal{G} = (Q, \Sigma, \Delta, Q_0, \mathcal{F})$ we denote by $\mathcal{G}(q)$ with $q \in Q$ the generalised non-deterministic Büchi automaton that is defined exactly as \mathcal{G} except for Q_0 that is defined as $Q_0 := \{q\}$.

For the non-deterministic Büchi automaton $\mathcal{N} = (Q', \Sigma', \Delta', Q'_0, F)$ the notion $\mathcal{N}(q)$ is defined analogously.

Formally, that means:

$$\begin{aligned} \mathcal{G}(q) &= (Q, \Sigma, \Delta, \{q\}, \mathcal{F}) \\ \mathcal{N}(q) &= (Q', \Sigma', \Delta', \{q\}, F) \end{aligned}$$

Again, we can generalise the transition relation to a relation working on words.

Definition 2.8. Let $\Delta \subseteq Q \times \Sigma \times Q$ be the transition relation of a (generalised) non-deterministic Büchi automaton. We define $\hat{\Delta} \subseteq Q \times \Sigma^* \times Q$ to be the following:

- $(q, \varepsilon, q) \in \hat{\Delta}$ for all $q \in Q$
- $(q, \sigma w, q') \in \hat{\Delta}$ iff there is some $q'' \in Q$ with $(q, \sigma, q'') \in \Delta$ and $(q'', w, q') \in \hat{\Delta}$ for $\sigma \in \Sigma$ and $w \in \Sigma^*$

It is well-known that for every LTL-formula φ , one can construct a corresponding generalised non-deterministic Büchi automaton $\mathcal{G} = (Q, \Sigma, \Delta, Q_0, \mathcal{F})$ with the following properties. The alphabet of the automaton is defined as follows: $\Sigma := 2^{\mathcal{P}}$ where \mathcal{P} denotes the set of propositional variables occurring in φ . The generalised non-deterministic Büchi automaton is constructed in such a way that for every word $\sigma_0\sigma_1 \dots \in \Sigma^\omega$, we have the following:

$$\sigma_0\sigma_1 \dots \models \varphi \text{ iff } \sigma_0\sigma_1 \dots \in L(\mathcal{G})$$

The construction [17, 16] is adapted for \mathcal{ALC} -LTL later on.

The next theorem shows the connection between generalised non-deterministic Büchi automata and non-deterministic Büchi automata. It turns out that the generalised version of these automata is not mightier than the standard version. Both types of Büchi automata accept the same class of languages.

Theorem 2.9. Let $\mathcal{G} = (Q, \Sigma, \Delta, Q_0, \mathcal{F})$ be a generalised non-deterministic Büchi automaton. In this case, there exists a non-deterministic Büchi automaton \mathcal{N} with $\mathcal{N} = (Q', \Sigma, \Delta', Q'_0, F)$ with $L(\mathcal{G}) = L(\mathcal{N})$.

Proof. We assume without loss of generality that $\mathcal{F} = \{F_1, \dots, F_k\}$ for $k \geq 1$. This is without loss of generality since adding Q to \mathcal{F} does not affect the accepted language of \mathcal{G} .

The non-deterministic Büchi automaton $\mathcal{N} = (Q', \Sigma, \Delta', Q'_0, F)$ is defined as follows.

- $Q' := Q \times \{1, \dots, k\}$
- $Q'_0 := \{(q_0, 1) \in Q' \mid q_0 \in Q_0\}$
- $F := \{(q, 1) \in Q' \mid q \in F_1\}$
- The transition relation Δ' has the following properties:

- For all $1 \leq i \leq k$, we have that for all $q \in Q \setminus F_i$ and $\sigma \in \Sigma$ we have:

$$(q, \sigma, q') \in \Delta \text{ iff } ((q, i), \sigma, (q', i)) \in \Delta'$$

- For all $1 \leq i < k$, we have that for all $q \in F_i$ and $\sigma \in \Sigma$ we have:

$$(q, \sigma, q') \in \Delta \text{ iff } ((q, i), \sigma, (q', i + 1)) \in \Delta'$$

- For all $q \in F_k$ and $\sigma \in \Sigma$, we have:

$$(q, \sigma, q') \in \Delta \text{ iff } ((q, k), \sigma, (q', 1)) \in \Delta'$$

It remains to show that $L(\mathcal{G}) = L(\mathcal{N})$.

“ \supseteq ”: \mathcal{N} consists of k copies of \mathcal{G} that are linked via the elements of F_i for $1 \leq i \leq k$. Every accepting path of \mathcal{N} contains by construction infinitely many states (q, i) for $q \in F_i$ for all $1 \leq i \leq k$. But this is also the acceptance condition of \mathcal{G} .

“ \subseteq ”: Every accepting path of \mathcal{G} contains by construction infinitely many states $q \in F_i$ for all $1 \leq i \leq k$. Thus, there exists a path in \mathcal{N} with the property that every copy of \mathcal{G} is visited infinitely often. This path is obviously accepting.

□

2.2 Temporal description logics

Temporal description logics combine description logics and temporal logics. Starting from the description logic \mathcal{ALC} , which was introduced in [15], and the temporal logic LTL [14], we will introduce the temporal description logic \mathcal{ALC} -LTL as it is done in [2].

2.2.1 Description logics

Description logics are a whole family of knowledge representation formalisms. All the basic notions regarding description logics can be found in [1]. The main advantage

of description logics over early knowledge representation formalisms is the fact that description logics have a formal semantics.

We will consider a very prominent description logic here—namely \mathcal{ALC} . The name of this description logic is an acronym for “attributive language with complement”. We will now define the syntax and the semantics of this basic description logic.

Definition 2.10. *Let N_C , N_R and N_I be three disjoint sets of concept names, role names and individual names, respectively.*

The set of \mathcal{ALC} -concept descriptions—or \mathcal{ALC} -formulas—is the smallest set that satisfies all of the following properties.

- All concept names $A \in N_C$ are \mathcal{ALC} -concept descriptions.
- \top (top concept) and \perp (bottom concept) are \mathcal{ALC} -concept descriptions.
- If C and D are \mathcal{ALC} -concept descriptions and $r \in N_R$, then we can use the following concept constructors in order to obtain further \mathcal{ALC} -concept descriptions.
 - $\neg C$ (negation)
 - $C \sqcap D$ (conjunction)
 - $C \sqcup D$ (disjunction)
 - $\exists r.C$ (existential restriction)
 - $\forall r.C$ (value restriction)

Note: We do not require the above defined sets to be finite since their cardinality is limited by specific utilisations.

Having defined the syntax of \mathcal{ALC} , it remains to define the semantics of \mathcal{ALC} -concept descriptions. This is usually done by means of the notion of an interpretation. The next definition introduces this notion.

Definition 2.11. *An \mathcal{ALC} -interpretation is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ that consists of a non-empty domain $\Delta^{\mathcal{I}}$ and a mapping $\cdot^{\mathcal{I}}$ with the following properties.*

- For every $A \in N_C$ the mapping \mathcal{I} assigns a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$.
- For every $r \in N_R$ the mapping \mathcal{I} assigns a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.
- For every $a \in N_I$ the mapping \mathcal{I} assigns an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

The mapping \mathcal{I} is extended to \mathcal{ALC} -concept descriptions as follows.

- $\top^{\mathcal{I}} := \Delta^{\mathcal{I}}$ and $\perp^{\mathcal{I}} := \emptyset$
- $(\neg C)^{\mathcal{I}} := \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
- $(C \sqcap D)^{\mathcal{I}} := C^{\mathcal{I}} \cap D^{\mathcal{I}}$
- $(C \sqcup D)^{\mathcal{I}} := C^{\mathcal{I}} \cup D^{\mathcal{I}}$
- $(\exists r.C)^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid \text{there exists some } e \in \Delta^{\mathcal{I}} \text{ with } (d, e) \in r^{\mathcal{I}} \text{ and } e \in C^{\mathcal{I}}\}$
- $(\forall r.C)^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid \text{for all } e \in \Delta^{\mathcal{I}} \text{ we have that } (d, e) \in r^{\mathcal{I}} \text{ implies } e \in C^{\mathcal{I}}\}$

We say that an \mathcal{ALC} -interpretation \mathcal{I} satisfies an \mathcal{ALC} -formula φ (written: $\mathcal{I} \models \varphi$) iff $\varphi^{\mathcal{I}} \neq \emptyset$. In this case we call \mathcal{I} a model of φ .

Let us now look at an example of an \mathcal{ALC} -formula.

Example 2.12. *Let us assume that we want to define the following concept description in \mathcal{ALC} : the men that have at least one child and all of them are daughters. We formalise this as follows.*

$$\varphi = \text{Male} \sqcap \exists \text{child} . \top \sqcap \forall \text{child} . \text{Female}$$

Here, *Male* and *Female* are concept names, i. e. $\{\text{Male}, \text{Female}\} \subseteq N_C$ and *child* is a role name, i. e. $\text{child} \in N_R$. We will consider the following interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$:

- $\Delta^{\mathcal{I}} := \{d \mid d \text{ is a human being}\}$
- $\text{Male}^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid d \text{ is male}\}$
- $\text{Female}^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid d \text{ is female}\}$
- $\text{child}^{\mathcal{I}} := \{(d, e) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid e \text{ is a child of } d\}$

Obviously, $\varphi^{\mathcal{I}}$ contains exactly those human beings that are male and have at least one child and all of them are daughters.

Note: Please note that the semantics is entirely given by the interpretation. Names do not imply semantics; we thus do not have for all interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ that $\text{Male}^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus \text{Female}^{\mathcal{I}}$.

Often, one wants to constrain the interpretations so that only those interpretations that have further properties can satisfy an \mathcal{ALC} -formula. Usually, one introduces knowledge bases that are, informally speaking, collections of knowledge over the domain. Now, one can define satisfiability of an \mathcal{ALC} -formula as satisfiability with respect to a knowledge base. The next definitions will follow this approach.

Definition 2.13. A general concept inclusion axiom is of the form $C \sqsubseteq D$ where C and D are \mathcal{ALC} -concept descriptions. An \mathcal{ALC} -interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies a general concept inclusion axiom iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

A TBox (terminological box) \mathcal{T} is a finite set of general concept inclusion axioms. We call \mathcal{I} a model of \mathcal{T} iff it satisfies all general concept inclusion axioms in \mathcal{T} . We write $\mathcal{I} \models \mathcal{T}$.

Let φ be an \mathcal{ALC} -formula and \mathcal{T} a TBox. An \mathcal{ALC} -interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies φ w. r. t. \mathcal{T} (written: $\mathcal{I} \models_{\mathcal{T}} \varphi$) iff $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \varphi$.

We call a TBox \mathcal{T} consistent iff it has a model.

Now, we have a formalism for defining terminological knowledge of the domain. The next example shows how the TBox formalism can be used.

Example 2.14. We will continue with the previous example. We have defined an \mathcal{ALC} -formula φ as follows.

$$\varphi = \text{Male} \sqcap \exists \text{child} . \top \sqcap \forall \text{child} . \text{Female}$$

A useful TBox could be the following:

$$\mathcal{T} := \{ \text{Male} \sqcap \text{Female} \sqsubseteq \perp \}$$

This TBox ensures that for all models $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ the sets $\text{Male}^{\mathcal{I}}$ and $\text{Female}^{\mathcal{I}}$ are disjoint. Of course, one could formalise a lot more terminological knowledge of the domain in order to constrain the models of φ w. r. t. \mathcal{T} even more.

Terminological knowledge does not say anything about the individual names. For this purpose we will define the notion of assertional knowledge.

Definition 2.15. Let $a, b \in N_I$ and let C be an \mathcal{ALC} -concept description and let $r \in N_R$.

We call $a : C$ a concept assertion. An \mathcal{ALC} -interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies a concept assertion $a : C$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$.

We denote by $(a, b) : r$ a role assertion. The \mathcal{ALC} -interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies the role assertion $(a, b) : r$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$.

An ABox (assertional box) \mathcal{A} is a finite set of concept assertions and role assertions. We call \mathcal{I} a model of \mathcal{A} iff it satisfies all assertions in \mathcal{A} . We write $\mathcal{I} \models \mathcal{A}$.

We call an ABox \mathcal{A} consistent iff it has a model.

Note: It is not expedient to define satisfiability of an \mathcal{ALC} -formula with respect to an ABox, for consistent ABoxes have no effect on the satisfiability of concept descriptions.

We will now look at an example that shows a possible application of ABoxes.

Example 2.16. *If we stay with the example involving human beings, then the concept assertion $greg : Male$ and the role assertion $(greg, peggy) : child$ would be appropriate assertions where $\{greg, peggy\} \subseteq N_{\mathcal{I}}$.*

Let

$$\mathcal{A} := \{greg : Male, (greg, peggy) : child, peggy : Female\}$$

and let

$$\varphi = Male \sqcap \exists child.Female$$

Then we have for all models $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of φ w. r. t. \mathcal{A} that $greg \in \varphi^{\mathcal{I}}$.

When dealing with ABoxes we make the open world assumption, i. e. knowledge that is not mentioned is assumed to be unknown rather than negative. If we consider the ABox \mathcal{A} of Example 2.16, then we see that it does not say anything about the individual *paul*. Proceeding on the open world assumption, it is not known whether $paul : Male$ holds.

The opposite of the open world assumption is the closed world assumption. Proceeding on this assumption, not mentioned knowledge is assumed to be false. In our example, *paul* is neither in the extension of *Male* nor in the extension of *Female*.

It makes sense to proceed on the open world assumption, since no observer has necessarily complete knowledge over the domain. Thus, an ABox represents incomplete knowledge, whereas an interpretation over the domain represents complete knowledge.

Often, one wants to formalise both assertional and terminological knowledge. Therefore, we define the notion of knowledge base—i. e. informally speaking, a collection of assertional and terminological knowledge—formally.

Definition 2.17. *A knowledge base \mathcal{K} is of the form $\mathcal{K} := (\mathcal{T}, \mathcal{A})$ where \mathcal{T} is a TBox and \mathcal{A} is an ABox.*

An \mathcal{ALC} -interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a model of $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ (written: $\mathcal{I} \models \mathcal{K}$) iff it is a model of \mathcal{T} and \mathcal{A} .

Let φ be an \mathcal{ALC} -formula. An \mathcal{ALC} -interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies φ w. r. t. \mathcal{K} (written: $\mathcal{I} \models_{\mathcal{K}} \varphi$) iff $\mathcal{I} \models \varphi$ and $\mathcal{I} \models \mathcal{K}$.

We call a knowledge base \mathcal{K} consistent iff it has at least one model.

Knowledge bases are a very powerful tool for describing domain knowledge. Later on, we will need the notion of a Boolean knowledge base, which leads us to the next definition.

Definition 2.18. A Boolean knowledge base is a Boolean combination of description logic axioms, i. e. general concept inclusion axioms, concept assertions and role assertions.

- Every description logic axiom is a Boolean knowledge base.
- If \mathcal{B}_1 and \mathcal{B}_2 are Boolean knowledge bases, then so are $\neg\mathcal{B}_1$, $\mathcal{B}_1 \wedge \mathcal{B}_2$ and $\mathcal{B}_1 \vee \mathcal{B}_2$.

The notion of an \mathcal{ALC} -interpretation is extended to Boolean knowledge bases as follows. For an \mathcal{ALC} -interpretation \mathcal{I} we have:

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

If \mathcal{I} is a model of \mathcal{B} , we write $\mathcal{I} \models \mathcal{B}$.

We call the Boolean knowledge base \mathcal{B} consistent iff it has at least one model.

2.2.2 Temporal logics

A lot of different temporal logics were presented in the past. For introductory reading, see for example [9]. There were also proposed a lot of extensions to these logics—e. g. probabilistic and real-time extensions.

In this subsection, we are going to introduce linear-time temporal logic [14]—or LTL for short. This logic is a very prominent temporal logic that is often used in formal verification [3, 6]. The next definition introduces the syntax of LTL.

Definition 2.19. Let \mathcal{P} denote the set of propositional variables. The set of LTL-formulas is defined as follows.

- Every propositional variable $p \in \mathcal{P}$ is an LTL-formula.
- If φ and ψ are LTL-formulas, then so are:
 - $\neg\varphi$ (negation)
 - $\varphi \wedge \psi$ (conjunction)
 - $\bigcirc\varphi$ (next)
 - $\varphi \mathcal{U}\psi$ (until)

There are several possibilities to define the semantics of an LTL-formula. We define the semantics using infinite words over the alphabet $2^{\mathcal{P}}$.

Definition 2.20. Let $w = \sigma_0\sigma_1\sigma_2\dots$ be an infinite word over $2^{\mathcal{P}}$ where \mathcal{P} denotes the set of propositional variables, i. e. $w \in (2^{\mathcal{P}})^\omega$. Now we will define inductively what it means that w models an LTL-formula φ (written: $w \models \varphi$). Let $p \in \mathcal{P}$ and let φ and ψ , respectively, be LTL-formulas.

$$\begin{aligned}
 w, i \models p & \quad \text{iff } p \in \sigma_i \\
 w, i \models \neg\varphi & \quad \text{iff } w, i \not\models \varphi \\
 w, i \models \varphi \wedge \psi & \quad \text{iff } w, i \models \varphi \text{ and } w, i \models \psi \\
 w, i \models \bigcirc\varphi & \quad \text{iff } w, i + 1 \models \varphi \\
 w, i \models \varphi \mathcal{U}\psi & \quad \text{iff } \text{there exists some } k \geq i \text{ with } w, k \models \psi \\
 & \quad \text{and we have for all } j \in \{i, i + 1, \dots, k - 1\} : w, j \models \varphi
 \end{aligned}$$

The LTL-formula φ is satisfiable iff there exists some $w \in (2^{\mathcal{P}})^\omega$ with $w, 0 \models \varphi$.

Before giving an example, we will introduce some useful abbreviations.

Definition 2.21. Let φ and ψ , respectively, be LTL-formulas. We define the following abbreviations:

- $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$ (disjunction)
- $\varphi \rightarrow \psi := \neg\varphi \vee \psi$ (implication)
- $\text{true} := p \vee \neg p$ for some $p \in \mathcal{P}$
- $\text{false} := \neg\text{true}$
- $\diamond\varphi := \text{true} \mathcal{U}\varphi$ (eventually/finally)

- $\Box\varphi := \neg\Diamond\neg\varphi$ *(always/globally)*

Let us discuss the intuitive meaning of the \Diamond -operator and the \Box -operator. \Diamond is read as “eventually” whereas \Box is read as “always”. Obviously, $\Diamond\varphi$ means that the formula φ must be true at some point in the future. $\Box\varphi$ means that the formula φ has to be true at all times. So the readings for these temporal modalities are appropriate.

Let us give an example. Since LTL is often used in the context of formal verification, this example has to do with parallel processes working with a critical section.

Example 2.22. *Let us assume that we have two processes dealing with a critical section. The propositional variable $crit_1$ denotes that process 1 is in the critical section. The propositional variable $crit_2$ is defined analogously.*

The safety property “there are never both processes in the critical section” could be formalised as:

$$\varphi_1 := \Box\neg(crit_1 \wedge crit_2)$$

The liveness property “every process reaches the critical section infinitely often” could be expressed as:

$$\varphi_2 := \Box\Diamond crit_1 \wedge \Box\Diamond crit_2$$

Let $\varphi := \varphi_1 \wedge \varphi_2$. If we consider the system and note at each discrete point of time under consideration which processes are in the critical section—where \emptyset means that no process is in the critical section, $crit_1$ means that process 1 is in the critical section and so forth—then we get a finite prefix of a word $w \in (2^{\{crit_1, crit_2\}})^\omega$.

Now, $w \models \varphi$ iff during the specific run of the system, represented by w , it never occurs that both processes are in the critical section simultaneously and every process reaches the critical section infinitely often. Obviously, this is no universal prediction; there could be also a run of the system, represented by a word $w' \in (2^{\{crit_1, crit_2\}})^\omega$, which does not satisfy the demanded properties.

Section 2.3 deals with two techniques for formal verification of linear-time properties, namely explicit model checking for LTL and runtime verification for LTL.

2.2.3 The combination: temporal description logics

There are several approaches for combining description logics with temporal logics. A survey can be found in [11]. One has to decide which temporal logic and which description logic one wants to use and whether all temporal modalities should be available. Here we introduce LTL over description logic axioms. The combination of LTL and \mathcal{ALC} is called \mathcal{ALC} -LTL [2].

We begin by defining the syntax and the semantics of this temporal description logic.

Definition 2.23. *The syntax of \mathcal{ALC} -LTL is defined inductively:*

- Every description logic axiom is an \mathcal{ALC} -LTL formula.
- If φ and ψ are \mathcal{ALC} -LTL formulas, then so are:
 - $\neg\varphi$ (negation)
 - $\varphi \wedge \psi$ (conjunction)
 - $\bigcirc\varphi$ (next)
 - $\varphi \mathcal{U}\psi$ (until)

Obviously, this definition is very similar to the definition of the syntax of LTL. We do not mix temporal modalities with description logic concept constructors. Temporal operators are only allowed in front of description logic axioms. The semantics is defined in terms of \mathcal{ALC} -LTL interpretations, i. e. sequences of \mathcal{ALC} -interpretations. Formally, this is defined as follows.

Definition 2.24. *An \mathcal{ALC} -LTL interpretation \mathfrak{I} is a sequence of \mathcal{ALC} -interpretations $\mathcal{I}_i = (\Delta^{\mathcal{I}_i}, \mathcal{I}_i)$ that respect the so-called rigid-name assumption, i. e. for all individual names $a \in N_I$ and all $i, j \in \{0, 1, 2, \dots\}$, we have that $a^{\mathcal{I}_i} = a^{\mathcal{I}_j}$. Additionally, for all \mathcal{I}_i , we proceed on the unique-name assumption, i. e. for all individual names $a \in N_I$ we have that $a^{\mathcal{I}_i} \neq b^{\mathcal{I}_i}$ for $a \neq b$.*

Given an \mathcal{ALC} -LTL interpretation $\mathfrak{I} = (\mathcal{I}_i)_{i \geq 0}$ and a point of time $i \in \mathbb{N}$, satisfiability of an \mathcal{ALC} -LTL formula φ at the point of time $i \in \mathbb{N}$ (written: $\mathfrak{I}, i \models \varphi$) is defined inductively as follows:

$\mathfrak{I}, i \models C \sqsubseteq D$	<i>iff</i>	$C^{\mathcal{I}_i} \subseteq D^{\mathcal{I}_i}$
$\mathfrak{I}, i \models a : C$	<i>iff</i>	$a^{\mathcal{I}_i} \in C^{\mathcal{I}_i}$
$\mathfrak{I}, i \models (a, b) : r$	<i>iff</i>	$(a^{\mathcal{I}_i}, b^{\mathcal{I}_i}) \in r^{\mathcal{I}_i}$
$\mathfrak{I}, i \models \neg\varphi$	<i>iff</i>	$\mathfrak{I}, i \not\models \varphi$
$\mathfrak{I}, i \models \varphi \wedge \psi$	<i>iff</i>	$\mathfrak{I}, i \models \varphi$ and $\mathfrak{I}, i \models \psi$
$\mathfrak{I}, i \models \bigcirc\varphi$	<i>iff</i>	$\mathfrak{I}, i + 1 \models \varphi$
$\mathfrak{I}, i \models \varphi \mathcal{U}\psi$	<i>iff</i>	there is some $k \geq i$ with $\mathfrak{I}, k \models \psi$ and we have for all $j \in \{i, i + 1, \dots, k - 1\} : \mathfrak{I}, j \models \varphi$

The \mathcal{ALC} -LTL formula φ is satisfiable iff there exists some \mathcal{ALC} -LTL interpretation \mathfrak{I} with $\mathfrak{I}, 0 \models \varphi$

Again, we want to introduce some useful abbreviations.

Definition 2.25. Let φ and ψ be \mathcal{ALC} -LTL formulas. We define the following abbreviations.

- $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$ *(disjunction)*
- $\varphi \rightarrow \psi := \neg\varphi \vee \psi$ *(implication)*
- $true := \top \sqsubseteq \top$
(alternative definition: $true := a : A \vee \neg(a : A)$ for some $a \in N_I$ and some $A \in N_C$)
- $false := \neg true$
- $\diamond\varphi := true \mathcal{U}\varphi$ *(eventually/finally)*
- $\square\varphi := \neg\diamond\neg\varphi$ *(always/globally)*

Often it is desired that the interpretation of a concept or role name does not change over time. Thus, we will assume that there is a subset of N_C and a subset of N_R whose elements are designated as rigid. We call the other concept names and role names, which do not belong to this subset, flexible.

Definition 2.26. Let \mathfrak{I} be an \mathcal{ALC} -LTL interpretation. We say that \mathfrak{I} respects rigid names iff we have for all rigid concept names A that $A^{\mathcal{I}_i} = A^{\mathcal{I}_j}$ for all $i, j \in \mathbb{N}$ and we have for all rigid role names r that $r^{\mathcal{I}_i} = r^{\mathcal{I}_j}$ for all $i, j \in \mathbb{N}$.

Let us now look at an example for an \mathcal{ALC} -LTL formula. This example also shows the limitations of \mathcal{ALC} -LTL.

Example 2.27. *Let us assume that we want to express the fact that every offender eventually gets punishment by a judge. Let $N_C = \{Human, Offence, Judge, \dots\}$ and let $N_R = \{commit, get_punishment_by, \dots\}$.*

Since in \mathcal{ALC} -LTL we allow temporal modalities only in front of description logic axioms, the favoured fact cannot be expressed as an \mathcal{ALC} -LTL formula. The following attempt does not work, i. e. it does not formalise exactly what is desired:

$$\Box \left(\bigwedge_{a \in N_I} (a : Human \sqcap \exists commit. Offence) \rightarrow \Diamond (a : \exists get_punishment_by. Judge) \right)$$

Why does this attempt not work? We treat only named individuals! Additionally, we proceed on the assumption that N_I is finite, for \mathcal{ALC} -LTL formulas must be finite. The benefit of the rigid-name assumption is obvious: individuals must not change their names over time.

Nonetheless, this example shows how a syntactically valid \mathcal{ALC} -LTL formula could look like. We will give further examples later on.

2.3 Formal verification of linear-time properties

In this section, we are going to introduce two methods for formal verification of linear-time properties. We will consider model checking and runtime verification formally. We will realise the similarities and differences of these techniques. Later on, we will adopt runtime verification in order to treat specifications given in terms of a temporal description logic formula.

By linear-time properties we denote properties of systems that can be formalised using a temporal logic that formalises with respect to the linear-time view—in contrast to branching-time view—like linear-time temporal logic (LTL) [14] or a temporal description logic like \mathcal{ALC} -LTL [2].

2.3.1 Explicit-state model checking for LTL

Model checking in general [6, 3] deals with the important question whether a formalised representation—or abstraction—of a system satisfies a specification. We call a formalised representation of the system a model \mathcal{M} . The specification is given in terms of a formula of temporal logic φ . Thus, the main question is whether \mathcal{M} satisfies φ or not, i. e. whether $\mathcal{M} \models \varphi$ or $\mathcal{M} \not\models \varphi$ holds true.

We distinguish between explicit-state model checking and symbolic model checking. Explicit-state model checking is called “explicit” because the state space is constructed explicitly. We will consider explicit-state model checking for LTL here. Symbolic model checking follows a different approach. In this case the state space is described using binary decision diagrams (BDDs) or propositional formulas. The analysis of the system is done also symbolically so that the state space is never constructed explicitly. This is done using BDD tools or SAT solvers. More information on symbolic model checking can be found in [12].

Subsequently, we will discuss explicit-state model checking for LTL in more detail. As argued before, we need an explicit representation of the system under validation. We will formalise the system behaviour using labelled transition systems. The next definition introduces labelled transition systems formally.

Definition 2.28. *A labelled transition system \mathcal{T} is of the form: $\mathcal{T} = (S, A, \rightarrow, S_0, \mathcal{P}, L)$ where*

- S denotes the set of states,
- A denotes the set actions,
- $\rightarrow \subseteq S \times A \times S$ denotes the transition relation,
- $S_0 \subseteq S$ denotes the set of initial states,
- \mathcal{P} denotes the set of propositional variables and
- $L : S \rightarrow 2^{\mathcal{P}}$ denotes the labelling function.

\mathcal{T} is called finite if S , A and \mathcal{P} are finite.

Here, we consider only model checking with respect to finite labelled transition systems.

Intuitively, labelled transition systems formalise non-deterministic systems. It starts in some initial state $s_0 \in S_0$ and evolves according to the transition relation \rightarrow .

Transitions are labelled with actions. These are used for communication mechanisms of systems and for synchronisation of systems. They could also be used to formalise system behaviour that depends on user input.

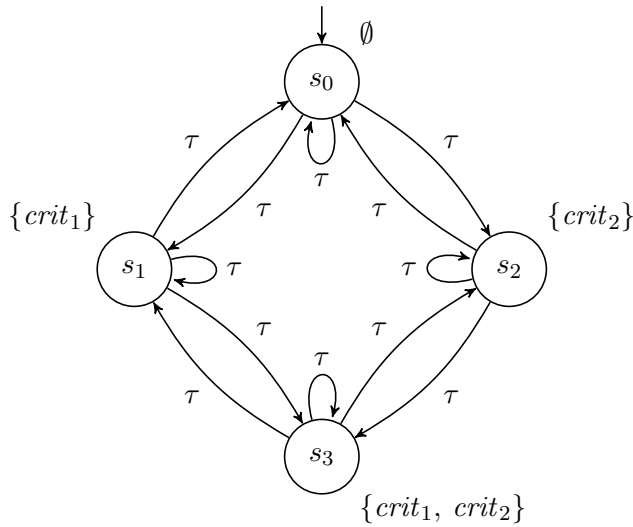
States are labelled by the labelling function L with sets of propositional variables. Intuitively, $L(s) = \{p_0, p_1, \dots, p_k\}$ for $s \in S$ and $p_0, p_1, \dots, p_k \in \mathcal{P}$ for $k \in \mathbb{N}$ means that the propositional variables p_0, p_1, \dots, p_k are true in state s . Formally, L yields a valuation of truth values for all propositional variables in \mathcal{P} for each state in S .

We will now give an example for a labelled transition system.

Example 2.29. Let $\mathcal{T} := (\{s_0, s_1, s_2, s_3\}, \{\tau\}, \rightarrow, \{s_0\}, \{crit_1, crit_2\}, L)$ be a labelled transition system with:

$$\begin{aligned} \rightarrow := & \{ (s_0, \tau, s_0), (s_0, \tau, s_1), (s_0, \tau, s_2), (s_1, \tau, s_0), (s_1, \tau, s_1), (s_1, \tau, s_3), \\ & (s_2, \tau, s_0), (s_2, \tau, s_2), (s_2, \tau, s_3), (s_3, \tau, s_1), (s_3, \tau, s_2), (s_3, \tau, s_3) \} \\ L(s_0) := & \emptyset, L(s_1) := \{crit_1\}, L(s_2) := \{crit_2\}, L(s_3) := \{crit_1, crit_2\} \end{aligned}$$

As usual, we visualise labelled transition systems as a labelled digraph.



This labelled transition system formalises two concurrent processes working with a critical section. The states in S denote the four possible states of the system, as also seen on the labelling function L : there is no process in the critical section (s_0), only process 1 is in the critical section (s_1), only process 2 is in the critical section (s_2) and both processes are in the critical section (s_3). The transition relation \rightarrow shows all the possible state changes the system can perform. We call such state changes “actions” and mark them with a proper action symbol. Since in our case the names of the actions are irrelevant, we use the pseudo action symbol τ .

Having introduced how the system behaviour is formalised appropriately, we will consider how explicit-state model checking is performed. The model-checking algorithm is shown in Algorithm 2.1.

Algorithm 2.1 Explicit-state model checking for LTL

Input: System behaviour formalised as labelled transition system \mathcal{M} and system specification formalised as LTL-formula φ

Output: $\mathcal{M} \models \varphi$ or $\mathcal{M} \not\models \varphi$

- 1: Construct non-deterministic Büchi automaton \mathcal{N} corresponding to $\neg\varphi$.
 - 2: Construct product labelled transition system $\mathcal{M} \times \mathcal{N}$.
 - 3: Check whether there exists some state in $\mathcal{M} \times \mathcal{N}$ that is labelled with a final state of \mathcal{N} and that is located on a cycle.
 - 4: **IF** Cycle check was successful **THEN**
 - 5: **return** $\mathcal{M} \not\models \varphi$
 - 6: **ELSE**
 - 7: **return** $\mathcal{M} \models \varphi$
 - 8: **FI**
-

We will now go into details about Algorithm 2.1. The algorithm starts by constructing the non-deterministic Büchi automaton \mathcal{N} corresponding to $\neg\varphi$. This can be done using the construction of [17, 16]. This way one gets a generalised non-deterministic Büchi automaton corresponding to an LTL-formula. This automaton can be transferred into a non-deterministic Büchi automaton using the construction in the proof of Theorem 2.9. There are also faster algorithms available, e. g. the one implemented in LTL 2 BA [10].

The second step is the construction of the product labelled transition system $\mathcal{M} \times \mathcal{N}$. The next definition shows how this product labelled transition system is actually computed.

Definition 2.30. Let $\mathcal{T} = (S, A, \rightarrow, S_0, \mathcal{P}, L)$ be a labelled transition system and let $\mathcal{N} = (Q, \Sigma, \Delta, Q_0, F)$ be a non-deterministic Büchi automaton. The product labelled transition system $\mathcal{T} \times \mathcal{N}$ is defined as follows: $\mathcal{T} \times \mathcal{N} := (S \times Q, A, \rightarrow', S'_0, Q, L')$ where:

- $L'((s, q)) := \{q\}$ for $s \in S$ and $q \in Q$
- $S'_0 := \{(s_0, q) \mid s_0 \in S_0, (q_0, L(s_0), q) \in \Delta \text{ for } q_0 \in Q_0\}$
- $((s, q), \alpha, (s', q')) \in \rightarrow'$ iff $(s, \alpha, s') \in \rightarrow$ and $(q, L(s'), q') \in \Delta$ for $s, s' \in S, q, q' \in Q$ and $\alpha \in A$

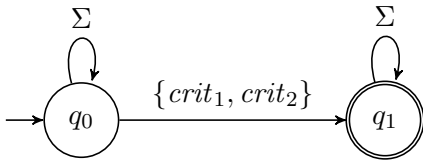
The third step consists of performing a graph analysis on the product labelled transition system $\mathcal{M} \times \mathcal{N}$. First, one searches for states which are labelled with a final state of \mathcal{N} . This can be done using depth-first search. If the depth-first search finds a state that is labelled with a final state of \mathcal{N} , then a second depth-first search checks whether this state is located on a cycle. If this analysis is successful, the search algorithms have found a counterexample, i. e. we have found a path in the labelled transition system that violates the specification.

We will now give an example. We will use the labelled transition system \mathcal{T} of Example 2.29 and the specification φ_1 of Example 2.22.

Example 2.31. First, we need the automaton for $\neg\varphi_1 = \neg\Box\neg(\text{crit}_1 \wedge \text{crit}_2)$. After replacing the abbreviations, we get $\neg\varphi_1 = \text{true}\mathcal{U}(\text{crit}_1 \wedge \text{crit}_2)$. The tool *LTL 2 BA* [10] yields the following non-deterministic Büchi automaton $\mathcal{N} = (Q, \Sigma, \Delta, Q_0, F)$ with:

- $Q := \{q_0, q_1\}$
- $\Sigma := 2^{\{\text{crit}_1, \text{crit}_2\}}$
- $Q_0 := \{q_0\}$
- $F := \{q_1\}$
- The transition relation Δ has only the following elements.
 - $(q_0, \sigma, q_0) \in \Delta$ for all $\sigma \in \Sigma$
 - $(q_0, \{\text{crit}_1, \text{crit}_2\}, q_1) \in \Delta$
 - $(q_1, \sigma, q_1) \in \Delta$ for all $\sigma \in \Sigma$

The visualisation of \mathcal{N} looks as follows.



Note that $\xrightarrow{\Sigma}$ signifies all transitions $\xrightarrow{\sigma}$ with $\sigma \in \Sigma$.

Next, we construct the reachable part of the product labelled transition system $\mathcal{T} \times \mathcal{N}$. The reachable part is called \mathcal{T}' . We have: $\mathcal{T}' := (S', \{\tau\}, \rightarrow', S'_0, Q, L')$ with:

- $S' := \{ (s_0, q_0), (s_1, q_0), (s_2, q_0), (s_3, q_0), (s_3, q_1) \}$
- $S'_0 := \{(s_0, q_0)\}$
- $L'((s, q)) := \{q\}$ for $(s, q) \in S'$
- We have for the transition relation \rightarrow' :

$$\begin{aligned} \rightarrow' := \{ & ((s_0, q_0), \tau, (s_0, q_0)), ((s_0, q_0), \tau, (s_1, q_0)), ((s_0, q_0), \tau, (s_2, q_0)), \\ & ((s_1, q_0), \tau, (s_0, q_0)), ((s_1, q_0), \tau, (s_1, q_0)), ((s_1, q_0), \tau, (s_3, q_0)), \\ & ((s_1, q_0), \tau, (s_3, q_1)), ((s_2, q_0), \tau, (s_0, q_0)), ((s_2, q_0), \tau, (s_2, q_0)), \\ & ((s_2, q_0), \tau, (s_3, q_0)), ((s_2, q_0), \tau, (s_3, q_1)), ((s_3, q_0), \tau, (s_1, q_0)), \\ & ((s_3, q_0), \tau, (s_2, q_0)), ((s_3, q_0), \tau, (s_3, q_0)), ((s_3, q_0), \tau, (s_3, q_1)), \\ & ((s_3, q_1), \tau, (s_1, q_0)), ((s_3, q_1), \tau, (s_2, q_0)), ((s_3, q_1), \tau, (s_3, q_1)) \} \end{aligned}$$

The visualisation of \mathcal{T}' looks as shown in Figure 2.1 on page 25.

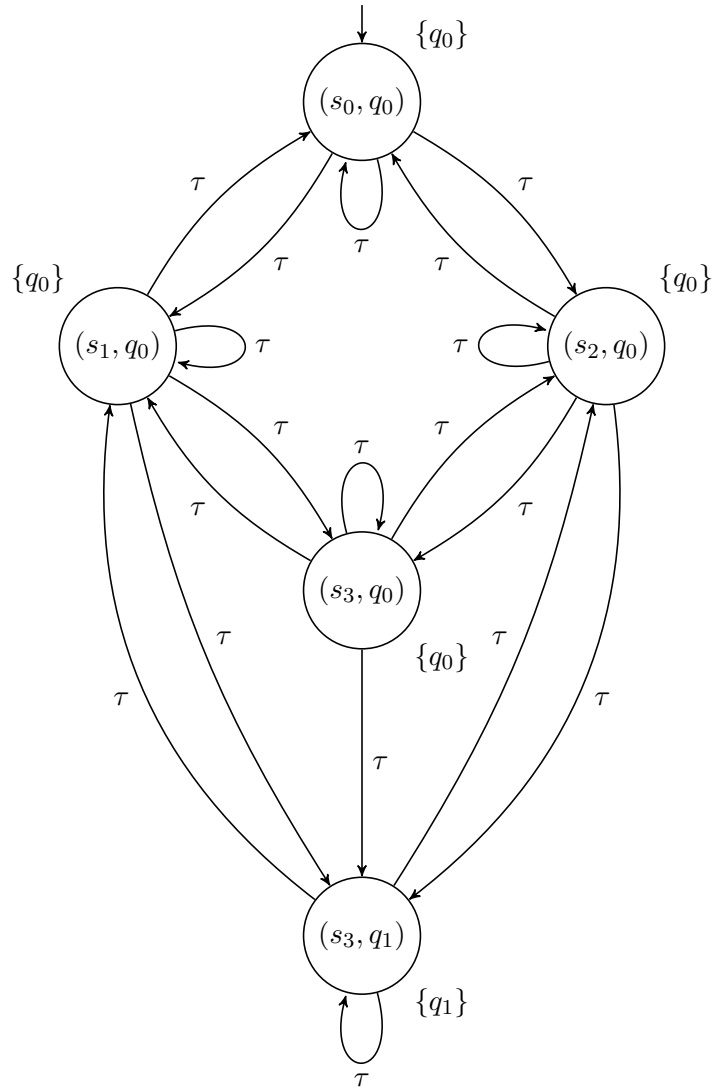
The explicit-state model checking algorithm searches now for reachable states that are labelled with a final state of \mathcal{N} . In our example, there is only one such state—namely (s_3, q_1) .

As we can see in Figure 2.1, the state (s_3, q_1) is reached for example via (s_0, q_0) and (s_1, q_0) . Since there exists a self-loop of (s_3, q_1) , this state is trivially located on a cycle. Thus, we have the following path as a counterexample:

$$(s_0, q_0), (s_1, q_0), (s_3, q_1), (s_3, q_1), (s_3, q_1), \dots$$

Hence, the specification does not hold for the labelled transition system \mathcal{T} . This behaviour is to be expected since the system does not ensure that the two processes never work within the critical section at the same time.

Figure 2.1: The reachable part of the product labelled transition system \mathcal{T}'



Note that this counterexample is not the only one, but finding one is sufficient.

We have now considered explicit-state model checking for LTL in more detail. As we have seen, two things are needed for formal verification: the full model for the system behaviour and an LTL-formula for the specification.

For runtime verification, we do not need the full model for the system behaviour. It suffices to observe the relevant system behaviour. Subsequently, we will consider runtime verification for LTL and its differences to explicit-state model checking for LTL.

2.3.2 Runtime verification for LTL

In this subsection, we are going to introduce runtime verification for LTL on a formal level. An informal idea of runtime verification was given in the introductory text.

Runtime verification [7] deals with the question whether a observed system meets the specification. The specification is formalised in terms of a formula of temporal logic—in this case, the temporal logic used is LTL. The system’s behaviour is not formalised explicitly like in the case of explicit-state model checking. But the system behaviour can be observed and formally represented as a finite sequence of sets of propositional variables that evaluate to true in the current state of the system.

For performing runtime verification, one defines a so-called monitor corresponding to the specification. This monitor is defined in terms of a finite-state machine with output. The input alphabet of this finite-state machine are sets of propositional variables and negated propositional variables.

The next definition introduces the monitor construction.

Definition 2.32. *Let φ be an LTL-formula and let $\mathcal{G}_\varphi = (Q^\varphi, \Sigma^\varphi, \Delta^\varphi, Q_0^\varphi, \mathcal{F}^\varphi)$ be the corresponding generalised non-deterministic Büchi automaton.*

The generalised non-deterministic Büchi automaton $\mathcal{G}_{\neg\varphi} = (Q^{\neg\varphi}, \Sigma^{\neg\varphi}, \Delta^{\neg\varphi}, Q_0^{\neg\varphi}, \mathcal{F}^{\neg\varphi})$ is defined analogously.

The finite-state machine $\mathcal{A}_\varphi = (Q, \Sigma, \delta, q_0, \Gamma, \lambda)$ is defined as follows:

- $Q := 2^{Q^\varphi} \times 2^{Q^{\neg\varphi}}$

- $\Sigma := \Sigma^\varphi = \Sigma^{\neg\varphi}$
- $q_0 := (Q_0^\varphi, Q_0^{\neg\varphi})$
- For all $S \subseteq Q^\varphi$, $S' \subseteq Q^{\neg\varphi}$ and $\sigma \in \Sigma$ we have:

$\delta((S, S'), \sigma) := (\hat{S}, \hat{S}')$ with:

$$\begin{aligned} - \hat{S} &= \bigcup_{q \in S} \{\hat{q} \in Q^\varphi \mid (q, \sigma, \hat{q}) \in \Delta^\varphi\} \\ - \hat{S}' &= \bigcup_{q' \in S'} \{\hat{q}' \in Q^{\neg\varphi} \mid (q', \sigma, \hat{q}') \in \Delta^{\neg\varphi}\} \end{aligned}$$

- $\Gamma := \mathbb{B}_3$ where $\mathbb{B}_3 := \{0, 1, ?\}$ where ? stands for inconclusive
- $\lambda : Q \rightarrow \mathbb{B}_3$ is defined by

$$\lambda((S, S')) = \begin{cases} 1 & \text{if } L(\mathcal{G}_{\neg\varphi}(q')) = \emptyset \text{ for all } q' \in S' \text{ and} \\ & L(\mathcal{G}_\varphi(q)) \neq \emptyset \text{ for some } q \in S \\ 0 & \text{if } L(\mathcal{G}_\varphi(q)) = \emptyset \text{ for all } q \in S \text{ and} \\ & L(\mathcal{G}_{\neg\varphi}(q')) \neq \emptyset \text{ for some } q' \in S' \\ ? & \text{otherwise} \end{cases}$$

This definition is adopted for \mathcal{ALC} -LTL later on—see Definition 3.16. The proof of correctness is also provided later on.

We will now consider the definition for the monitor construction more precisely. Given an LTL-formula φ , the first step is to construct the generalised non-deterministic Büchi automata for φ and $\neg\varphi$. Note that this construction also works using the standard version of non-deterministic Büchi automata. These two automata \mathcal{G}_φ and $\mathcal{G}_{\neg\varphi}$ are composed in parallel and made deterministic using the power-set construction.

The most interesting step is the definition of the output function λ . As we can see in the above definition, the function value for a particular state depends on the solution of the emptiness problem for several generalised non-deterministic Büchi automata. This has to be kept in mind if one performs a complexity analysis.

This finite-state machine can now be used as monitoring device. The monitor reads the system's behaviour in terms of words over 2^P . This means that all the propositional variables occurring in φ that evaluate to true in the current state of the system belong to the current set of propositional variables that the monitor consumes.

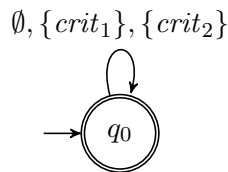
Note that the monitor is completely independent of the system. It only depends on the specification. Thus, one can use the same monitor for several systems, provided the specifications match.

We will now give an example for the monitor construction. We will use again the specification φ_1 of Example 2.22.

Example 2.33. *First, we need the automata for $\varphi_1 = \Box\neg(\text{crit}_1 \wedge \text{crit}_2)$ and $\neg\varphi_1 = \neg\Box\neg(\text{crit}_1 \wedge \text{crit}_2)$. The tool LTL 2 BA [10] yields the following non-deterministic Büchi automaton $\mathcal{N}_{\varphi_1} = (Q, \Sigma, \Delta, Q_0, F)$ with:*

- $Q := \{q_0\}$
- $Q_0 := \{q_0\}$
- $F := \{q_0\}$
- $\Sigma := 2^{\{\text{crit}_1, \text{crit}_2\}}$
- *The transition relation Δ has only the following elements.*
 - $(q_0, \emptyset, q_0) \in \Delta$
 - $(q_0, \{\text{crit}_1\}, q_0) \in \Delta$
 - $(q_0, \{\text{crit}_2\}, q_0) \in \Delta$

The visualisation of \mathcal{N}_{φ_1} looks as follows.

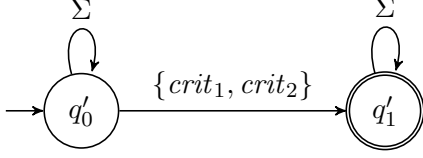


LTL 2 BA yields the following non-deterministic Büchi automaton $\mathcal{N}_{\neg\varphi_1}$ with $\mathcal{N}_{\neg\varphi_1} = (Q', \Sigma, \Delta', Q'_0, F')$ where:

- $Q' := \{q'_0, q'_1\}$
- $Q'_0 := \{q'_0\}$
- $F' := \{q'_1\}$
- *The transition relation Δ' has only the following elements.*
 - $(q'_0, \sigma, q'_0) \in \Delta'$ for all $\sigma \in \Sigma$
 - $(q'_0, \{\text{crit}_1, \text{crit}_2\}, q'_1) \in \Delta'$

- $(q'_1, \sigma, q'_1) \in \Delta'$ for all $\sigma \in \Sigma$

The visualisation of $\mathcal{N}_{-\varphi_1}$ looks as follows.

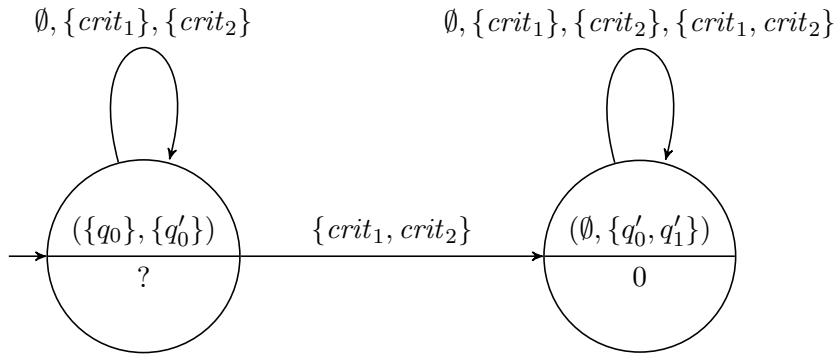


Note that $\xrightarrow{\Sigma}$ signifies all transitions $\xrightarrow{\sigma}$ with $\sigma \in \Sigma$.

Next, we construct the monitor using Definition 2.32. We get the following reachable part of the finite-state machine with output, denoted by $\mathcal{A}_\varphi = (Q, \Sigma, \delta, q_0, \Gamma, \lambda)$ where:

- $Q := \{ (\{q_0\}, \{q'_0\}), (\emptyset, \{q'_0, q'_1\}) \}$
- $\Sigma := 2^{\{crit_1, crit_2\}}$
- $q_0 := (\{q_0\}, \{q'_0\})$
- $\Gamma := \mathbb{B}_3$
- The transition function $\delta : Q \times \Sigma \rightarrow Q$ is defined as follows:
 - $\delta((\{q_0\}, \{q'_0\}), \emptyset) := (\{q_0\}, \{q'_0\})$
 - $\delta((\{q_0\}, \{q'_0\}), \{crit_1\}) := (\{q_0\}, \{q'_0\})$
 - $\delta((\{q_0\}, \{q'_0\}), \{crit_2\}) := (\{q_0\}, \{q'_0\})$
 - $\delta((\{q_0\}, \{q'_0\}), \{crit_1, crit_2\}) := (\emptyset, \{q'_0, q'_1\})$
 - $\delta((\emptyset, \{q'_0, q'_1\}), \emptyset) := (\emptyset, \{q'_0, q'_1\})$
 - $\delta((\emptyset, \{q'_0, q'_1\}), \{crit_1\}) := (\emptyset, \{q'_0, q'_1\})$
 - $\delta((\emptyset, \{q'_0, q'_1\}), \{crit_2\}) := (\emptyset, \{q'_0, q'_1\})$
 - $\delta((\emptyset, \{q'_0, q'_1\}), \{crit_1, crit_2\}) := (\emptyset, \{q'_0, q'_1\})$
- For the output function $\lambda : Q \rightarrow \Gamma$ we have the following:
 - $\lambda((\{q_0\}, \{q'_0\})) := ?$
 - $\lambda((\emptyset, \{q'_0, q'_1\})) := 0$

The visualisation of \mathcal{A}_φ looks as follows.



If the observation of the system behaviour is formalised as the following finite word

$$w = \emptyset \emptyset \{crit_1\} \{crit_1\} \emptyset \{crit_2\} \{crit_1, crit_2\}$$

then \mathcal{A}_φ yields the following output:

$$? ? ? ? ? ? 0$$

As we can see, the specification is violated at the current state of the system. The system engineer could take action in order to prevent severe consequences.

Obviously, in this example the monitor never outputs 1. This behaviour is to be expected since safety properties have to be satisfied at all times. Using runtime verification we do evaluate the system whilst it is running. It is impossible to verify a safety property without knowing the complete system's behaviour. Having observed the system's behaviour, a finite time does not suffice. In order to verify safety properties, we could use explicit-state model checking.

Explicit-state model checking and runtime verification differ in many aspects. Explicit-state model checking requires a formalisation of the complete system's behaviour. Thus one can also verify safety properties using explicit-state model checking. For runtime verification, one needs only to observe the system. The monitor for a specification can be reused for other systems with the same specification. Thus, model checking can only be used for white-box system whereas runtime verification can also be used for black-box systems.

3 Runtime verification for \mathcal{ALC} -LTL without rigid names

In this chapter, we will consider the easiest case for runtime verification using the temporal description logic \mathcal{ALC} -LTL. We define how to construct the monitor for a specification given in terms of an \mathcal{ALC} -LTL formula without rigid names.

First, we will give details for the notion of rigid names. We will show why it makes sense to distinguish between \mathcal{ALC} -LTL with rigid names and \mathcal{ALC} -LTL without rigid names.

Next, we will give details about the monitor construction for \mathcal{ALC} -LTL without rigid names. This construction is done step-by-step and necessary definitions are given successively. Additionally, the correctness of the monitor construction is proved formally.

This is followed by an analysis of the complexity of the monitor construction for \mathcal{ALC} -LTL without rigid names.

3.1 \mathcal{ALC} -LTL with and without rigid names

Often it is desired that the interpretations of specific concept and role names do not vary over time. Definition 2.26 introduces a formal notion of rigid names.

We distinguish between three cases:

- \mathcal{ALC} -LTL without rigid names
- \mathcal{ALC} -LTL with rigid concept names
- \mathcal{ALC} -LTL with rigid names

\mathcal{ALC} -LTL without rigid names means that all concept names as well as all role names are flexible. By \mathcal{ALC} -LTL with rigid concept names we denote the case where rigid concept names are available, but all role names are flexible. \mathcal{ALC} -LTL with rigid names denotes the logic where both rigid concept names and rigid role names are available.

It makes sense to distinguish these three cases if we want to determine the complexity of reasoning; see [2] for details. This will also play a role when analysing the complexity for the monitor construction.

However, this distinction is not only crucial for determining the complexity, but also for the monitor construction itself. In fact, the construction of the monitor differs depending on the fact whether rigid names are available or not. In this chapter, we consider runtime verification for \mathcal{ALC} -LTL without rigid names.

3.2 Defining the monitor for an \mathcal{ALC} -LTL formula without rigid names

In this section, the monitor construction for \mathcal{ALC} -LTL without rigid names is introduced stepwise. We start by defining for each \mathcal{ALC} -LTL formula without rigid names the corresponding generalised non-deterministic Büchi automaton. We will explain what the notion corresponding means in this context. This sets the basis for the construction of the monitor given in terms of a finite-state machine with output. This section concludes by proving the correctness of the monitor construction.

Please note that most of the definitions in this chapter do also apply for \mathcal{ALC} -LTL with rigid names. Therefore the term “without rigid names” is often put in parentheses or this term is not given at all in order to refer to the general case.

3.2.1 The generalised non-deterministic Büchi automaton for an \mathcal{ALC} -LTL formula without rigid names

Before we can define the construction of the corresponding generalised non-deterministic Büchi automaton for a given \mathcal{ALC} -LTL formula without rigid names, we need some definitions and notations. Note that we follow the approach of [17, 16]. For the case of \mathcal{ALC} -LTL without rigid names, we adapt the construction of the Büchi automaton for LTL only slightly.

We start by defining the notion of the closure of an \mathcal{ALC} -LTL formula with or without rigid names.

Definition 3.1. *Let φ be an \mathcal{ALC} -LTL formula (without rigid names). The closure of φ —denoted by $cl(\varphi)$ —is defined as the set that consists of all sub-formulas ψ of φ and their negations $\neg\psi$. Thereby we identify ψ and $\neg\neg\psi$.*

For the construction of the generalised non-deterministic Büchi automaton for a given \mathcal{ALC} -LTL formula without rigid names, it is also important to denote the description logic axioms occurring in this formula. The next definition introduces this set as well as the set of description logic axioms that is closed under complementation.

Definition 3.2. *Let φ be an \mathcal{ALC} -LTL formula (without rigid names). We denote the set of description logic axioms occurring in φ by $ax(\varphi)$. Additionally, the set $ax'(\varphi)$ is defined by*

$$ax'(\varphi) := ax(\varphi) \cup \{\neg\psi \mid \psi \in ax(\varphi)\}$$

Having defined these three sets, let us now look at an example for these definitions.

Example 3.3. *Let $\varphi := (C \sqsubseteq D) \mathcal{U} (\bigcirc((C \sqsubseteq \exists r.A) \wedge (a : C)))$. We have*

$$cl(\varphi) = \{ \begin{array}{l} \varphi, \neg\varphi, C \sqsubseteq D, \neg(C \sqsubseteq D), \\ \bigcirc((C \sqsubseteq \exists r.A) \wedge (a : C)), \neg \bigcirc((C \sqsubseteq \exists r.A) \wedge (a : C)), \\ (C \sqsubseteq \exists r.A) \wedge (a : C), \neg((C \sqsubseteq \exists r.A) \wedge (a : C)), \\ C \sqsubseteq \exists r.A, \neg(C \sqsubseteq \exists r.A), a : C, \neg(a : C) \end{array} \}$$

$$ax(\varphi) = \{ C \sqsubseteq D, C \sqsubseteq \exists r.A, a : C \}$$

$$ax'(\varphi) = \{ \begin{array}{l} C \sqsubseteq D, C \sqsubseteq \exists r.A, a : C, \\ \neg(C \sqsubseteq D), \neg(C \sqsubseteq \exists r.A), \neg(a : C) \end{array} \}$$

As in the case of propositional LTL, the construction of the corresponding generalised non-deterministic Büchi automaton requires the notion of a type. The set of types will represent the set of states of the Büchi automaton later on.

Definition 3.4. *Let φ be an \mathcal{ALC} -LTL formula (without rigid names). An \mathcal{ALC} -LTL type (or short: type) for φ is a subset $T \subseteq cl(\varphi)$ with the following properties:*

- T is consistent w. r. t. propositional logic and maximal, i. e.:

- $\psi \in T \Leftrightarrow \neg\psi \notin T$ for all $\neg\psi \in cl(\varphi)$
- $\psi_1 \wedge \psi_2 \in T \Leftrightarrow \{\psi_1, \psi_2\} \subseteq T$ for all $\psi_1 \wedge \psi_2 \in cl(\varphi)$
- T is locally consistent w. r. t. the \mathcal{U} -modality, i. e. we have for all $\psi_1 \mathcal{U} \psi_2 \in cl(\varphi)$ that the following holds:
 - $\psi_2 \in T \Rightarrow \psi_1 \mathcal{U} \psi_2 \in T$
 - $\psi_1 \mathcal{U} \psi_2 \in T$ and $\psi_2 \notin T \Rightarrow \psi_1 \in T$
- T is consistent w. r. t. \mathcal{ALC} , i. e. the Boolean knowledge base

$$\mathcal{B} := \bigwedge_{\alpha \in T \cap ax'(\varphi)} \alpha$$

is consistent.

We denote the set of all types for φ by $ty(\varphi)$.

Since description logic axioms may depend on each other, it makes sense to define the notion of a type also on the level of \mathcal{ALC} . The set of \mathcal{ALC} -types is supposed to be the alphabet for the Büchi automaton.

Definition 3.5. Let φ be an \mathcal{ALC} -LTL formula (without rigid names). An \mathcal{ALC} -type for φ is a subset $T \subseteq ax'(\varphi)$ with the following properties:

- T is consistent w. r. t. propositional logic and maximal, i. e.:
 - $\psi \in T \Leftrightarrow \neg\psi \notin T$ for all $\neg\psi \in ax'(\varphi)$
- T is consistent w. r. t. \mathcal{ALC} , i. e. the Boolean knowledge base

$$\mathcal{B} := \bigwedge_{\alpha \in T} \alpha$$

is consistent.

We denote the set of all \mathcal{ALC} -types for φ by $ty_{ax}(\varphi)$.

Note: The second requirement of the definition does not imply the first one. This is due to the fact that a Boolean knowledge base does not need to be maximal, i. e. for all $\psi \in ax'(\varphi)$, either ψ or $\neg\psi$ is contained in the \mathcal{ALC} -type.

Obviously, \mathcal{ALC} -types and \mathcal{ALC} -LTL types are closely related. We will now show the connection between these two sorts of types.

Lemma 3.6. *Let T be an \mathcal{ALC} -LTL type for an \mathcal{ALC} -LTL formula φ . Then it holds that $T \cap ax'(\varphi)$ is always an \mathcal{ALC} -type.*

Proof. Obviously, consistency of T w. r. t. \mathcal{ALC} yields that $T \cap ax'(\varphi)$ is consistent w. r. t. \mathcal{ALC} .

If T is consistent w. r. t. propositional logic, then it holds that

$$\psi \in T \text{ iff } \neg\psi \notin T \text{ for all } \neg\psi \in cl(\varphi)$$

Thus, we have that

$$\psi \in T \cap ax'(\varphi) \text{ iff } \neg\psi \notin T \cap ax'(\varphi) \text{ for all } \neg\psi \in ax'(\varphi)$$

□

The following definition shows how \mathcal{ALC} -types and \mathcal{ALC} -interpretations are connected.

Definition 3.7. *Let φ be an \mathcal{ALC} -LTL formula and let $Int_{\mathcal{ALC}}(\varphi)$ denote the set of all \mathcal{ALC} -interpretations over names occurring in $ax(\varphi)$.*

We define a function $\tau : Int_{\mathcal{ALC}}(\varphi) \rightarrow ty_{ax}(\varphi)$ as follows:

$$\tau(\mathcal{I}) = \{\alpha \in ax'(\varphi) \mid \mathcal{I} \models \alpha\}$$

for all $\mathcal{I} \in Int_{\mathcal{ALC}}(\varphi)$.

Next, we will show that every \mathcal{ALC} -interpretation \mathcal{I} has an \mathcal{ALC} -type, i. e. $\tau(\mathcal{I})$ is really an \mathcal{ALC} -type. Additionally, we show that for each \mathcal{ALC} -type, there exists at least one \mathcal{ALC} -interpretation.

Lemma 3.8. *Let φ be an \mathcal{ALC} -LTL formula and let $Int_{\mathcal{ALC}}(\varphi)$ denote the set of all \mathcal{ALC} -interpretations over names occurring in $ax(\varphi)$. Then we have the following:*

- (1) *For every \mathcal{ALC} -interpretation $\mathcal{I} \in Int_{\mathcal{ALC}}(\varphi)$: $\tau(\mathcal{I})$ is really an \mathcal{ALC} -type.*
- (2) *For every $T \in ty_{ax}(\varphi)$, there exists an \mathcal{ALC} -interpretation $\mathcal{I} \in Int_{\mathcal{ALC}}(\varphi)$ such that $\tau(\mathcal{I}) = T$.*

Proof. (1) Obviously, for every \mathcal{I} , the value of $\tau(\mathcal{I})$ is always a non-empty subset of $ax'(\varphi)$. $\tau(\mathcal{I})$ is an \mathcal{ALC} -type since it is consistent w. r. t. \mathcal{ALC} and $\tau(\mathcal{I})$ is maximal, i. e. for all $\neg\psi \in ax'(\varphi)$, we have that $\psi \in \tau(\mathcal{I})$ iff $\neg\psi \notin \tau(\mathcal{I})$.

(2) By definition of $ty_{ax}(\varphi)$, every $T \in ty_{ax}(\varphi)$ is consistent w. r. t. \mathcal{ALC} , i. e. there exists an interpretation $\mathcal{I} \in Int_{\mathcal{ALC}}(\varphi)$ such that $\mathcal{I} \models T$. The definition of \mathcal{ALC} -types and the definition of $\tau(\mathcal{I})$ yield that $\tau(\mathcal{I}) = T$.

□

Next, we define for each \mathcal{ALC} -LTL formula without rigid names φ a generalised non-deterministic Büchi automaton $\mathcal{G}_\varphi = (Q^\varphi, \Sigma^\varphi, \Delta^\varphi, Q_0^\varphi, \mathcal{F}^\varphi)$ such that the following holds for all \mathcal{ALC} -LTL interpretations $\mathfrak{J} = (\mathcal{I}_i)_{i \geq 0}$:

$$\mathfrak{J}, 0 \models \varphi \text{ iff } \tau(\mathcal{I}_0)\tau(\mathcal{I}_1)\tau(\mathcal{I}_2) \cdots \in L(\mathcal{G}_\varphi)$$

Definition 3.9. *Let φ be an \mathcal{ALC} -LTL formula without rigid names. We define the corresponding generalised non-deterministic Büchi automaton $\mathcal{G}_\varphi = (Q^\varphi, \Sigma^\varphi, \Delta^\varphi, Q_0^\varphi, \mathcal{F}^\varphi)$ as follows.*

- $Q^\varphi := ty(\varphi)$
- $\Sigma^\varphi := ty_{ax}(\varphi)$
- $Q_0^\varphi := \{q \in Q^\varphi \mid \varphi \in q\}$
- $\mathcal{F}^\varphi := \{F_{\psi_1 \mathcal{U} \psi_2} \mid \psi_1 \mathcal{U} \psi_2 \in cl(\varphi)\}$ where $F_{\psi_1 \mathcal{U} \psi_2} := \{q \in Q^\varphi \mid \psi_1 \mathcal{U} \psi_2 \notin q \text{ or } \psi_2 \in q\}$
- $\Delta^\varphi \subseteq Q^\varphi \times \Sigma^\varphi \times Q^\varphi$. This relation is defined as follows. $(q, \sigma, q') \in \Delta^\varphi$ iff
 - $\sigma = q \cap ax'(\varphi)$
 - For every formula $\bigcirc\psi \in cl(\varphi)$, we have that $\bigcirc\psi \in q \Leftrightarrow \psi \in q'$.
 - For every formula $\psi_1 \mathcal{U} \psi_2 \in cl(\varphi)$, we have that $\psi_1 \mathcal{U} \psi_2 \in q \Leftrightarrow (\psi_2 \in q \vee (\psi_1 \in q \wedge \psi_1 \mathcal{U} \psi_2 \in q'))$.

The next theorem states the soundness and completeness of this construction, i. e. we prove that the generalised non-deterministic Büchi automaton for an \mathcal{ALC} -LTL formula without rigid names accepts all desired sequences of \mathcal{ALC} -types but not more.

Theorem 3.10. *Let φ be an \mathcal{ALC} -LTL formula without rigid names and \mathcal{G}_φ the corresponding generalised non-deterministic Büchi automaton. Then the following holds for all \mathcal{ALC} -LTL interpretations $\mathfrak{J} = (\mathcal{I}_i)_{i \geq 0}$.*

$$\tau(\mathcal{I}_0)\tau(\mathcal{I}_1)\tau(\mathcal{I}_2) \cdots \in L(\mathcal{G}_\varphi) \Leftrightarrow \mathfrak{J}, 0 \models \varphi$$

Proof. “ \Leftarrow ”. Suppose $\mathfrak{I}, 0 \models \varphi$. We define

$$S_i := \{\psi \in cl(\varphi) \mid \mathfrak{I}, i \models \psi\}$$

$S_0S_1S_2\dots$ is a run of \mathcal{G}_φ on $\tau(\mathcal{I}_0)\tau(\mathcal{I}_1)\tau(\mathcal{I}_2)\dots$ since:

- $S_i \in Q^\varphi$ for all $i \geq 0$. This holds since S_i is a non-empty subset of $cl(\varphi)$ that is consistent w. r. t. \mathcal{ALC} -LTL, maximal, and thus an \mathcal{ALC} -LTL type.
- $\varphi \in S_0$ and thus $S_0 \in Q_0^\varphi$
- $\tau(\mathcal{I}_i) = S_i \cap ax'(\varphi)$. This holds because of the definition of $\tau(\mathcal{I}_i)$ and Lemma 3.6.
- for all $\bigcirc\psi \in cl(\varphi)$, we have: $\bigcirc\psi \in S_i$ iff $\mathfrak{I}, i \models \bigcirc\psi$ iff $\mathfrak{I}, i+1 \models \psi$ iff $\psi \in S_{i+1}$
- for all $\psi_1\mathcal{U}\psi_2 \in cl(\varphi)$, we have:

$$\begin{aligned} & \psi_1\mathcal{U}\psi_2 \in S_i \\ \text{iff } & \mathfrak{I}, i \models \psi_1\mathcal{U}\psi_2 \\ \text{iff } & \mathfrak{I}, i \models \psi_2 \text{ or } \mathfrak{I}, i \models \psi_1 \text{ and } \mathfrak{I}, i+1 \models \psi_1\mathcal{U}\psi_2 \\ \text{iff } & \psi_2 \in S_i \text{ or } \psi_1 \in S_i \text{ and } \psi_1\mathcal{U}\psi_2 \in S_{i+1} \end{aligned}$$

- thus, $(S_i, \tau(\mathcal{I}_i), S_{i+1}) \in \Delta^\varphi$ for all $i \geq 0$

Moreover, $S_0S_1S_2\dots$ is accepting: Suppose that for some $\psi_1\mathcal{U}\psi_2 \in cl(\varphi)$ the set $\{i \in \mathbb{N} \mid S_i \in F_{\psi_1\mathcal{U}\psi_2}\}$ is finite. Then there exists a $k \in \mathbb{N}$ such that $S_i \notin F_{\psi_1\mathcal{U}\psi_2}$ for all $i \geq k$. This means $\psi_1\mathcal{U}\psi_2 \in S_i$ and $\psi_2 \notin S_i$ for all $i \geq k$. Hence, $\mathfrak{I}, k \models \psi_1\mathcal{U}\psi_2$ and $\mathfrak{I}, i \not\models \psi_2$ for all $i \geq k$. This is a contradiction to the semantics of the \mathcal{U} -modality. ζ

“ \Rightarrow ”. Let $\tau(\mathcal{I}_0)\tau(\mathcal{I}_1)\tau(\mathcal{I}_2)\dots \in L(\mathcal{G}_\varphi)$ and let $S_0S_1S_2\dots$ be an accepting run of \mathcal{G}_φ on $\tau(\mathcal{I}_0)\tau(\mathcal{I}_1)\tau(\mathcal{I}_2)\dots$.

We have to show that for the \mathcal{ALC} -LTL interpretation $\mathfrak{I} = (\mathcal{I}_i)_{i \geq 0}$ it holds that $\mathfrak{I}, 0 \models \varphi$.

Since $\varphi \in S_0$ by definition of Q_0^φ and all the \mathcal{ALC} -interpretations \mathcal{I}_i are completely independent from each other, it suffices to show that for all $i \geq 0$ and $\psi \in cl(\varphi)$, we have:

$$\psi \in S_i \text{ iff } \mathfrak{I}, i \models \psi$$

We prove this by induction on the structure of ψ .

- $\psi = \alpha \in ax'(\varphi)$. We have: $\alpha \in S_i$ iff $\alpha \in S_i \cap ax'(\varphi)$ iff $\alpha \in \tau(\mathcal{I}_i)$ iff $\mathcal{I}_i \models \alpha$ iff $\mathfrak{I}, i \models \alpha$.

- $\psi = \neg\vartheta$. We have: $\neg\vartheta \in S_i$ iff $\vartheta \notin S_i$ iff $\mathfrak{I}, i \not\models \vartheta$ iff $\mathfrak{I}, i \models \neg\vartheta$.
- $\psi = \vartheta_1 \wedge \vartheta_2$. We have: $\vartheta_1 \wedge \vartheta_2 \in S_i$ iff $\{\vartheta_1, \vartheta_2\} \subseteq S_i$ iff $\mathfrak{I}, i \models \{\vartheta_1, \vartheta_2\}$ iff $\mathfrak{I}, i \models \vartheta_1 \wedge \vartheta_2$.
- $\psi = \bigcirc\vartheta$. We have: $\bigcirc\vartheta \in S_i$ iff $\vartheta \in S_{i+1}$ iff $\mathfrak{I}, i+1 \models \vartheta$ iff $\mathfrak{I}, i \models \bigcirc\vartheta$.
- $\psi = \vartheta_1 \mathcal{U} \vartheta_2$.

“if”. Let $\mathfrak{I}, i \models \psi$. Then there exists a $k \geq i$ such that $\mathfrak{I}, k \models \vartheta_2$ and $\mathfrak{I}, l \models \vartheta_1$ for $i \leq l < k$. We show by induction on j that $\vartheta_1 \mathcal{U} \vartheta_2 \in S_{k-j}$ for $j \leq k - i$.

Induction start: $\mathfrak{I}, k \models \vartheta_2$ implies $\vartheta_2 \in S_k$ by the outer induction hypothesis, and local consistency w. r. t. \mathcal{U} yields $\vartheta_1 \mathcal{U} \vartheta_2 \in S_k$.

Induction step: $\mathfrak{I}, k - j \models \vartheta_1$ implies by outer induction hypothesis $\vartheta_1 \in S_{k-j}$. The inner induction hypothesis yields the following: $\vartheta_1 \mathcal{U} \vartheta_2 \in S_{k-j+1}$. Thus, by definition of Δ^φ , it follows that $\vartheta_1 \mathcal{U} \vartheta_2 \in S_{k-j}$.

“only if”. Let $\psi \in S_i$. Since states of $F_{\vartheta_1 \mathcal{U} \vartheta_2}$ appear infinitely often among $S_0 S_1 S_2 \dots$, there is a $k \geq i$ such that $S_k \in F_{\vartheta_1 \mathcal{U} \vartheta_2}$. Let k be smallest with this property. Then it follows that $\vartheta_1 \mathcal{U} \vartheta_2 \in S_l$ and $\vartheta_2 \notin S_l$ for $i \leq l < k$.

- (1) $\vartheta_1 \mathcal{U} \vartheta_2 \in S_l$ and $\vartheta_2 \in S_l$ for $i \leq l < k$ yield $\vartheta_1 \in S_l$ because of the local consistency w. r. t. the \mathcal{U} -modality. Thus, $\mathfrak{I}, l \models \vartheta_1$ for $i \leq l < k$.
 - (2) $\vartheta_1 \mathcal{U} \vartheta_2 \in S_{k-1}$ and $\vartheta_2 \notin S_{k-1}$ imply $\vartheta_1 \mathcal{U} \vartheta_2 \in S_k$ because of the definition of Δ^φ . This yields $\vartheta_2 \in S_k$ since $S_k \in F_{\vartheta_1 \mathcal{U} \vartheta_2}$, and hence $\mathfrak{I}, k \models \vartheta_2$.
- (1) and (2) yield with the semantics of the \mathcal{U} -modality the following: $\mathfrak{I}, i \models \vartheta_1 \mathcal{U} \vartheta_2$.

□

3.2.2 The monitor construction for \mathcal{ALC} -LTL without rigid names

Having defined the corresponding generalised non-deterministic Büchi automaton for an \mathcal{ALC} -LTL formula without rigid names, we can proceed to define the monitor in terms of a finite-state machine with output as indicated in Subsection 2.3.2. Beforehand, we need further definitions and notations.

We start by defining the concatenation of finite sequences of \mathcal{ALC} -interpretations and \mathcal{ALC} -LTL interpretations.

Definition 3.11. Let $\tilde{\mathcal{I}}$ be a finite sequence of \mathcal{ALC} -interpretations, i. e. $\tilde{\mathcal{I}} = \mathcal{I}'_0 \mathcal{I}'_1 \dots \mathcal{I}'_k$ for some $k \in \mathbb{N}$, and let $\mathfrak{I} = (\mathcal{I}_i)_{i \geq 0}$ denote an \mathcal{ALC} -LTL interpretation. The concatenation $\tilde{\mathcal{I}} \cdot \mathfrak{I}$ is defined as follows.

$$\tilde{\mathcal{I}} \cdot \mathcal{J} := \mathcal{I}'_0 \mathcal{I}'_1 \dots \mathcal{I}'_k \mathcal{I}_0 \mathcal{I}_1 \dots$$

Next, we want to define a function that maps an \mathcal{ALC} -LTL formula and a finite sequence of \mathcal{ALC} -interpretations to the set of truth values $\mathbb{B}_3 := \{0, 1, ?\}$ where $?$ stands for *inconclusive*. This function is used to evaluate \mathcal{ALC} -LTL formulas w. r. t. finite sequences of \mathcal{ALC} -interpretations.

Definition 3.12. *Let φ be an \mathcal{ALC} -LTL formula and let $\tilde{\mathcal{I}}$ be a finite sequence of \mathcal{ALC} -interpretations. The function f denotes the evaluation of φ w. r. t. the finite sequence $\tilde{\mathcal{I}}$ and is defined as follows.*

$$f(\varphi, \tilde{\mathcal{I}}) := \begin{cases} 1 & \text{if for all } \mathcal{ALC}\text{-LTL interpretations } \mathcal{J}: \tilde{\mathcal{I}} \cdot \mathcal{J}, 0 \models \varphi \\ 0 & \text{if for all } \mathcal{ALC}\text{-LTL interpretations } \mathcal{J}: \tilde{\mathcal{I}} \cdot \mathcal{J}, 0 \not\models \varphi \\ ? & \text{otherwise} \end{cases}$$

Next, we define the set of states of the generalised non-deterministic Büchi automaton corresponding to an \mathcal{ALC} -LTL formula that are reachable via transitions labelled with the types of a specific sequence of \mathcal{ALC} -interpretations.

Definition 3.13. *Let φ be an \mathcal{ALC} -LTL formula and let $\mathcal{G}_\varphi = (Q^\varphi, \Sigma^\varphi, \Delta^\varphi, Q_0^\varphi, \mathcal{F}^\varphi)$ be the corresponding generalised non-deterministic Büchi automaton.*

For a finite sequence of \mathcal{ALC} -interpretations $\tilde{\mathcal{I}} = \mathcal{I}_0 \mathcal{I}_1 \dots \mathcal{I}_k$, we define:

$$Q'_\varphi(\tilde{\mathcal{I}}) := \{q \in Q^\varphi \mid (q_0, \tau(\mathcal{I}_0)\tau(\mathcal{I}_1) \dots \tau(\mathcal{I}_k), q) \in \hat{\Delta}^\varphi \text{ for } q_0 \in Q_0^\varphi\}$$

$Q'_{\neg\varphi}(\tilde{\mathcal{I}})$ is defined analogously.

Note that this definition applies also to the case of \mathcal{ALC} -LTL with rigid names. Of course, the corresponding generalised Büchi automaton is defined differently. This is done in Chapter 4.

The next lemma shows the connection between the evaluation function f and the generalised non-deterministic Büchi automaton corresponding to an \mathcal{ALC} -LTL formula—with or without rigid names.

Lemma 3.14. *Let φ be an \mathcal{ALC} -LTL formula and let $\mathcal{G}_\varphi = (Q^\varphi, \Sigma^\varphi, \Delta^\varphi, Q_0^\varphi, \mathcal{F}^\varphi)$ be the corresponding generalised non-deterministic Büchi automaton.*

$\mathcal{G}_{\neg\varphi} = (Q^{\neg\varphi}, \Sigma^{\neg\varphi}, \Delta^{\neg\varphi}, Q_0^{\neg\varphi}, \mathcal{F}^{\neg\varphi})$ is defined analogously.

For all finite sequences of \mathcal{ALC} -interpretations $\tilde{\mathcal{I}} = \mathcal{I}'_0 \mathcal{I}'_1 \dots \mathcal{I}'_k$ with $k \in \mathbb{N}$, the following holds:

- (1) $f(\varphi, \tilde{\mathcal{I}}) \neq 0$ iff there exists some $q \in Q'_\varphi(\tilde{\mathcal{I}})$ such that $L(\mathcal{G}_\varphi(q)) \neq \emptyset$
- (2) $f(\varphi, \tilde{\mathcal{I}}) \neq 1$ iff there exists some $q \in Q'_{\neg\varphi}(\tilde{\mathcal{I}})$ such that $L(\mathcal{G}_{\neg\varphi}(q)) \neq \emptyset$

Proof. (1) $f(\varphi, \tilde{\mathcal{I}}) \neq 0$

\Leftrightarrow there is an \mathcal{ALC} -LTL interpretation $\mathfrak{J} = (\mathcal{I}_i)_{i \geq 0}$ with $\tilde{\mathcal{I}} \cdot \mathfrak{J}, 0 \models \varphi$

$\stackrel{(*)}{\Leftrightarrow} \tau(\mathcal{I}'_0)\tau(\mathcal{I}'_1) \dots \tau(\mathcal{I}'_k)\tau(\mathcal{I}_0)\tau(\mathcal{I}_1) \dots$ is an accepting path in \mathcal{G}_φ for some $\mathfrak{J} = (\mathcal{I}_i)_{i \geq 0}$

$\Leftrightarrow \tau(\mathcal{I}_0)\tau(\mathcal{I}_1) \dots$ is an accepting path in $\mathcal{G}_\varphi(q)$ for some $q \in Q'_\varphi(\tilde{\mathcal{I}})$ and some $\mathfrak{J} = (\mathcal{I}_i)_{i \geq 0}$

$\stackrel{(**)}{\Leftrightarrow}$ there exists a $q \in Q'_\varphi(\tilde{\mathcal{I}})$ such that $L(\mathcal{G}_\varphi(q)) \neq \emptyset$

(2) $f(\varphi, \tilde{\mathcal{I}}) \neq 1$

\Leftrightarrow there is an \mathcal{ALC} -LTL interpretation $\mathfrak{J} = (\mathcal{I}_i)_{i \geq 0}$ with $\tilde{\mathcal{I}} \cdot \mathfrak{J}, 0 \not\models \varphi$

\Leftrightarrow there is an \mathcal{ALC} -LTL interpretation $\mathfrak{J} = (\mathcal{I}_i)_{i \geq 0}$ with $\tilde{\mathcal{I}} \cdot \mathfrak{J}, 0 \models \neg\varphi$

$\stackrel{(*)}{\Leftrightarrow} \tau(\mathcal{I}'_0)\tau(\mathcal{I}'_1) \dots \tau(\mathcal{I}'_k)\tau(\mathcal{I}_0)\tau(\mathcal{I}_1) \dots$ is an accepting path in $\mathcal{G}_{\neg\varphi}$ for some $\mathfrak{J} = (\mathcal{I}_i)_{i \geq 0}$

$\Leftrightarrow \tau(\mathcal{I}_0)\tau(\mathcal{I}_1) \dots$ is an accepting path in $\mathcal{G}_{\neg\varphi}(q)$ for some $q \in Q'_{\neg\varphi}(\tilde{\mathcal{I}})$ and some $\mathfrak{J} = (\mathcal{I}_i)_{i \geq 0}$

$\stackrel{(**)}{\Leftrightarrow}$ there exists a $q \in Q'_{\neg\varphi}(\tilde{\mathcal{I}})$ such that $L(\mathcal{G}_{\neg\varphi}(q)) \neq \emptyset$

Note on ():* This equivalence holds because of the acceptance condition for generalised non-deterministic Büchi automata.

*Note on (**):* The direction “ \Leftarrow ” holds since every word $\sigma_0\sigma_1 \dots$ in the non-empty language of the automaton induces a sequence of \mathcal{ALC} -interpretations because of Lemma 3.8. This sequence $\mathcal{I}_0, \mathcal{I}_1, \dots$ yields the \mathcal{ALC} -LTL interpretation $\mathfrak{J} = (\mathcal{I}_i)_{i \geq 0}$.

□

The next lemma shows a consequence of the close relation between the generalised non-deterministic Büchi automata \mathcal{G}_φ and $\mathcal{G}_{\neg\varphi}$ for an \mathcal{ALC} -LTL formula φ —with or without

rigid names.

Lemma 3.15. *Let φ be an \mathcal{ALC} -LTL formula and let $\mathcal{G}_\varphi = (Q^\varphi, \Sigma^\varphi, \Delta^\varphi, Q_0^\varphi, \mathcal{F}^\varphi)$ be the corresponding generalised non-deterministic Büchi automaton.*

The generalised non-deterministic Büchi automaton $\mathcal{G}_{\neg\varphi} = (Q^{\neg\varphi}, \Sigma^{\neg\varphi}, \Delta^{\neg\varphi}, Q_0^{\neg\varphi}, \mathcal{F}^{\neg\varphi})$ is defined analogously.

For all finite sequences of \mathcal{ALC} -interpretations $\tilde{\mathcal{I}} = \mathcal{I}_0\mathcal{I}_1 \dots \mathcal{I}_k$ with $k \in \mathbb{N}$ the following holds:

- (1) *For all $q' \in Q'_{\neg\varphi}(\tilde{\mathcal{I}})$, we have that $L(\mathcal{G}_{\neg\varphi}(q')) = \emptyset$ implies there exists some $q \in Q'_\varphi(\tilde{\mathcal{I}})$ with $L(\mathcal{G}_\varphi(q)) \neq \emptyset$.*
- (2) *For all $q \in Q'_\varphi(\tilde{\mathcal{I}})$, we have that $L(\mathcal{G}_\varphi(q)) = \emptyset$ implies there exists some $q' \in Q'_{\neg\varphi}(\tilde{\mathcal{I}})$ with $L(\mathcal{G}_{\neg\varphi}(q')) \neq \emptyset$.*

Proof. We prove this lemma using Lemma 3.14.

- (1) For all $q' \in Q'_{\neg\varphi}(\tilde{\mathcal{I}})$, we have that $L(\mathcal{G}_{\neg\varphi}(q')) = \emptyset$
iff $f(\varphi, \tilde{\mathcal{I}}) = 1$
implies $f(\varphi, \tilde{\mathcal{I}}) \neq 0$
iff there exists some $q \in Q'_\varphi(\tilde{\mathcal{I}})$ with $L(\mathcal{G}_\varphi(q)) \neq \emptyset$.
- (2) For all $q \in Q'_\varphi(\tilde{\mathcal{I}})$, we have that $L(\mathcal{G}_\varphi(q)) = \emptyset$
iff $f(\varphi, \tilde{\mathcal{I}}) = 0$
implies $f(\varphi, \tilde{\mathcal{I}}) \neq 1$
iff there exists some $q' \in Q'_{\neg\varphi}(\tilde{\mathcal{I}})$ with $L(\mathcal{G}_{\neg\varphi}(q')) \neq \emptyset$.

□

After providing these essential definitions and notations, we are now ready to define the monitor for an \mathcal{ALC} -LTL formula without rigid names in terms of a finite-state machine.

Definition 3.16. *Let φ be an \mathcal{ALC} -LTL formula and let $\mathcal{G}_\varphi = (Q^\varphi, \Sigma^\varphi, \Delta^\varphi, Q_0^\varphi, \mathcal{F}^\varphi)$ be the corresponding generalised non-deterministic Büchi automaton.*

The generalised non-deterministic Büchi automaton $\mathcal{G}_{\neg\varphi} = (Q^{\neg\varphi}, \Sigma^{\neg\varphi}, \Delta^{\neg\varphi}, Q_0^{\neg\varphi}, \mathcal{F}^{\neg\varphi})$ is defined analogously.

The finite-state machine $\mathcal{A}_\varphi = (Q, \Sigma, \delta, q_0, \Gamma, \lambda)$ is defined as follows:

- $Q := 2^{Q^\varphi} \times 2^{Q^{\neg\varphi}}$
- $\Sigma := \Sigma^\varphi = \Sigma^{\neg\varphi}$
- $q_0 := (Q_0^\varphi, Q_0^{\neg\varphi})$
- For all $S \subseteq Q^\varphi$, $S' \subseteq Q^{\neg\varphi}$ and $\sigma \in \Sigma$, we have:

$\delta((S, S'), \sigma) := (\hat{S}, \hat{S}')$ where:

$$\begin{aligned} - \hat{S} &= \bigcup_{q \in S} \{\hat{q} \in Q^\varphi \mid (q, \sigma, \hat{q}) \in \Delta^\varphi\} \\ - \hat{S}' &= \bigcup_{q' \in S'} \{\hat{q}' \in Q^{\neg\varphi} \mid (q', \sigma, \hat{q}') \in \Delta^{\neg\varphi}\} \end{aligned}$$

- $\Gamma := \mathbb{B}_3$
- $\lambda : Q \rightarrow \mathbb{B}_3$ is defined by

$$\lambda((S, S')) = \begin{cases} 1 & \text{if } L(\mathcal{G}_{\neg\varphi}(q')) = \emptyset \text{ for all } q' \in S' \text{ and} \\ & L(\mathcal{G}_\varphi(q)) \neq \emptyset \text{ for some } q \in S \\ 0 & \text{if } L(\mathcal{G}_\varphi(q)) = \emptyset \text{ for all } q \in S \text{ and} \\ & L(\mathcal{G}_{\neg\varphi}(q')) \neq \emptyset \text{ for some } q' \in S' \\ ? & \text{otherwise} \end{cases}$$

Please note the definition of the output function λ . It looks as though we put redundant conditions into the case definition, but upon closer examination it turns out that otherwise the function is not well-defined. This is due to the fact that Lemma 3.15 holds only for states that are reachable via some transitions which are labelled with the types of a specific finite sequence of \mathcal{ALC} -interpretations. For some non-reachable state (S, S') , it can be possible that both $L(\mathcal{G}_{\neg\varphi}(q')) = \emptyset$ for all $q' \in S'$ and $L(\mathcal{G}_\varphi(q)) = \emptyset$ for all $q \in S$ are satisfied. In this case, λ would not be well-defined. To overcome this issue, we add the negated condition to the case definition. Thus, the output function yields for such non-reachable states the value ?.

The next theorem states the correctness of the monitor construction. Intuitively, correctness means that the value of the evaluation function f and the value of the output function λ coincide.

Theorem 3.17. *Let φ be an \mathcal{ALC} -LTL formula and let $\mathcal{A}_\varphi = (Q, \Sigma, \delta, q_0, \lambda)$ be the corresponding monitor. Then, for all $\tilde{\mathcal{I}} = \mathcal{I}'_0 \mathcal{I}'_1 \dots \mathcal{I}'_k$, we have that:*

$$f(\varphi, \tilde{\mathcal{I}}) = \lambda(\hat{\delta}(q_0, \tau(\mathcal{I}'_0) \dots \tau(\mathcal{I}'_k)))$$

Proof. We prove this theorem by showing, using both Lemma 3.14 and Lemma 3.15, that $f(\varphi, \tilde{\mathcal{I}}) = 1 \Leftrightarrow \lambda(\hat{\delta}(q_0, \tau(\mathcal{I}'_0) \dots \tau(\mathcal{I}'_k))) = 1$. Additionally, we have to show that the following holds: $f(\varphi, \tilde{\mathcal{I}}) = 0 \Leftrightarrow \lambda(\hat{\delta}(q_0, \tau(\mathcal{I}'_0) \dots \tau(\mathcal{I}'_k))) = 0$.

These yield the third equivalence $f(\varphi, \tilde{\mathcal{I}}) = ? \Leftrightarrow \lambda(\hat{\delta}(q_0, \tau(\mathcal{I}'_0) \dots \tau(\mathcal{I}'_k))) = ?$.

- (1) $f(\varphi, \tilde{\mathcal{I}}) = 1$
 - iff there exists no $q \in Q'_{\neg\varphi}(\tilde{\mathcal{I}})$ such that $L(\mathcal{G}_{\neg\varphi}(q)) \neq \emptyset$
 - iff for all $q \in Q'_{\neg\varphi}(\tilde{\mathcal{I}})$, we have that $L(\mathcal{G}_{\neg\varphi}(q)) = \emptyset$
 - iff $\lambda((S, S')) = 1$ for $S' = Q'_{\neg\varphi}(\tilde{\mathcal{I}})$ and $S = Q'_\varphi(\tilde{\mathcal{I}})$
 - iff $\lambda(\hat{\delta}(q_0, \tau(\mathcal{I}'_0) \dots \tau(\mathcal{I}'_k))) = 1$
- (2) $f(\varphi, \tilde{\mathcal{I}}) = 0$
 - iff there exists no $q \in Q'_\varphi(\tilde{\mathcal{I}})$ such that $L(\mathcal{G}_\varphi(q)) \neq \emptyset$
 - iff for all $q \in Q'_\varphi(\tilde{\mathcal{I}})$, we have that $L(\mathcal{G}_\varphi(q)) = \emptyset$
 - iff $\lambda((S, S')) = 0$ for $S = Q'_\varphi(\tilde{\mathcal{I}})$ and $S' = Q'_{\neg\varphi}(\tilde{\mathcal{I}})$
 - iff $\lambda(\hat{\delta}(q_0, \tau(\mathcal{I}'_0) \dots \tau(\mathcal{I}'_k))) = 0$

□

Having proved the correctness of the monitor construction, we will now take a closer look at the complexity of this construction.

3.3 The complexity for the monitor construction for \mathcal{ALC} -LTL without rigid names

We will now analyse the complexity of the monitor construction for \mathcal{ALC} -LTL without rigid names. Beforehand, we need the definition of the length of an \mathcal{ALC} -LTL formula.

Definition 3.18. *Let φ be an \mathcal{ALC} -LTL formula. The length of φ —denoted by $|\varphi|$ —is the number of temporal modalities ($\neg, \wedge, \bigcirc, \mathcal{U}$) occurring in φ .*

The construction of the monitor starts with the construction of the generalised non-deterministic Büchi automata for the specification and the negated specification. The next lemma states the complexity of the construction of these automata.

Lemma 3.19. *Let φ be an \mathcal{ALC} -LTL formula without rigid names. The construction of the corresponding generalised non-deterministic Büchi automata \mathcal{G}_φ and $\mathcal{G}_{\neg\varphi}$ can be done in EXPTIME.*

Proof. The states in \mathcal{G}_φ are the types of φ ; see Definition 3.9. Since all types contain half of the elements of $cl(\varphi)$, we have that the number of states in \mathcal{G}_φ is bounded by $2^{\frac{|cl(\varphi)|}{2}}$. Note that $\frac{|cl(\varphi)|}{2}$ is the number of sub-formulas of φ . Clearly, the number of sub-formulas is bounded by $2 \cdot |\varphi|$ and thus $|cl(\varphi)| \leq 4 \cdot |\varphi|$. Hence, the number of states in \mathcal{G}_φ is bounded by $2^{\mathcal{O}(|\varphi|)}$.

In order to determine the types of φ , one has to check the consistency of the Boolean knowledge base $\mathcal{B} = \bigwedge_{\alpha \in T \cap ax'(\varphi)} \alpha$ for each type T . We regard \mathcal{B} as \mathcal{ALC} -LTL formula and check for satisfiability. This can be done in EXPTIME as proved in [2].

The same applies to the construction of $\mathcal{G}_{\neg\varphi}$, which means that the construction of \mathcal{G}_φ and $\mathcal{G}_{\neg\varphi}$ can be done in EXPTIME. □

Using this lemma, we can prove the complexity bound for the whole monitor construction. The next theorem states this complexity bound.

Theorem 3.20. *Let φ be an \mathcal{ALC} -LTL formula without rigid names. The corresponding monitor \mathcal{A}_φ can be constructed in 2-EXPTIME.*

Proof. The monitor \mathcal{A}_φ results from building the cross-product of both \mathcal{G}_φ and $\mathcal{G}_{\neg\varphi}$. Additionally, we use the power-set construction in order to make the monitor deterministic; see Definition 3.16. This causes another exponential blow-up of the size of the state space of \mathcal{A}_φ . Thus, the overall construction can be done in 2-EXPTIME. □

Let us sum up the results of this chapter. We introduced a construction for a monitor for an \mathcal{ALC} -LTL formula without rigid names. That allows us to perform runtime

verification for \mathcal{ALC} -LTL without rigid names. A specification in terms of an \mathcal{ALC} -LTL formula without rigid names enables us to effectively construct a monitor in terms of a finite-state machine with output. This can be done in 2-EXPTIME.

Due to the high complexity of the computation, possible applications of the proposed construction are rather limited in practice, but this is a general problem of runtime verification using temporal description logics. Nonetheless, if the length of the formula for the specification is rather short, the high complexity is no major impediment. Another point worth mentioning is that we considered the worst-case complexity. In practical applications, this complexity bound will not necessarily be reached at all. Further research is needed to determine how the proposed monitor construction behaves in practice.

In this work, we focused on the basic functionality of the construction of a monitor for an \mathcal{ALC} -LTL formula with and without rigid names. As we have seen, it is possible to construct a monitor for \mathcal{ALC} -LTL without rigid names effectively, and we provided formal proof of the correctness of the construction.

The next chapter deals with runtime verification for \mathcal{ALC} -LTL where rigid names are available.

4 Runtime verification for \mathcal{ALC} -LTL with respect to rigid names

In this chapter, we will consider runtime verification using the temporal description logic \mathcal{ALC} -LTL where rigid names are available. We adapt the monitor construction of Chapter 3 in order to respect rigid names.

We explained the notion of rigid names in Section 3.1. We distinguish between \mathcal{ALC} -LTL with rigid concept names and \mathcal{ALC} -LTL with rigid names. For the construction of the monitor, this distinction does not make sense, because the monitor construction is the same for both cases.

First, we will give details for the monitor construction for \mathcal{ALC} -LTL with rigid names. This construction is also done stepwise—as in the case of \mathcal{ALC} -LTL without rigid names. Necessary definitions are given successively while we will use a lot of definitions provided in Chapter 3 that do apply for this case as well. Additionally, we prove the correctness of the monitor construction formally.

This is followed by an analysis of the complexity of the monitor construction for \mathcal{ALC} -LTL with rigid names.

4.1 Defining the monitor for an \mathcal{ALC} -LTL formula with rigid names

In this section, the monitor construction for \mathcal{ALC} -LTL with rigid names is introduced stepwise. Again, we start by defining for each \mathcal{ALC} -LTL formula with rigid names the corresponding generalised non-deterministic Büchi automaton. We will explain what the notion corresponding means in this context. This sets the basis for the construction of the monitor defined in terms of a finite-state machine with output. This chapter finishes by proving of the correctness of the monitor construction.

4.1.1 The generalised non-deterministic Büchi automaton for an \mathcal{ALC} -LTL formula with rigid names

First, we need some definitions and notations before we can define the construction of the corresponding generalised non-deterministic Büchi automaton for a given \mathcal{ALC} -LTL formula with rigid names. Again—like in Chapter 3—we follow the approach of [17, 16]. Of course, we have to adapt the construction in order to treat \mathcal{ALC} -LTL formulas with rigid names.

Since the Büchi automaton corresponding to an \mathcal{ALC} -LTL formula with rigid names has to respect rigid names, the transitions of this automaton are not independent of each other. This is by reason of the fact that rigid names have to be interpreted always in the same way. Having taken a transition in the generalised non-deterministic Büchi automaton, we thus must not forget the previous state. Additionally, we have to check in the current state whether there is an \mathcal{ALC} -interpretation that satisfies both the type of the last state and the type of the current state and also interprets the rigid names in the same manner.

In order to respect this, the state needs not only depend on an \mathcal{ALC} -LTL type but also on the types of the states from which this state is reachable. Hence, the set of the states of the generalised non-deterministic Büchi automaton will consist of pairs where the first component is an \mathcal{ALC} -LTL type and the second component is a set of \mathcal{ALC} -LTL types.

Now, we have to define what it means for a set of \mathcal{ALC} -LTL types to be consistent with respect to rigid names. Beforehand, we need the definition of functions that replace flexible names in a given \mathcal{ALC} -LTL formula.

Definition 4.1. *Let φ be an \mathcal{ALC} -LTL formula with rigid names. The function ρ_x marks all flexible concept and role names occurring in φ by replacing them with a version that has superscript x , i. e. all flexible concept (role) names occurring in φ are replaced by $A^{(x)}$ ($r^{(x)}$) where x is an arbitrary symbol.*

We now give an example for the definition of these renaming functions.

Example 4.2. *Let $\varphi := (C \sqsubseteq D) \mathcal{U} (\bigcirc((C \sqsubseteq \exists r.A) \wedge (a : C)))$ where A is a flexible concept name and r is a flexible role name. We have:*

$$\rho_{fl}(\varphi) = (C \sqsubseteq D) \mathcal{U} (\bigcirc((C \sqsubseteq \exists r^{(fl)}.A^{(fl)}) \wedge (a : C)))$$

This renaming function is used in order to define what it means for a set of types to be consistent with respect to rigid names. What this means is stated in the next definition.

Definition 4.3. *Let φ be an \mathcal{ALC} -LTL formula with rigid names. We define the set $T(\varphi)$ as follows.*

$$T(\varphi) := \{A \subseteq 2^{ty(\varphi)} \mid \bigwedge_{B \in A, \alpha \in B} \rho_B(\alpha) \text{ is consistent} \}$$

As we operate on pairs, we need projection functions in order to access the first and the second component, respectively. These functions are defined hereafter.

Definition 4.4. *Let Ξ be an arbitrary alphabet and let $(\xi_1, \xi_2, \dots, \xi_k)$ be a k -tuple with $\xi_i \in \Xi$ for $1 \leq i \leq k$ for some $k \in \mathbb{N}$. We define projection functions π_j^k for all $1 \leq j \leq k$ as follows:*

$$\pi_j^k((\xi_1, \xi_2, \dots, \xi_k)) := \xi_j$$

Now we are ready for defining for each \mathcal{ALC} -LTL formula φ with rigid names a corresponding generalised non-deterministic Büchi automaton that respects rigid names $\mathcal{G}_\varphi = (Q^\varphi, \Sigma^\varphi, \Delta^\varphi, Q_0^\varphi, \mathcal{F}^\varphi)$. By corresponding we mean the following:

$\sigma_0 \sigma_1 \sigma_2 \dots \in L(\mathcal{G}_\varphi)$ iff there exists an \mathcal{ALC} -LTL interpretation $\mathcal{I} = (\mathcal{I}_i)_{i \geq 0}$ respecting rigid names with $\mathcal{I}, 0 \models \varphi$ and $\tau(\mathcal{I}_0) \tau(\mathcal{I}_1) \dots = \sigma_0 \sigma_1 \dots$

Please note that this condition differs from the condition in the case of \mathcal{ALC} -LTL without rigid names.

Definition 4.5. *Let φ be an \mathcal{ALC} -LTL formula with rigid names. We define the corresponding generalised non-deterministic Büchi automaton $\mathcal{G}_\varphi = (Q^\varphi, \Sigma^\varphi, \Delta^\varphi, Q_0^\varphi, \mathcal{F}^\varphi)$ as follows.*

- $Q^\varphi := ty(\varphi) \times T(\varphi)$
- $\Sigma^\varphi := ty_{ax}(\varphi)$
- $Q_0^\varphi := \{q \in Q^\varphi \mid \varphi \in \pi_1^2(q), \pi_2^2(q) = \{\pi_1^2(q)\}\}$
- $\mathcal{F}^\varphi := \{F_{\psi_1 \mathcal{U} \psi_2} \mid \psi_1 \mathcal{U} \psi_2 \in cl(\varphi)\}$ where
 $F_{\psi_1 \mathcal{U} \psi_2} := \{q \in Q^\varphi \mid \psi_1 \mathcal{U} \psi_2 \notin \pi_1^2(q) \text{ or } \psi_2 \in \pi_1^2(q)\}$
- $\Delta^\varphi \subseteq Q^\varphi \times \Sigma^\varphi \times Q^\varphi$. This relation is defined as follows. $(q, \sigma, q') \in \Delta^\varphi$ iff
 - $\sigma = \pi_1^2(q) \cap ax'(\varphi)$

- For every formula $\bigcirc\psi \in cl(\varphi)$, we have that $\bigcirc\psi \in \pi_1^2(q) \Leftrightarrow \psi \in \pi_1^2(q')$.
- For every formula $\psi_1\mathcal{U}\psi_2 \in cl(\varphi)$, we have that $\psi_1\mathcal{U}\psi_2 \in \pi_1^2(q) \Leftrightarrow (\psi_2 \in \pi_1^2(q) \vee (\psi_1 \in \pi_1^2(q) \wedge \psi_1\mathcal{U}\psi_2 \in \pi_1^2(q')))$.
- $\{\pi_1^2(q)\} \cup \pi_2^2(q) = \pi_2^2(q')$

The next theorem states the soundness and completeness of this construction, i. e. we prove that the generalised non-deterministic Büchi automaton for an \mathcal{ALC} -LTL formula with rigid names accepts all desired sequences of \mathcal{ALC} -types but not more.

Theorem 4.6. *Let φ be an \mathcal{ALC} -LTL formula with rigid names and let \mathcal{G}_φ be the corresponding generalised non-deterministic Büchi automaton. Then the following holds: $\sigma_0\sigma_1\sigma_2\cdots \in L(\mathcal{G}_\varphi)$ iff there exists an \mathcal{ALC} -LTL interpretation $\mathfrak{I} = (\mathcal{I}_i)_{i \geq 0}$ respecting rigid names with $\mathfrak{I}, 0 \models \varphi$ and $\tau(\mathcal{I}_0)\tau(\mathcal{I}_1)\cdots = \sigma_0\sigma_1\cdots$.*

Proof. The proof is very similar to the one established in Theorem 3.10.

“ \Leftarrow ”. Suppose $\mathfrak{I}, 0 \models \varphi$. We define

$$S_0 := (\{\psi \in cl(\varphi) \mid \mathfrak{I}, 0 \models \psi\}, \{\{\psi \in cl(\varphi) \mid \mathfrak{I}, 0 \models \psi\}\})$$

and for $i \geq 1$:

$$S_i := (\{\psi \in cl(\varphi) \mid \mathfrak{I}, i \models \psi\}, \pi_2^2(S_{i-1}) \cup \{\{\psi \in cl(\varphi) \mid \mathfrak{I}, i \models \psi\}\})$$

$S_0S_1S_2\cdots$ is a run of \mathcal{G}_φ on $\tau(\mathcal{I}_0)\tau(\mathcal{I}_1)\tau(\mathcal{I}_2)\cdots$ since:

- $S_i \in Q^\varphi$ for all $i \geq 0$. This holds since S_i is a non-empty subset of $cl(\varphi)$ that is consistent w. r. t. \mathcal{ALC} -LTL, maximal, and thus an \mathcal{ALC} -LTL type.
- $\varphi \in \pi_1^2(S_0)$ and thus $S_0 \in Q_0^\varphi$
- $\tau(\mathcal{I}_i) = \pi_1^2(S_i) \cap ax'(\varphi)$. This holds because of the definition of $\tau(\mathcal{I}_i)$ and Lemma 3.6.
- for all $\bigcirc\psi \in cl(\varphi)$, we have: $\bigcirc\psi \in \pi_1^2(S_i)$ iff $\mathfrak{I}, i \models \bigcirc\psi$ iff $\mathfrak{I}, i+1 \models \psi$ iff $\psi \in \pi_1^2(S_{i+1})$

- for all $\psi_1\mathcal{U}\psi_2 \in cl(\varphi)$, we have:

$$\begin{aligned} & \psi_1\mathcal{U}\psi_2 \in \pi_1^2(S_i) \\ \text{iff } & \mathfrak{I}, i \models \psi_1\mathcal{U}\psi_2 \\ \text{iff } & \mathfrak{I}, i \models \psi_2 \text{ or } \mathfrak{I}, i \models \psi_1 \text{ and } \mathfrak{I}, i+1 \models \psi_1\mathcal{U}\psi_2 \\ \text{iff } & \psi_2 \in \pi_1^2(S_i) \text{ or } \psi_1 \in \pi_1^2(S_i) \text{ and } \psi_1\mathcal{U}\psi_2 \in \pi_1^2(S_{i+1}) \end{aligned}$$

- thus, $(S_i, \tau(\mathcal{I}_i), S_{i+1}) \in \Delta^\varphi$ for all $i \geq 0$

Moreover, $S_0S_1S_2\dots$ is accepting: Suppose that for some $\psi_1\mathcal{U}\psi_2 \in cl(\varphi)$ the set $\{i \in \mathbb{N} \mid S_i \in F_{\psi_1\mathcal{U}\psi_2}\}$ is finite. Then there exists a $k \in \mathbb{N}$ such that $S_i \notin F_{\psi_1\mathcal{U}\psi_2}$ for all $i \geq k$. This means $\psi_1\mathcal{U}\psi_2 \in \pi_1^2(S_i)$ and $\psi_2 \notin \pi_1^2(S_i)$ for all $i \geq k$. Hence, $\mathfrak{I}, k \models \psi_1\mathcal{U}\psi_2$ and $\mathfrak{I}, i \not\models \psi_2$ for all $i \geq k$. This is a contradiction to the semantics of the \mathcal{U} -modality. $\not\downarrow$

“ \Rightarrow ”. Let $\sigma_0\sigma_1\sigma_2\dots \in L(\mathcal{G}_\varphi)$ and let $S_0S_1S_2\dots$ be an accepting run of \mathcal{G}_φ on $\sigma_0\sigma_1\sigma_2\dots$.

We show that then there exists at least one \mathcal{ALC} -LTL interpretation $\mathfrak{I} = (\mathcal{I}_i)_{i \geq 0}$ with $\tau(\mathcal{I}_i) = \sigma_i$ for all $i \geq 0$. We use Lemma 3.8 and the fact that the second component of every state in \mathcal{G}_φ is consistent, i. e. there exists at least one interpretation that satisfies all types of the second component.

The fact that the number of states of \mathcal{G}_φ is finite and the definition of the transition relation yield that there exists some $k \in \mathbb{N}$ such that for all $l \geq k$ we have: $\pi_2^2(S_k) = \pi_2^2(S_l)$. Thus we have:

$$\pi_2^2(S_0) \subseteq \pi_2^2(S_1) \subseteq \pi_2^2(S_2) \subseteq \dots \subseteq \pi_2^2(S_k) = \pi_2^2(S_{k+1}) = \dots$$

Additionally, we know that the Boolean knowledge base

$$\mathcal{B} := \bigwedge_{\alpha \in T, T \in \pi_2^2(S_k)} \rho_T(\alpha)$$

is consistent, i. e. there is an \mathcal{ALC} -interpretation \mathcal{I} that satisfies \mathcal{B} .

We can construct $\mathfrak{I} = (\mathcal{I}_i)_{i \geq 0}$ as follows. Lemma 3.8 implies that there exist some \mathcal{ALC} -interpretations \mathcal{I}'_i for all $i \geq 0$ such that $\tau(\mathcal{I}'_i) = \sigma_i$. The interpretation of a rigid concept name A occurring in φ is defined as $A^{\mathcal{I}'_i} := A^{\mathcal{I}}$ for all $i \geq 0$. The interpretation of a rigid role name is defined analogously. For a flexible concept name we define the interpretation as follows. Let B be a flexible concept name occurring in φ . $B^{\mathcal{I}'_i} := B^{\mathcal{I}'_i}$

for all $i \geq 0$. The interpretation of a flexible role name is defined analogously. Obviously, we have that $\tau(\mathcal{I}_i) = \sigma_i$ for all $i \geq 0$.

It remains to show that $\mathfrak{J}, 0 \models \varphi$.

Since $\varphi \in \pi_1^2(S_0)$ by definition of Q_0^φ , it suffices to show that for all $i \geq 0$ and $\psi \in cl(\varphi)$ we have:

$$\psi \in \pi_1^2(S_i) \text{ iff } \mathfrak{J}, i \models \psi$$

This is proved by induction on the structure of ψ as has been done in the proof of Theorem 3.10.

□

4.1.2 The monitor construction for \mathcal{ALC} -LTL without rigid names

Now we can proceed to construct the monitor for an \mathcal{ALC} -LTL formula φ with rigid names as in the case of \mathcal{ALC} -LTL without rigid names. We will directly define the monitor in the following.

Definition 4.7. *Let φ be an \mathcal{ALC} -LTL formula with rigid names and let \mathcal{G}_φ with $\mathcal{G}_\varphi = (Q^\varphi, \Sigma^\varphi, \Delta^\varphi, Q_0^\varphi, \mathcal{F}^\varphi)$ be the corresponding generalised non-deterministic Büchi automaton.*

The generalised non-deterministic Büchi automaton $\mathcal{G}_{\neg\varphi} = (Q^{\neg\varphi}, \Sigma^{\neg\varphi}, \Delta^{\neg\varphi}, Q_0^{\neg\varphi}, \mathcal{F}^{\neg\varphi})$ is defined analogously.

The finite-state machine $\mathcal{A}_\varphi = (Q, \Sigma, \delta, q_0, \Gamma, \lambda)$ is defined as follows:

- $Q := 2^{Q^\varphi} \times 2^{Q^{\neg\varphi}}$
- $\Sigma := \Sigma^\varphi = \Sigma^{\neg\varphi}$
- $q_0 := (Q_0^\varphi, Q_0^{\neg\varphi})$
- For all $S \subseteq Q^\varphi$, $S' \subseteq Q^{\neg\varphi}$ and $\sigma \in \Sigma$, we have:

$\delta((S, S'), \sigma) := (\hat{S}, \hat{S}')$ where:

- $\hat{S} = \bigcup_{q \in S} \{\hat{q} \in Q^\varphi \mid (q, \sigma, \hat{q}) \in \Delta^\varphi\}$
- $\hat{S}' = \bigcup_{q' \in S'} \{\hat{q}' \in Q^{\neg\varphi} \mid (q', \sigma, \hat{q}') \in \Delta^{\neg\varphi}\}$

- $\Gamma := \mathbb{B}_3$
- $\lambda : Q \rightarrow \mathbb{B}_3$ is defined by

$$\lambda((S, S')) = \begin{cases} 1 & \text{if } L(\mathcal{G}_{\neg\varphi}(q')) = \emptyset \text{ for all } q' \in S' \text{ and} \\ & L(\mathcal{G}_{\varphi}(q)) \neq \emptyset \text{ for some } q \in S \\ 0 & \text{if } L(\mathcal{G}_{\varphi}(q)) = \emptyset \text{ for all } q \in S \text{ and} \\ & L(\mathcal{G}_{\neg\varphi}(q')) \neq \emptyset \text{ for some } q' \in S' \\ ? & \text{otherwise} \end{cases}$$

Again, we will state the correctness theorem for the monitor construction. Beforehand, we need to define what it means for a finite sequence of \mathcal{ALC} -interpretations to respect rigid names.

Definition 4.8. Let $\tilde{\mathcal{I}} = \mathcal{I}_0\mathcal{I}_1\dots\mathcal{I}_k$ for some $k \in \mathbb{N}$ be a finite sequence of \mathcal{ALC} -interpretations. We say that $\tilde{\mathcal{I}}$ respects rigid names iff we have for all rigid concept names A that $A^{\mathcal{I}_i} = A^{\mathcal{I}_j}$ for all $0 \leq i, j \leq k$ and we have for all rigid role names r that $r^{\mathcal{I}_i} = r^{\mathcal{I}_j}$ for all $0 \leq i, j \leq k$.

After giving this definition, we are ready to state the correctness theorem for the monitor construction for \mathcal{ALC} -LTL with rigid names.

Theorem 4.9. Let φ be an \mathcal{ALC} -LTL formula with rigid names and let \mathcal{A}_{φ} be the corresponding monitor with $\mathcal{A}_{\varphi} = (Q, \Sigma, \delta, q_0, \lambda)$. Then, for all $\tilde{\mathcal{I}} = \mathcal{I}'_0\mathcal{I}'_1\dots\mathcal{I}'_k$ that respect rigid names, we have that:

$$f(\varphi, \tilde{\mathcal{I}}) = \lambda(\hat{\delta}(q_0, \tau(\mathcal{I}'_0) \dots \tau(\mathcal{I}'_k)))$$

Proof. Since this theorem is basically the same as Theorem 3.17, we have proved already the correctness of the construction of the monitor.

□

Having proved the correctness of the monitor construction, we will now perform an analysis of the complexity of the construction.

4.2 The complexity for the monitor construction for \mathcal{ALC} -LTL with rigid names

We will now analyse the complexity of the monitor construction for \mathcal{ALC} -LTL with rigid names.

The construction of the monitor starts with the construction of the generalised non-deterministic Büchi automata for the specification and the negated specification. The next lemma states the complexity of the construction of these automata.

Lemma 4.10. *Let φ be an \mathcal{ALC} -LTL formula with rigid names. The construction of the corresponding generalised non-deterministic Büchi automata \mathcal{G}_φ and $\mathcal{G}_{\neg\varphi}$ can be done in 2-EXPTIME.*

Proof. The states in \mathcal{G}_φ are obtained by building the cross product of the types of φ and the set of consistent types $T(\varphi)$; see Definition 4.5. Since all types contain half of the elements of $cl(\varphi)$, we have that the number of states in \mathcal{G}_φ is bounded by $2^{\frac{|cl(\varphi)|}{2}} \cdot 2^{2^{\frac{|cl(\varphi)|}{2}}}$. Hence, the number of states in \mathcal{G}_φ is bounded by $2^{2^{\mathcal{O}(|\varphi|)}}$.

In order to determine the types of φ , one has to check for each type T the consistency of the Boolean knowledge base $\mathcal{B} = \bigwedge_{\alpha \in T \cap ax'(\varphi)} \alpha$. We regard \mathcal{B} as \mathcal{ALC} -LTL formula and check for satisfiability. This can be done in 2-EXPTIME as proved in [2]. Additionally, one has to check for each set of types whether they are consistent, which can obviously be done also in 2-EXPTIME.

The same applies to the construction of $\mathcal{G}_{\neg\varphi}$, which means that the construction of \mathcal{G}_φ and $\mathcal{G}_{\neg\varphi}$ can be done in 2-EXPTIME. □

Using this lemma, we can prove the complexity bound for the whole monitor construction. The next theorem states this complexity bound.

Theorem 4.11. *Let φ be an \mathcal{ALC} -LTL formula with rigid names. The corresponding monitor \mathcal{A}_φ can be constructed in 2-2-EXPTIME.*

Proof. The monitor \mathcal{A}_φ results from building the cross-product of both \mathcal{G}_φ and $\mathcal{G}_{\neg\varphi}$. Additionally, we use the power-set construction in order to render the monitor deterministic; see Definition 4.7. This causes another exponential blow-up of the size of the state space of \mathcal{A}_φ . Thus, the overall construction can be done in 2-2-EXPTIME.

□

In this chapter, we considered runtime verification using the temporal description logic \mathcal{ACC} -LTL with rigid names. We introduced a construction for a monitor for \mathcal{ACC} -LTL with rigid names. We provided formal proof of the correctness of the monitor construction. As we have seen, treating rigid names yields another exponential blow-up. Hence, the construction can be done in 2-2-EXPTIME. This entails limitations for applications in practice. As noted in the previous chapter, further research is needed to determine how the construction behaves in practice.

In the next chapter, we go the extra mile and show how we can additionally treat incomplete knowledge.

5 Runtime verification for \mathcal{ALC} -LTL with rigid names with respect to incomplete knowledge

In this chapter, we will consider runtime verification for \mathcal{ALC} -LTL with rigid names with respect to incomplete knowledge. In Subsection 2.2.1, we introduced assertional knowledge in terms of ABoxes. Since we proceed on the open world assumption, ABoxes represent incomplete knowledge in contrast to interpretations that represent complete knowledge.

As argued in Subsection 2.2.1, it makes sense to consider reasoning with respect to incomplete knowledge, i. e. reasoning with respect to ABoxes, since it is rarely the case that the observer of a system obtains complete knowledge about the state of the system by watching closely. This motivates us to go the extra mile and extend the monitor construction for \mathcal{ALC} -LTL with rigid names in order to treat incomplete knowledge.

First, we will give details about the monitor construction for \mathcal{ALC} -LTL with rigid names that works with incomplete knowledge. Additionally, we prove the correctness of the monitor construction formally. Next, the chapter contains also an analysis of the complexity of the monitor construction.

The chapter finishes with an example of the monitor construction.

5.1 Defining the monitor for an \mathcal{ALC} -LTL formula with rigid names that respects incomplete knowledge

In this section, we are going to define the construction of the monitor for \mathcal{ALC} -LTL with rigid names that respects incomplete knowledge. We already defined the corresponding generalised non-deterministic Büchi automaton for each \mathcal{ALC} -LTL formula with rigid

names; see Definition 4.5. We use this definition for the construction of the monitor. It is constructed almost the same way as in previous cases, but there is one major difference.

For an \mathcal{ALC} -LTL formula with rigid names, the input alphabet of the monitor reasoning with complete knowledge is the set of \mathcal{ALC} -types occurring in φ —namely $ty_{ax}(\varphi)$. Obviously, the corresponding generalised non-deterministic Büchi automata \mathcal{G}_φ and $\mathcal{G}_{\neg\varphi}$ work on the same alphabet. The input alphabet of the monitor differs for reasoning with incomplete knowledge. The monitor for an \mathcal{ALC} -LTL formula with rigid names that works with respect to incomplete knowledge takes ABoxes, instead of \mathcal{ALC} -types, as input.

Since the monitor for \mathcal{ALC} -LTL with rigid names that respects incomplete knowledge takes finite sequences of ABoxes as input, the question arises under what condition it is allowed to take a transition. In other words: What kind of relation between an ABox and an \mathcal{ALC} -type would make a transition possible? There are two possibilities: Let \mathcal{A} be an ABox and let T be an \mathcal{ALC} -type. Either \mathcal{A} is a logical consequence of T ($T \Vdash \mathcal{A}$) or \mathcal{A} and T are consistent, i. e. the Boolean knowledge base $\bigwedge_{\alpha \in T} \alpha \wedge \bigwedge_{\beta \in \mathcal{A}} \beta$ is consistent. When we look at the proof of the correctness theorem, it turns out that the latter is correct.

Now we are ready to define the monitor for an \mathcal{ALC} -LTL formula with rigid names that respects incomplete knowledge.

Definition 5.1. *Let φ be an \mathcal{ALC} -LTL formula with rigid names and let \mathcal{G}_φ with $\mathcal{G}_\varphi = (Q^\varphi, \Sigma^\varphi, \Delta^\varphi, Q_0^\varphi, \mathcal{F}^\varphi)$ be the corresponding generalised non-deterministic Büchi automaton as defined in Definition 4.5.*

The generalised non-deterministic Büchi automation $\mathcal{G}_{\neg\varphi} = (Q^{\neg\varphi}, \Sigma^{\neg\varphi}, \Delta^{\neg\varphi}, Q_0^{\neg\varphi}, \mathcal{F}^{\neg\varphi})$ is defined analogously.

The finite-state machine $\mathcal{A}_\varphi = (Q, \Sigma, \delta, q_0, \Gamma, \lambda)$ is defined as follows:

- $Q := 2^{Q^\varphi} \times 2^{Q^{\neg\varphi}}$
- Σ is the set of all consistent ABoxes over the vocabulary occurring in φ
- $q_0 := (Q_0^\varphi, Q_0^{\neg\varphi})$
- For all $S \subseteq Q^\varphi$, $S' \subseteq Q^{\neg\varphi}$ and $\mathcal{A} \in \Sigma$, we have:
 $\delta((S, S'), \mathcal{A}) := (\hat{S}, \hat{S}')$ with:

$$\begin{aligned}
 - \hat{S} &= \bigcup_{\sigma \in \Sigma^\varphi} \bigcup_{q \in S} \{\hat{q} \in Q^\varphi \mid (q, \sigma, \hat{q}) \in \Delta^\varphi \text{ and } \bigwedge_{\alpha \in \mathcal{A}} \alpha \wedge \bigwedge_{\beta \in \sigma} \beta \text{ is consistent}\} \\
 - \hat{S}' &= \bigcup_{\sigma \in \Sigma^{\neg\varphi}} \bigcup_{q' \in S'} \{\hat{q}' \in Q^{\neg\varphi} \mid (q', \sigma, \hat{q}') \in \Delta^{\neg\varphi} \text{ and } \bigwedge_{\alpha \in \mathcal{A}} \alpha \wedge \bigwedge_{\beta \in \sigma} \beta \text{ is consistent}\}
 \end{aligned}$$

- $\Gamma := \mathbb{B}_3$
- $\lambda : Q \rightarrow \mathbb{B}_3$ is defined by

$$\lambda((S, S')) = \begin{cases} 1 & \text{if } L(\mathcal{G}_{\neg\varphi}(q')) = \emptyset \text{ for all } q' \in S' \text{ and} \\ & L(\mathcal{G}_\varphi(q)) \neq \emptyset \text{ for some } q \in S \\ 0 & \text{if } L(\mathcal{G}_\varphi(q)) = \emptyset \text{ for all } q \in S \text{ and} \\ & L(\mathcal{G}_{\neg\varphi}(q')) \neq \emptyset \text{ for some } q' \in S' \\ ? & \text{otherwise} \end{cases}$$

Note: The alphabet Σ of the finite-state machine \mathcal{A}_φ is infinite, but there is no need to constrain the alphabet. As we will see in the example later on, the infinite alphabet is no problem.

Before formulating and proving the correctness theorem, we are going to prove some lemmas. These lemmas yield the correctness theorem. First, we prove that the monitor outputs 1 if and only if this is the correct answer.

Lemma 5.2. *Let φ be an \mathcal{ALC} -LTL formula with rigid names. Let \mathcal{G}_φ and $\mathcal{G}_{\neg\varphi}$ be the corresponding generalised non-deterministic Büchi automata and let \mathcal{A}_φ be the corresponding monitor that respects incomplete knowledge; see Definition 5.1.*

We have the following for all finite sequences of ABoxes $\mathcal{A}_0\mathcal{A}_1 \dots \mathcal{A}_k$ for $k \in \mathbb{N}$:

$$\begin{aligned}
 & f(\varphi, \tilde{\mathcal{I}}) = 1 \\
 & \text{for all } \tilde{\mathcal{I}} = \mathcal{I}_0\mathcal{I}_1 \dots \mathcal{I}_k \text{ that respect rigid names with } \mathcal{I}_0 \models \mathcal{A}_0, \dots, \mathcal{I}_k \models \mathcal{A}_k \\
 \text{iff } & \lambda(\hat{\delta}(q_0, \mathcal{A}_0\mathcal{A}_1\mathcal{A}_2 \dots \mathcal{A}_k)) = 1
 \end{aligned}$$

Proof. We are using Lemma 3.14 and Lemma 3.15 for the proof.

$$\begin{aligned}
 & f(\varphi, \tilde{\mathcal{I}}) = 1 \text{ for all } \tilde{\mathcal{I}} = \mathcal{I}_0\mathcal{I}_1 \dots \mathcal{I}_k \text{ with } \mathcal{I}_0 \models \mathcal{A}_0, \mathcal{I}_1 \models \mathcal{A}_1, \dots, \mathcal{I}_k \models \mathcal{A}_k \\
 \text{iff } & \text{for all } \tilde{\mathcal{I}} = \mathcal{I}_0\mathcal{I}_1 \dots \mathcal{I}_k \text{ with } \mathcal{I}_0 \models \mathcal{A}_0, \mathcal{I}_1 \models \mathcal{A}_1, \dots, \mathcal{I}_k \models \mathcal{A}_k \\
 & \text{there exists no } q \in Q'_{\neg\varphi}(\tilde{\mathcal{I}}) \text{ such that } L(\mathcal{G}_{\neg\varphi}(q)) \neq \emptyset
 \end{aligned}$$

iff for all $\tilde{\mathcal{I}} = \mathcal{I}_0 \mathcal{I}_1 \dots \mathcal{I}_k$ with $\mathcal{I}_0 \models \mathcal{A}_0, \mathcal{I}_1 \models \mathcal{A}_1, \dots, \mathcal{I}_k \models \mathcal{A}_k$
 for all $q \in Q'_{\neg\varphi}(\tilde{\mathcal{I}})$ we have that $L(\mathcal{G}_{\neg\varphi}(q)) = \emptyset$

iff for all $q \in \bigcup_{\mathcal{I}_0 \models \mathcal{A}_0} \dots \bigcup_{\mathcal{I}_k \models \mathcal{A}_k} Q'_{\neg\varphi}(\mathcal{I}_0 \dots \mathcal{I}_k)$ we have that $L(\mathcal{G}_{\neg\varphi}(q)) = \emptyset$

iff $\lambda((S, S')) = 1$ with:

$$S = \bigcup_{\mathcal{I}_0 \models \mathcal{A}_0} \dots \bigcup_{\mathcal{I}_k \models \mathcal{A}_k} Q'_\varphi(\mathcal{I}_0 \dots \mathcal{I}_k)$$

$$S' = \bigcup_{\mathcal{I}_0 \models \mathcal{A}_0} \dots \bigcup_{\mathcal{I}_k \models \mathcal{A}_k} Q'_{\neg\varphi}(\mathcal{I}_0 \dots \mathcal{I}_k)$$

iff $\lambda((S, S')) = 1$ with:

$$S = \{\hat{q} \in Q^\varphi \mid \exists \mathcal{I}_0 \in \text{Int}_{\mathcal{ALC}}(\varphi) \dots \exists \mathcal{I}_k \in \text{Int}_{\mathcal{ALC}}(\varphi) \cdot \exists q \in Q_0^\varphi \cdot \bigwedge_{i=0}^k \mathcal{I}_i \models \mathcal{A}_i \wedge (q, \tau(\mathcal{I}_0) \dots \tau(\mathcal{I}_k), \hat{q}) \in \hat{\Delta}^\varphi\}$$

$$S' = \{\hat{q}' \in Q^{\neg\varphi} \mid \exists \mathcal{I}_0 \in \text{Int}_{\mathcal{ALC}}(\varphi) \dots \exists \mathcal{I}_k \in \text{Int}_{\mathcal{ALC}}(\varphi) \cdot \exists q' \in Q_0^{\neg\varphi} \cdot \bigwedge_{i=0}^k \mathcal{I}_i \models \mathcal{A}_i \wedge (q', \tau(\mathcal{I}_0) \dots \tau(\mathcal{I}_k), \hat{q}') \in \hat{\Delta}^{\neg\varphi}\}$$

iff $\lambda((S, S')) = 1$ with:

$$S = \{\hat{q} \in Q^\varphi \mid \exists \sigma_0 \in \Sigma^\varphi \dots \exists \sigma_k \in \Sigma^\varphi \cdot \exists q \in Q_0^\varphi \cdot \bigwedge_{i=0}^k \exists \mathcal{I}_i \in \text{Int}_{\mathcal{ALC}}(\varphi) \cdot (\mathcal{I}_i \models \mathcal{A}_i \wedge \sigma_i = \tau(\mathcal{I}_i)) \wedge (q, \sigma_0 \dots \sigma_k, \hat{q}) \in \hat{\Delta}^\varphi\}$$

$$S' = \{\hat{q}' \in Q^{\neg\varphi} \mid \exists \sigma_0 \in \Sigma^{\neg\varphi} \dots \exists \sigma_k \in \Sigma^{\neg\varphi} \cdot \exists q' \in Q_0^{\neg\varphi} \cdot \bigwedge_{i=0}^k \exists \mathcal{I}_i \in \text{Int}_{\mathcal{ALC}}(\varphi) \cdot (\mathcal{I}_i \models \mathcal{A}_i \wedge \sigma_i = \tau(\mathcal{I}_i)) \wedge (q', \sigma_0 \dots \sigma_k, \hat{q}') \in \hat{\Delta}^{\neg\varphi}\}$$

iff $\lambda((S, S')) = 1$ with:

$$S = \{\hat{q} \in Q^\varphi \mid \exists \sigma_0 \in \Sigma^\varphi \dots \exists \sigma_k \in \Sigma^\varphi \cdot \exists q \in Q_0^\varphi \cdot \exists \mathcal{I}_0 \in \text{Int}_{\mathcal{ALC}}(\varphi) \dots \exists \mathcal{I}_k \in \text{Int}_{\mathcal{ALC}}(\varphi) \cdot \bigwedge_{i=0}^k (\mathcal{I}_i \models \mathcal{A}_i \wedge \sigma_i = \tau(\mathcal{I}_i)) \wedge (q, \sigma_0 \dots \sigma_k, \hat{q}) \in \hat{\Delta}^\varphi\}$$

$$\begin{aligned}
 S' = \{ \hat{q}' \in Q^{-\varphi} \mid & \exists \sigma_0 \in \Sigma^{-\varphi} \dots \exists \sigma_k \in \Sigma^{-\varphi} . \exists q' \in Q_0^{-\varphi} . \\
 & \exists \mathcal{I}_0 \in \text{Int}_{\mathcal{ALC}}(\varphi) \dots \exists \mathcal{I}_k \in \text{Int}_{\mathcal{ALC}}(\varphi) . \\
 & \bigwedge_{i=0}^k (\mathcal{I}_i \models \mathcal{A}_i \wedge \sigma_i = \tau(\mathcal{I}_i)) \wedge (q', \sigma_0 \dots \sigma_k, \hat{q}') \in \hat{\Delta}^{-\varphi} \}
 \end{aligned}$$

iff $\lambda((S, S')) = 1$ with:

$$\begin{aligned}
 S = \{ \hat{q} \in Q^\varphi \mid & \exists \sigma_0 \in \Sigma^\varphi \dots \exists \sigma_k \in \Sigma^\varphi . \exists q \in Q_0^\varphi . \\
 & \bigwedge_{i=0}^k (\bigwedge_{\alpha_i \in \mathcal{A}_i} \alpha_i \wedge \bigwedge_{\beta_i \in \sigma_i} \beta_i \text{ is consistent}) \wedge (q, \sigma_0 \dots \sigma_k, \hat{q}) \in \hat{\Delta}^\varphi \}
 \end{aligned}$$

$$\begin{aligned}
 S' = \{ \hat{q}' \in Q^{-\varphi} \mid & \exists \sigma_0 \in \Sigma^{-\varphi} \dots \exists \sigma_k \in \Sigma^{-\varphi} . \exists q' \in Q_0^{-\varphi} . \\
 & \bigwedge_{i=0}^k (\bigwedge_{\alpha_i \in \mathcal{A}_i} \alpha_i \wedge \bigwedge_{\beta_i \in \sigma_i} \beta_i \text{ is consistent}) \wedge (q', \sigma_0 \dots \sigma_k, \hat{q}') \in \hat{\Delta}^{-\varphi} \}
 \end{aligned}$$

iff $\lambda((S, S')) = 1$ with:

$$\begin{aligned}
 S = \bigcup_{\sigma_0 \in \Sigma^\varphi} \dots \bigcup_{\sigma_k \in \Sigma^\varphi} \bigcup_{q \in Q_0^\varphi} \{ \hat{q} \in Q^\varphi \mid & (q, \sigma_0 \dots \sigma_k, \hat{q}) \in \hat{\Delta}^\varphi \\
 & \wedge \bigwedge_{i=0}^k (\bigwedge_{\alpha_i \in \mathcal{A}_i} \alpha_i \wedge \bigwedge_{\beta_i \in \sigma_i} \beta_i \text{ is consistent}) \}
 \end{aligned}$$

$$\begin{aligned}
 S' = \bigcup_{\sigma_0 \in \Sigma^{-\varphi}} \dots \bigcup_{\sigma_k \in \Sigma^{-\varphi}} \bigcup_{q' \in Q_0^{-\varphi}} \{ \hat{q}' \in Q^{-\varphi} \mid & (q', \sigma_0 \dots \sigma_k, \hat{q}') \in \hat{\Delta}^{-\varphi} \\
 & \wedge \bigwedge_{i=0}^k (\bigwedge_{\alpha_i \in \mathcal{A}_i} \alpha_i \wedge \bigwedge_{\beta_i \in \sigma_i} \beta_i \text{ is consistent}) \}
 \end{aligned}$$

iff $\lambda(\hat{\delta}((Q_0^\varphi, Q_0^{-\varphi}), \mathcal{A}_0 \mathcal{A}_1 \dots \mathcal{A}_k)) = 1$

iff $\lambda(\hat{\delta}(q_0, \mathcal{A}_0 \mathcal{A}_1 \dots \mathcal{A}_k)) = 1$

□

The next lemma is similar to Lemma 5.2. It covers the case in which the output of the monitor has to be 0.

Lemma 5.3. *Let φ be an \mathcal{ALC} -LTL formula with rigid names. Let \mathcal{G}_φ and $\mathcal{G}_{-\varphi}$ be the corresponding generalised non-deterministic Büchi automata and let \mathcal{A}_φ be the corresponding monitor that respects incomplete knowledge; see Definition 5.1.*

We have the following for all finite sequences of ABoxes $\mathcal{A}_0 \mathcal{A}_1 \dots \mathcal{A}_k$ for $k \in \mathbb{N}$:

$$f(\varphi, \tilde{\mathcal{I}}) = 0$$

for all $\tilde{\mathcal{I}} = \mathcal{I}_0 \mathcal{I}_1 \dots \mathcal{I}_k$ that respect rigid names with $\mathcal{I}_0 \models \mathcal{A}_0, \dots, \mathcal{I}_k \models \mathcal{A}_k$

$$\text{iff } \lambda(\hat{\delta}(q_0, \mathcal{A}_0 \mathcal{A}_1 \mathcal{A}_2 \dots \mathcal{A}_k)) = 0$$

Proof. This lemma is proved analogously to Lemma 5.2, again using again Lemma 3.14 and Lemma 3.15.

$$f(\varphi, \tilde{\mathcal{I}}) = 0 \text{ for all } \tilde{\mathcal{I}} = \mathcal{I}_0 \mathcal{I}_1 \dots \mathcal{I}_k \text{ with } \mathcal{I}_0 \models \mathcal{A}_0, \mathcal{I}_1 \models \mathcal{A}_1, \dots, \mathcal{I}_k \models \mathcal{A}_k$$

iff for all $\tilde{\mathcal{I}} = \mathcal{I}_0 \mathcal{I}_1 \dots \mathcal{I}_k$ with $\mathcal{I}_0 \models \mathcal{A}_0, \mathcal{I}_1 \models \mathcal{A}_1, \dots, \mathcal{I}_k \models \mathcal{A}_k$
there exists no $q \in Q'_\varphi(\tilde{\mathcal{I}})$ such that $L(\mathcal{G}_\varphi(q)) \neq \emptyset$

iff for all $\tilde{\mathcal{I}} = \mathcal{I}_0 \mathcal{I}_1 \dots \mathcal{I}_k$ with $\mathcal{I}_0 \models \mathcal{A}_0, \mathcal{I}_1 \models \mathcal{A}_1, \dots, \mathcal{I}_k \models \mathcal{A}_k$
for all $q \in Q'_\varphi(\tilde{\mathcal{I}})$ we have that $L(\mathcal{G}_\varphi(q)) = \emptyset$

iff for all $q \in \bigcup_{\mathcal{I}_0 \models \mathcal{A}_0} \dots \bigcup_{\mathcal{I}_k \models \mathcal{A}_k} Q'_\varphi(\mathcal{I}_0 \dots \mathcal{I}_k)$ we have that $L(\mathcal{G}_\varphi(q)) = \emptyset$

iff $\lambda((S, S')) = 0$ with:

$$S = \bigcup_{\mathcal{I}_0 \models \mathcal{A}_0} \dots \bigcup_{\mathcal{I}_k \models \mathcal{A}_k} Q'_\varphi(\mathcal{I}_0 \dots \mathcal{I}_k)$$

$$S' = \bigcup_{\mathcal{I}_0 \models \mathcal{A}_0} \dots \bigcup_{\mathcal{I}_k \models \mathcal{A}_k} Q'_{\neg\varphi}(\mathcal{I}_0 \dots \mathcal{I}_k)$$

iff $\lambda((S, S')) = 0$ with:

$$S = \{\hat{q} \in Q^\varphi \mid \exists \mathcal{I}_0 \in \text{Int}_{\mathcal{ALC}}(\varphi) \dots \exists \mathcal{I}_k \in \text{Int}_{\mathcal{ALC}}(\varphi) \cdot \exists q \in Q_0^\varphi \cdot \bigwedge_{i=0}^k \mathcal{I}_i \models \mathcal{A}_i \wedge (q, \tau(\mathcal{I}_0) \dots \tau(\mathcal{I}_k), \hat{q}) \in \hat{\Delta}^\varphi\}$$

$$S' = \{\hat{q}' \in Q^{\neg\varphi} \mid \exists \mathcal{I}_0 \in \text{Int}_{\mathcal{ALC}}(\varphi) \dots \exists \mathcal{I}_k \in \text{Int}_{\mathcal{ALC}}(\varphi) \cdot \exists q' \in Q_0^{\neg\varphi} \cdot \bigwedge_{i=0}^k \mathcal{I}_i \models \mathcal{A}_i \wedge (q', \tau(\mathcal{I}_0) \dots \tau(\mathcal{I}_k), \hat{q}') \in \hat{\Delta}^{\neg\varphi}\}$$

iff $\lambda((S, S')) = 0$ with:

$$S = \{\hat{q} \in Q^\varphi \mid \exists \sigma_0 \in \Sigma^\varphi \dots \exists \sigma_k \in \Sigma^\varphi \cdot \exists q \in Q_0^\varphi \cdot \bigwedge_{i=0}^k \exists \mathcal{I}_i \in \text{Int}_{\mathcal{ALC}}(\varphi) \cdot (\mathcal{I}_i \models \mathcal{A}_i \wedge \sigma_i = \tau(\mathcal{I}_i)) \wedge (q, \sigma_0 \dots \sigma_k, \hat{q}) \in \hat{\Delta}^\varphi\}$$

$$S' = \{\hat{q}' \in Q^{\neg\varphi} \mid \exists \sigma_0 \in \Sigma^{\neg\varphi} \dots \exists \sigma_k \in \Sigma^{\neg\varphi} . \exists q' \in Q_0^{\neg\varphi} . \\ \bigwedge_{i=0}^k \exists \mathcal{I}_i \in \text{Int}_{\mathcal{ALC}}(\varphi) . (\mathcal{I}_i \models \mathcal{A}_i \wedge \sigma_i = \tau(\mathcal{I}_i)) \\ \wedge (q', \sigma_0 \dots \sigma_k, \hat{q}') \in \hat{\Delta}^{\neg\varphi}\}$$

iff $\lambda((S, S')) = 0$ with:

$$S = \{\hat{q} \in Q^\varphi \mid \exists \sigma_0 \in \Sigma^\varphi \dots \exists \sigma_k \in \Sigma^\varphi . \exists q \in Q_0^\varphi . \\ \exists \mathcal{I}_0 \in \text{Int}_{\mathcal{ALC}}(\varphi) \dots \exists \mathcal{I}_k \in \text{Int}_{\mathcal{ALC}}(\varphi) . \\ \bigwedge_{i=0}^k (\mathcal{I}_i \models \mathcal{A}_i \wedge \sigma_i = \tau(\mathcal{I}_i)) \wedge (q, \sigma_0 \dots \sigma_k, \hat{q}) \in \hat{\Delta}^\varphi\}$$

$$S' = \{\hat{q}' \in Q^{\neg\varphi} \mid \exists \sigma_0 \in \Sigma^{\neg\varphi} \dots \exists \sigma_k \in \Sigma^{\neg\varphi} . \exists q' \in Q_0^{\neg\varphi} . \\ \exists \mathcal{I}_0 \in \text{Int}_{\mathcal{ALC}}(\varphi) \dots \exists \mathcal{I}_k \in \text{Int}_{\mathcal{ALC}}(\varphi) . \\ \bigwedge_{i=0}^k (\mathcal{I}_i \models \mathcal{A}_i \wedge \sigma_i = \tau(\mathcal{I}_i)) \wedge (q', \sigma_0 \dots \sigma_k, \hat{q}') \in \hat{\Delta}^{\neg\varphi}\}$$

iff $\lambda((S, S')) = 0$ with:

$$S = \{\hat{q} \in Q^\varphi \mid \exists \sigma_0 \in \Sigma^\varphi \dots \exists \sigma_k \in \Sigma^\varphi . \exists q \in Q_0^\varphi . \\ \bigwedge_{i=0}^k (\bigwedge_{\alpha_i \in \mathcal{A}_i} \alpha_i \wedge \bigwedge_{\beta_i \in \sigma_i} \beta_i \text{ is consistent}) \wedge (q, \sigma_0 \dots \sigma_k, \hat{q}) \in \hat{\Delta}^\varphi\}$$

$$S' = \{\hat{q}' \in Q^{\neg\varphi} \mid \exists \sigma_0 \in \Sigma^{\neg\varphi} \dots \exists \sigma_k \in \Sigma^{\neg\varphi} . \exists q' \in Q_0^{\neg\varphi} . \\ \bigwedge_{i=0}^k (\bigwedge_{\alpha_i \in \mathcal{A}_i} \alpha_i \wedge \bigwedge_{\beta_i \in \sigma_i} \beta_i \text{ is consistent}) \wedge (q', \sigma_0 \dots \sigma_k, \hat{q}') \in \hat{\Delta}^{\neg\varphi}\}$$

iff $\lambda((S, S')) = 0$ with:

$$S = \bigcup_{\sigma_0 \in \Sigma^\varphi} \dots \bigcup_{\sigma_k \in \Sigma^\varphi} \bigcup_{q \in Q_0^\varphi} \{\hat{q} \in Q^\varphi \mid (q, \sigma_0 \dots \sigma_k, \hat{q}) \in \hat{\Delta}^\varphi \\ \wedge \bigwedge_{i=0}^k (\bigwedge_{\alpha_i \in \mathcal{A}_i} \alpha_i \wedge \bigwedge_{\beta_i \in \sigma_i} \beta_i \text{ is consistent})\}$$

$$S' = \bigcup_{\sigma_0 \in \Sigma^{\neg\varphi}} \dots \bigcup_{\sigma_k \in \Sigma^{\neg\varphi}} \bigcup_{q' \in Q_0^{\neg\varphi}} \{\hat{q}' \in Q^{\neg\varphi} \mid (q', \sigma_0 \dots \sigma_k, \hat{q}') \in \hat{\Delta}^{\neg\varphi} \\ \wedge \bigwedge_{i=0}^k (\bigwedge_{\alpha_i \in \mathcal{A}_i} \alpha_i \wedge \bigwedge_{\beta_i \in \sigma_i} \beta_i \text{ is consistent})\}$$

iff $\lambda(\hat{\delta}((Q_0^\varphi, Q_0^{\neg\varphi}), \mathcal{A}_0 \mathcal{A}_1 \dots \mathcal{A}_k)) = 0$

iff $\lambda(\hat{\delta}(q_0, \mathcal{A}_0 \mathcal{A}_1 \dots \mathcal{A}_k)) = 0$

□

These two lemmas yield the correctness theorem for the monitor construction.

Theorem 5.4. *Let φ be an \mathcal{ALC} -LTL formula with rigid names. Let \mathcal{G}_φ and $\mathcal{G}_{\neg\varphi}$ be the corresponding generalised non-deterministic Büchi automata and let \mathcal{A}_φ be the corresponding monitor that works with incomplete knowledge.*

We have the following for all finite sequences of ABoxes $\mathcal{A}_0\mathcal{A}_1\dots\mathcal{A}_k$ for $k \in \mathbb{N}$:

- $f(\varphi, \tilde{\mathcal{I}}) = 1$
for all $\tilde{\mathcal{I}} = \mathcal{I}_0\mathcal{I}_1\dots\mathcal{I}_k$ that respect rigid names with $\mathcal{I}_0 \models \mathcal{A}_0, \dots, \mathcal{I}_k \models \mathcal{A}_k$
iff $\lambda(\hat{\delta}(q_0, \mathcal{A}_0\mathcal{A}_1\mathcal{A}_2\dots\mathcal{A}_k)) = 1$
- $f(\varphi, \tilde{\mathcal{I}}) = 0$
for all $\tilde{\mathcal{I}} = \mathcal{I}_0\mathcal{I}_1\dots\mathcal{I}_k$ that respect rigid names with $\mathcal{I}_0 \models \mathcal{A}_0, \dots, \mathcal{I}_k \models \mathcal{A}_k$
iff $\lambda(\hat{\delta}(q_0, \mathcal{A}_0\mathcal{A}_1\mathcal{A}_2\dots\mathcal{A}_k)) = 0$
- $f(\varphi, \tilde{\mathcal{I}}) \neq 1$
for all $\tilde{\mathcal{I}} = \mathcal{I}_0\mathcal{I}_1\dots\mathcal{I}_k$ that respect rigid names with $\mathcal{I}_0 \models \mathcal{A}_0, \dots, \mathcal{I}_k \models \mathcal{A}_k$ and
 $f(\varphi, \tilde{\mathcal{I}}) \neq 0$
for all $\tilde{\mathcal{I}} = \mathcal{I}_0\mathcal{I}_1\dots\mathcal{I}_k$ that respect rigid names with $\mathcal{I}_0 \models \mathcal{A}_0, \dots, \mathcal{I}_k \models \mathcal{A}_k$
iff $\lambda(\hat{\delta}(q_0, \mathcal{A}_0\mathcal{A}_1\mathcal{A}_2\dots\mathcal{A}_k)) = ?$

Proof. The first item was shown in Lemma 5.2 and the second item was shown in Lemma 5.3. These two items imply the third one.

□

Having proved the correctness theorem, we will now take a look at the complexity of the construction.

5.2 The complexity for the monitor construction for \mathcal{ALC} -LTL with rigid names with respect to incomplete knowledge

We will now analyse the complexity of the monitor construction for \mathcal{ALC} -LTL with rigid names w. r. t. incomplete knowledge. Since the alphabet of the monitor is infinite, the monitor, too, is infinite. The state space, however, is finite. Thus, we are considering only the complexity w. r. t. the number of states here.

The construction of the monitor starts with the construction of the generalised non-deterministic Büchi automata for the specification and the negated specification. Lemma 4.10 states that this can be done in 2-EXPTIME. Using this lemma, we can prove the complexity bound for the state space of the whole monitor construction. The next theorem states this complexity bound.

Theorem 5.5. *Let φ be an \mathcal{ALC} -LTL formula with rigid names. The corresponding monitor \mathcal{A}_φ that works with incomplete knowledge can be constructed in 2-2-EXPTIME.*

Proof. The state space of the monitor \mathcal{A}_φ results from building the cross-product of both \mathcal{G}_φ and $\mathcal{G}_{\neg\varphi}$. Additionally, we use the power-set construction in order to make the monitor deterministic; see Definition 5.1. This causes another exponential blow-up of the size of the state space of \mathcal{A}_φ . Thus, the overall construction can be done in 2-2-EXPTIME. □

Interestingly, the complexity for monitoring an \mathcal{ALC} -LTL formula with rigid names with respect to incomplete knowledge is the same as for \mathcal{ALC} -LTL with rigid names—at least if we consider the complexity only with respect to the number of states.

This chapter finishes with an example of the monitor construction.

5.3 An example of the monitor construction

In this section, we are going to give an example of the complete construction of the monitor for an \mathcal{ALC} -LTL formula with rigid names that works with incomplete knowledge.

We will again consider processes working with a critical section. In our example, we have two process types—privileged processes and unprivileged processes. One requirement is that instances of the process type of unprivileged processes never be in the critical section. An additional requirement is that instances of the process type of privileged processes reach the critical section infinitely often.

5.3.1 The \mathcal{ALC} -LTL formula for the specification

First, we are going to look at the set of concept names N_C . In the \mathcal{ALC} -LTL formula for the specification, we want to characterise processes of the privileged type and processes of the unprivileged type as well as the critical section. Take N_C to be the following:

$$N_C := \{ProcType_Privileged, ProcType_Unprivileged, Critical_Section\}$$

In our formalisation, we need some role name that optionally connects instances of a certain process type with the critical section. Intuitively, this connection exists iff the process is in the critical section, i. e. $N_R := \{is_in\}$.

We need to ensure that every process is *either* of the privileged type *or* of the unprivileged type, i. e. for all models $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, we have that there are no elements of $\Delta^{\mathcal{I}}$ which are in $ProcType_Privileged^{\mathcal{I}}$ and $ProcType_Unprivileged^{\mathcal{I}}$. This has to be satisfied at all times. Hence, we use the following formula to enforce the disjointness of these two sets:

$$\Box(ProcType_Privileged \sqcap ProcType_Unprivileged \sqsubseteq \perp)$$

Depending on the application, the concept name $ProcType_Privileged$ and also the concept name $ProcType_Unprivileged$ could be rigid names. Obviously, this means that a process cannot change its type. This implies that the monitor need to know all processes at the beginning, i. e. there are no processes started thereafter. This does not make sense in every application, but in this example we proceed on the assumption that we know all processes at the beginning.

Next, we want to formalise the fact that there are never instances of the unprivileged process type within the critical section. This is easily ensured by the following formula:

$$\Box(ProcType_Unprivileged \sqcap \exists is_in. Critical_Section \sqsubseteq \perp)$$

Finally, we want to ensure that instances of the privileged process type reach the critical section infinitely often. The following \mathcal{ALC} -LTL formula ensures this fact.

$$\Box \Diamond \neg (ProcType_Privileged \sqcap \exists is_in. Critical_Section \sqsubseteq \perp)$$

Let φ_{SPEC} denote the \mathcal{ALC} -LTL formula that formalises the complete specification above, i. e. φ_{SPEC} is the conjunction of all the formulas given above.

$$\begin{aligned} \varphi_{SPEC} &:= \Box(\text{ProcType_Privileged} \sqcap \text{ProcType_Unprivileged} \sqsubseteq \perp) \\ &\wedge \Box(\text{ProcType_Unprivileged} \sqcap \exists \text{is.in.Critical_Section} \sqsubseteq \perp) \\ &\wedge \Box\Diamond\neg(\text{ProcType_Privileged} \sqcap \exists \text{is.in.Critical_Section} \sqsubseteq \perp) \end{aligned}$$

This formula is rather complex, which means that also the construction of the generalised non-deterministic Büchi automaton for φ_{SPEC} is rather extensive. One could use tools that implement special optimisations of the construction. In the easiest case, in which we do not allow rigid names, a tool like LTL 2 BA [10] could be used in order to get small automata.

In our example, we simplify the formula by formalising only a part of the specification—just to keep things simple. First, we replace the abbreviations in φ_{SPEC} . We take

$$\varphi := \neg((\top \sqsubseteq \top) \mathcal{U} \neg(\text{ProcType_Unprivileged} \sqcap \exists \text{is.in.Critical_Section} \sqsubseteq \perp))$$

The next step for the construction of the monitor is computing the generalised non-deterministic Büchi automaton for φ .

5.3.2 The generalised non-deterministic Büchi automata for the \mathcal{ALC} -LTL formulas φ and $\neg\varphi$

We construct the generalised non-deterministic Büchi automaton for the \mathcal{ALC} -LTL formula φ using Definition 4.5.

First, we need to compute the sets $ax(\varphi)$, $ax'(\varphi)$ and $cl(\varphi)$.

The set of axioms within φ is the following:

$$ax(\varphi) = \{ \top \sqsubseteq \top, \text{ProcType_Unprivileged} \sqcap \exists \text{is.in.Critical_Section} \sqsubseteq \perp \}$$

The set $ax'(\varphi)$ contains all the elements of $ax(\varphi)$ and additionally a negated version of every axiom in $ax(\varphi)$.

Thus, the set $ax'(\varphi)$ is the following:

$$ax'(\varphi) = \{ \top \sqsubseteq \top, \neg(\top \sqsubseteq \top), \\ ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp, \\ \neg(ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp) \}$$

Next, we compute the closure of φ .

$$cl(\varphi) = \{ \top \sqsubseteq \top, \neg(\top \sqsubseteq \top), \\ ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp, \\ \neg(ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp), \\ (\top \sqsubseteq \top) \mathcal{U} \neg(ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp), \\ \neg((\top \sqsubseteq \top) \mathcal{U} \neg(ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp)) \}$$

For the construction of the generalised non-deterministic Büchi automaton, we need also the sets $ty(\varphi)$, $T(\varphi)$ and $ty_{ax}(\varphi)$ —see Definition 4.5.

Next, we compute the set of all \mathcal{ALC} -LTL types for φ . An \mathcal{ALC} -LTL type must meet the conditions of Definition 3.4. We have the following types:

$$T_1 = \{ \top \sqsubseteq \top, \\ ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp, \\ (\top \sqsubseteq \top) \mathcal{U} \neg(ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp) \}$$

$$T_2 = \{ \top \sqsubseteq \top, \\ ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp, \\ \neg((\top \sqsubseteq \top) \mathcal{U} \neg(ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp)) \}$$

$$T_3 = \{ \top \sqsubseteq \top, \\ \neg(ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp), \\ (\top \sqsubseteq \top) \mathcal{U} \neg(ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp) \}$$

Hence, the set $ty(\varphi)$ is defined as follows.

$$ty(\varphi) = \{T_1, T_2, T_3\}$$

Since we respect rigid names, we have to compute the set $T(\varphi)$ as well. This set contains sets of types that are consistent with respect to renaming—see Definition 4.3.

$$\begin{aligned} T(\varphi) = \{ & \emptyset, \{T_1\}, \{T_2\}, \{T_3\}, \\ & \{T_1, T_2\}, \{T_1, T_3\}, \{T_2, T_3\}, \\ & \{T_1, T_2, T_3\} \} = 2^{ty(\varphi)} \end{aligned}$$

The set of \mathcal{ALC} -types $ty_{ax}(\varphi)$ is the following.

$$\begin{aligned} ty_{ax}(\varphi) = \{ & \{\top \sqsubseteq \top, ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp\}, \\ & \{\top \sqsubseteq \top, \neg(ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp)\} \} \end{aligned}$$

This set is the alphabet of the generalised non-deterministic Büchi automata. We will abbreviate the symbols as follows:

$$\begin{aligned} t_1 &:= \{\top \sqsubseteq \top, ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp\} \\ t_2 &:= \{\top \sqsubseteq \top, \neg(ProcType_Unprivileged \sqcap \exists is_in.Critical_Section \sqsubseteq \perp)\} \end{aligned}$$

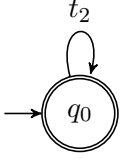
Using Definition 4.5, we can now proceed to construct the generalised non-deterministic Büchi automaton G_φ for φ . In this example, we are constructing only the reachable part of the automaton, i. e. we chop off all states that are not reachable from an initial state and their transitions, respectively. This can be regarded as a useful optimisation.

The initial state of G_φ is $(T_2, \{T_2\})$, since T_2 is the only type containing the formula φ . For this state— $(T_2, \{T_2\})$ —we have only outgoing transitions labelled with t_1 , because $t_1 = T_2 \cap ax'(\varphi)$. The only state we can reach from $(T_2, \{T_2\})$ via t_1 is the state itself; see the conditions of the transition relation in Definition 4.5.

Hence, we have the following for $G_\varphi = (Q^\varphi, \Sigma^\varphi, \Delta^\varphi, Q_0^\varphi, \mathcal{F}^\varphi)$:

- $Q^\varphi = \{q_0\}$ where $q_0 := (T_2, \{T_2\})$
- $\Sigma^\varphi = ty_{ax}(\varphi)$
- $Q_0^\varphi = \{q_0\}$
- $\mathcal{F}^\varphi = \{\{q_0\}\}$
- $\Delta^\varphi = \{(q_0, t_1, q_0)\}$

The visualisation of \mathcal{G}_φ looks as follows:



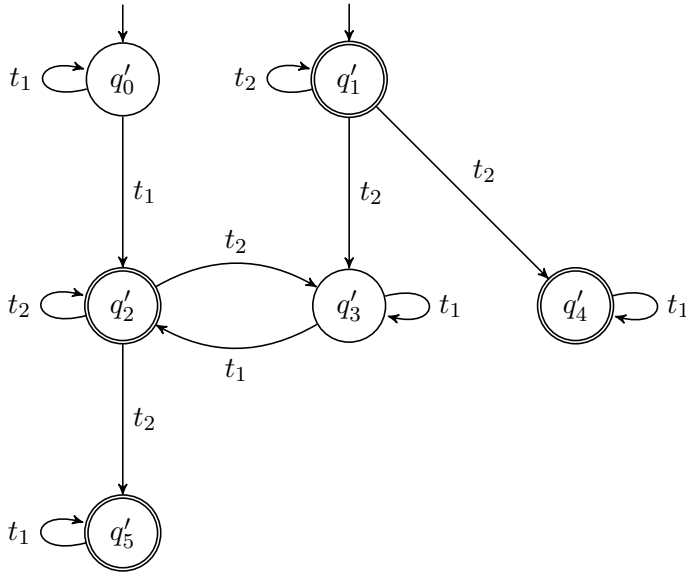
Next, we will construct the reachable part of the generalised non-deterministic Büchi automaton $\mathcal{G}_{\neg\varphi}$ for the \mathcal{ALC} -LTL formula $\neg\varphi$.

We have for $\mathcal{G}_{\neg\varphi} = (Q^{\neg\varphi}, \Sigma^{\neg\varphi}, \Delta^{\neg\varphi}, Q_0^{\neg\varphi}, \mathcal{F}^{\neg\varphi})$ the following:

- $Q^{\neg\varphi} = \{q'_0, q'_1, q'_2, q'_3, q'_4, q'_5\}$ where
 - $q'_0 := (T_1, \{T_1\})$,
 - $q'_1 := (T_3, \{T_3\})$,
 - $q'_2 := (T_3, \{T_1, T_3\})$,
 - $q'_3 := (T_1, \{T_1, T_3\})$,
 - $q'_4 := (T_2, \{T_2, T_3\})$ and
 - $q'_5 := (T_2, \{T_1, T_2, T_3\})$
- $\Sigma^{\neg\varphi} = ty_{ax}(\varphi)$
- $Q_0^{\neg\varphi} = \{q'_0, q'_1\}$
- $\mathcal{F}^{\neg\varphi} = \{\{q'_1, q'_2, q'_4, q'_5\}\}$
- For the transition relation, we have:

$$\Delta^{\neg\varphi} = \{ \begin{array}{l} (q'_0, t_1, q'_0), (q'_0, t_1, q'_2), \\ (q'_1, t_2, q'_1), (q'_1, t_2, q'_3), (q'_1, t_2, q'_4), \\ (q'_2, t_2, q'_2), (q'_2, t_2, q'_3), (q'_2, t_2, q'_5), \\ (q'_3, t_1, q'_2), (q'_3, t_1, q'_3), \\ (q'_4, t_1, q'_4), \\ (q'_5, t_1, q'_5) \end{array} \}$$

The visualisation of $\mathcal{G}_{\neg\varphi}$ looks as follows:



5.3.3 The monitor for the \mathcal{ALC} -LTL formula φ

Having computed the generalised non-deterministic Büchi automata \mathcal{G}_φ and $\mathcal{G}_{\neg\varphi}$, we can now proceed to construct the monitor for φ in terms of a finite-state machine with output. This is done directly, as indicated in Definition 5.1. In this example, we construct only the reachable part of the finite-state machine.

We have the following: $\mathcal{A}_\varphi = (Q, \Sigma, \delta, q_0, \Gamma, \lambda)$ where

- $Q = \{\bar{q}_i \mid 0 \leq i \leq 12\}$ where

$$\bar{q}_0 = (\{q_0\}, \{q'_0, q'_1\})$$

$$\bar{q}_2 = (\emptyset, \{q'_1, q'_3, q'_4\})$$

$$\bar{q}_4 = (\emptyset, \{q'_2, q'_3, q'_5\})$$

$$\bar{q}_6 = (\emptyset, \{q'_2, q'_3, q'_4\})$$

$$\bar{q}_8 = (\{q_0\}, \{q'_0, q'_2, q'_3, q'_4\})$$

$$\bar{q}_{10} = (\{q_0\}, \{q'_0, q'_1, q'_2, q'_3, q'_4, q'_5\})$$

$$\bar{q}_{12} = (\{q_0\}, \{q'_0, q'_2, q'_3, q'_4, q'_5\})$$

$$\bar{q}_1 = (\{q_0\}, \{q'_0, q'_2\})$$

$$\bar{q}_3 = (\{q_0\}, \{q'_0, q'_1, q'_2, q'_3, q'_4\})$$

$$\bar{q}_5 = (\{q_0\}, \{q'_0, q'_2, q'_3, q'_5\})$$

$$\bar{q}_7 = (\emptyset, \{q'_1, q'_2, q'_3, q'_4\})$$

$$\bar{q}_9 = (\emptyset, \{q'_1, q'_2, q'_3, q'_4, q'_5\})$$

$$\bar{q}_{11} = (\emptyset, \{q'_2, q'_3, q'_4, q'_5\})$$

- $\Sigma = \{\mathcal{A} \mid \mathcal{A} \text{ is a consistent ABox over the vocabulary occurring in } \varphi\}$

and there exist A_1, A_2 and A_3 with $\emptyset \neq A_1 \subseteq \Sigma$, $\emptyset \neq A_2 \subseteq \Sigma$ and $\emptyset \neq A_3 \subseteq \Sigma$:

$$\begin{aligned} A_1 &= \{ \mathcal{A} \in \Sigma \mid \bigwedge_{\alpha \in \mathcal{A}} \alpha \wedge \bigwedge_{\beta \in t_1} \beta \text{ is consistent and } \bigwedge_{\alpha \in \mathcal{A}} \alpha \wedge \bigwedge_{\beta \in t_2} \beta \text{ is inconsistent} \} \\ A_2 &= \{ \mathcal{A} \in \Sigma \mid \bigwedge_{\alpha \in \mathcal{A}} \alpha \wedge \bigwedge_{\beta \in t_1} \beta \text{ is inconsistent and } \bigwedge_{\alpha \in \mathcal{A}} \alpha \wedge \bigwedge_{\beta \in t_2} \beta \text{ is consistent} \} \\ A_3 &= \{ \mathcal{A} \in \Sigma \mid \bigwedge_{\alpha \in \mathcal{A}} \alpha \wedge \bigwedge_{\beta \in t_1} \beta \text{ is consistent and } \bigwedge_{\alpha \in \mathcal{A}} \alpha \wedge \bigwedge_{\beta \in t_2} \beta \text{ is consistent} \} \end{aligned}$$

- the transition function $\delta : Q \times \Sigma \rightarrow Q$ is defined as follows.

For all $\mathcal{A} \in A_1$, we have:

$$\begin{aligned} \delta(\overline{q_0}, \mathcal{A}) &= \overline{q_1} & \delta(\overline{q_1}, \mathcal{A}) &= \overline{q_1} & \delta(\overline{q_2}, \mathcal{A}) &= \overline{q_6} & \delta(\overline{q_3}, \mathcal{A}) &= \overline{q_8} \\ \delta(\overline{q_4}, \mathcal{A}) &= \overline{q_4} & \delta(\overline{q_5}, \mathcal{A}) &= \overline{q_5} & \delta(\overline{q_6}, \mathcal{A}) &= \overline{q_6} & \delta(\overline{q_7}, \mathcal{A}) &= \overline{q_6} \\ \delta(\overline{q_8}, \mathcal{A}) &= \overline{q_8} & \delta(\overline{q_9}, \mathcal{A}) &= \overline{q_{11}} & \delta(\overline{q_{10}}, \mathcal{A}) &= \overline{q_{12}} & \delta(\overline{q_{11}}, \mathcal{A}) &= \overline{q_{11}} \\ \delta(\overline{q_{12}}, \mathcal{A}) &= \overline{q_{12}} & & & & & & \end{aligned}$$

For all $\mathcal{A} \in A_2$, we have:

$$\begin{aligned} \delta(\overline{q_0}, \mathcal{A}) &= \overline{q_2} & \delta(\overline{q_1}, \mathcal{A}) &= \overline{q_4} & \delta(\overline{q_2}, \mathcal{A}) &= \overline{q_2} & \delta(\overline{q_3}, \mathcal{A}) &= \overline{q_9} \\ \delta(\overline{q_4}, \mathcal{A}) &= \overline{q_4} & \delta(\overline{q_5}, \mathcal{A}) &= \overline{q_4} & \delta(\overline{q_6}, \mathcal{A}) &= \overline{q_4} & \delta(\overline{q_7}, \mathcal{A}) &= \overline{q_9} \\ \delta(\overline{q_8}, \mathcal{A}) &= \overline{q_4} & \delta(\overline{q_9}, \mathcal{A}) &= \overline{q_9} & \delta(\overline{q_{10}}, \mathcal{A}) &= \overline{q_9} & \delta(\overline{q_{11}}, \mathcal{A}) &= \overline{q_4} \\ \delta(\overline{q_{12}}, \mathcal{A}) &= \overline{q_4} & & & & & & \end{aligned}$$

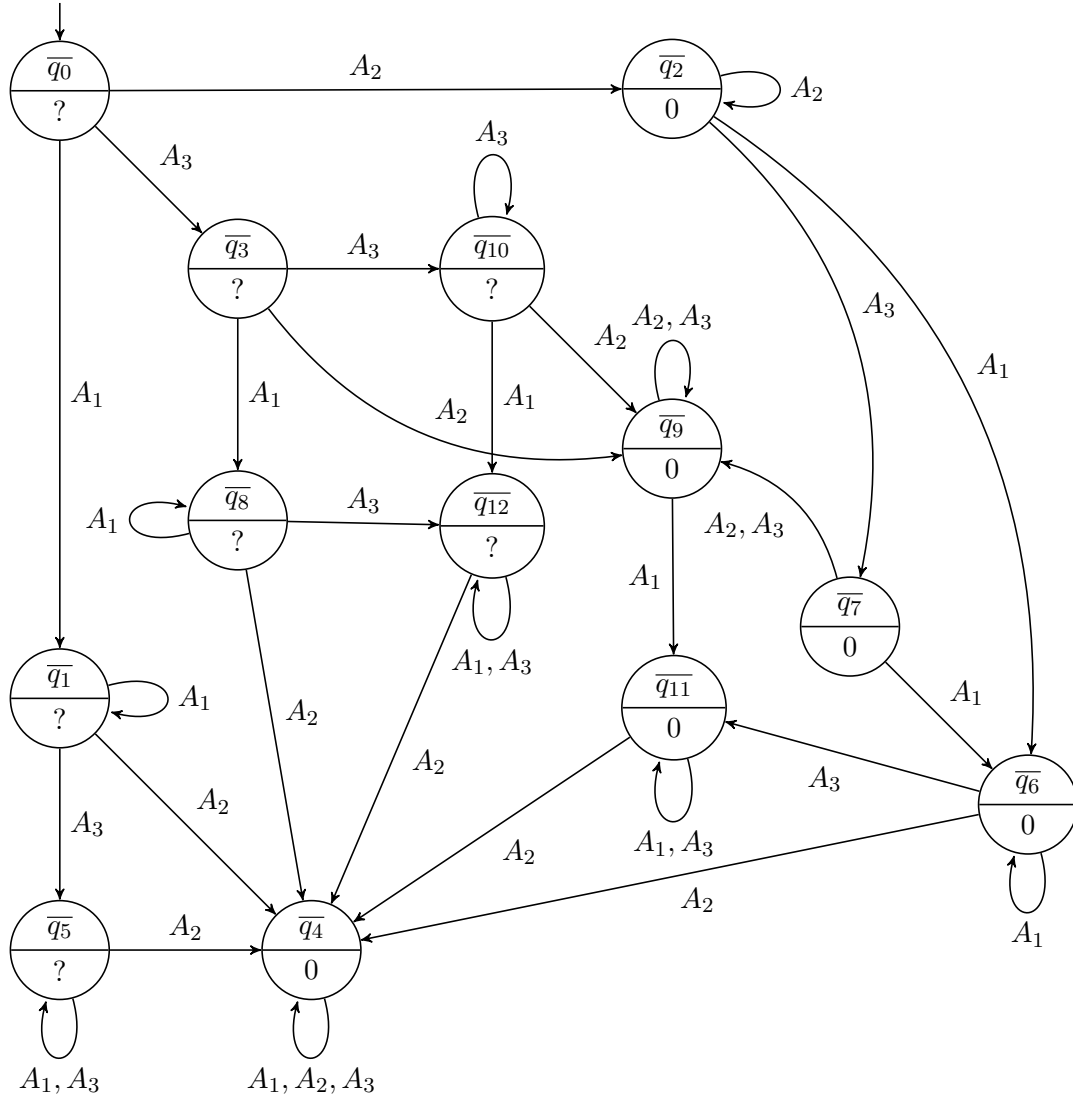
For all $\mathcal{A} \in A_3$, we have:

$$\begin{aligned} \delta(\overline{q_0}, \mathcal{A}) &= \overline{q_3} & \delta(\overline{q_1}, \mathcal{A}) &= \overline{q_5} & \delta(\overline{q_2}, \mathcal{A}) &= \overline{q_7} & \delta(\overline{q_3}, \mathcal{A}) &= \overline{q_{10}} \\ \delta(\overline{q_4}, \mathcal{A}) &= \overline{q_4} & \delta(\overline{q_5}, \mathcal{A}) &= \overline{q_5} & \delta(\overline{q_6}, \mathcal{A}) &= \overline{q_{11}} & \delta(\overline{q_7}, \mathcal{A}) &= \overline{q_9} \\ \delta(\overline{q_8}, \mathcal{A}) &= \overline{q_{12}} & \delta(\overline{q_9}, \mathcal{A}) &= \overline{q_9} & \delta(\overline{q_{10}}, \mathcal{A}) &= \overline{q_{10}} & \delta(\overline{q_{11}}, \mathcal{A}) &= \overline{q_{11}} \\ \delta(\overline{q_{12}}, \mathcal{A}) &= \overline{q_{12}} & & & & & & \end{aligned}$$

- $\Gamma = \mathbb{B}_3$
- for the output function $\lambda : Q \rightarrow \Gamma$, we have:

$$\begin{aligned} \lambda(\overline{q_0}) &= ? & \lambda(\overline{q_1}) &= ? & \lambda(\overline{q_2}) &= 0 & \lambda(\overline{q_3}) &= ? \\ \lambda(\overline{q_4}) &= 0 & \lambda(\overline{q_5}) &= ? & \lambda(\overline{q_6}) &= 0 & \lambda(\overline{q_7}) &= 0 \\ \lambda(\overline{q_8}) &= ? & \lambda(\overline{q_9}) &= 0 & \lambda(\overline{q_{10}}) &= ? & \lambda(\overline{q_{11}}) &= 0 \\ \lambda(\overline{q_{12}}) &= ? & & & & & & \end{aligned}$$

The visualisation of the monitor \mathcal{A}_φ for the \mathcal{ALC} -LTL formula φ looks as follows.



Note that $\xrightarrow{A_1}$ signifies all transitions $\xrightarrow{\mathcal{A}}$ with $\mathcal{A} \in A_1$. The shorthands $\xrightarrow{A_2}$ and $\xrightarrow{A_3}$ are defined analogously.

Let us sum up the results of this chapter. We provided a definition for a construction of a monitor for an \mathcal{ALC} -LTL formula with rigid names that works with incomplete knowledge.

We proved the correctness of the construction and showed that the construction can be

done in 2-2-EXPTIME. Again, further research is needed to determine how the monitor construction behaves with respect to the complexity in practice.

We have given an example of the complete monitor construction for an \mathcal{ALC} -LTL formula with rigid names that works with incomplete knowledge. This example showed one application of the monitor construction introduced in this chapter.

In the next chapter summarises this work and gives some pointers on possible further work.

6 Conclusions and further work

This thesis focuses on the basic functionality of runtime verification using the temporal description logic \mathcal{ALC} -LTL in several cases. Of course, further optimisations are possible and additional tasks and questions arise from the results of this work. In this chapter, we summarise some results of the work and give also some pointers for possible further work.

6.1 Conclusions of this work

In this section we summarise some results of this thesis. We considered runtime verification using temporal description logics in three cases. First, we presented—in Chapter 3—a definition for the monitor construction for \mathcal{ALC} -LTL formulas without rigid names. Additionally, we proved the correctness of the monitor construction. It turned out that the runtime verification problem for an \mathcal{ALC} -LTL formula without rigid names is in 2-EXPTIME.

The second case, as outlined in Chapter 4, was runtime verification for an \mathcal{ALC} -LTL formula with rigid names. Again, we introduced a definition and proved its correctness. In this case, the runtime verification problem is in 2-2-EXPTIME.

Finally, we looked at runtime verification for an \mathcal{ALC} -LTL formula with rigid names with respect to incomplete knowledge. This was done in Chapter 5. We presented a definition for the monitor and proved its correctness. It turned out that the monitor should work on an infinite alphabet. We thus considered the complexity only with respect to the number of states. Also in this case, the state space of the monitor can be constructed in 2-2-EXPTIME.

Unfortunately, due to the high complexity of runtime verification using \mathcal{ALC} -LTL with or without rigid names, possible applications of the proposed constructions are rather limited in practice.

6.2 Possible further work

This section contains some pointers for further work. As stated before, the monitor construction for all three cases is not yet optimised. There are fast algorithms for the translation of LTL-formulas to Büchi automata. One could adapt such algorithms for \mathcal{ALC} -LTL with or without rigid names. Note that such optimisations do not affect the worst-case complexity, but for practical applications they are nonetheless useful, since the worst-case complexity is not reached in every case.

Having done such optimisations, one can implement the algorithms. The resulting tool can be used for monitoring specifications given in terms of \mathcal{ALC} -LTL formulas with or without rigid names. This tool should also be applicable to the most interesting case—namely, runtime verification for \mathcal{ALC} -LTL with rigid names with respect to incomplete knowledge.

Parts of this work can also be applied in other fields. For example, we can use parts of the monitor construction for explicit-state model checking. We defined the corresponding generalised non-deterministic Büchi automata for some \mathcal{ALC} -LTL formula without rigid names and for some \mathcal{ALC} -LTL formula with rigid names, respectively. One can use these constructions for explicit-state model checking of \mathcal{ALC} -LTL formulas with or without rigid names. In order to do so, one has to adapt the definition of product labelled transition systems; see Definition 2.30. Nonetheless, one has to cope with the high complexity also in this case.

Bibliography

- [1] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- [2] Franz Baader, Silvio Ghilardi, and Carsten Lutz. LTL over Description Logic Axioms. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR2008)*, 2008.
- [3] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.
- [4] Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, Kolkata, India, December 2006. Springer-Verlag.
- [5] Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner. *Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
- [6] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 1999.
- [7] Séverine Colin and Leonardo Mariani. *Model-based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*, chapter Run-time Verification, pages 525–556. Springer Verlag, 2005.
- [8] Edith Elkind, Blaise Genest, Doron A. Peled, and Hongyang Qu. Grey-box checking. In *Proceedings of IFIP FORTE*, pages 420–435, 2006.

- [9] E. Allen Emerson. *Handbook of theoretical computer science (vol. B): formal models and semantics*, chapter Temporal and modal logic, pages 995–1072. The MIT Press, 1990.
- [10] 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'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65, Paris, France, July 2001. Springer.
- [11] Carsten Lutz, Frank Wolter, and Michael Zakharyashev. Temporal description logics: A survey. In *Proceedings of the Fifteenth International Symposium on Temporal Representation and Reasoning*. IEEE Computer Society Press, 2008.
- [12] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
- [13] Glenford J. Myers, Corey Sandler, Tom Badgett, and Todd M. Thomas. *The Art of Software Testing, Second Edition*. Wiley, 2004.
- [14] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Comp. Soc. Press, October–November 1977.
- [15] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.
- [16] A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic (Extended Abstract). In *Proceedings of the 12th Colloquium on Automata, Languages and Programming*, pages 465–474, London, UK, 1985. Springer-Verlag.
- [17] Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths. *Symposium on Foundations of Computer Science*, 0:185–194, 1983.

Declaration/Erklärung

I declare herewith that I have written this thesis myself and that the work contained herein is my own, except where explicitly stated otherwise.

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

Dresden, 2009-03-26

Marcel Lippmann