Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Bachelor Thesis

Monitoring First-order Parametric Linear-time Temporal Logic

Submitted by
Kai Hornung

Supervisor
Prof. Bernd Finkbeiner, PhD

Reviewers
Prof. Bernd Finkbeiner, PhD
Prof. Dr.-Ing. Holger Hermanns

Advisor
Peter Faymonville, M.Sc.

Submitted on:
April 30, 2015

Abstract

Linear-time temporal logic (LTL) is a widely used formalism to express specifications for runtime verification. By extending LTL with parameters (PLTL), monitoring algorithms cannot only be used to verify behaviour, but also to collect data on how long certain properties hold or how long it takes for a system to fulfil a property. The fact that LTL (and PLTL) usually work with atomic propositions to express a system's condition is debilitating, since specifications may also depend on data - e.g. values of variables or measurements to verify internal consistency - which is hard to model using solely absence and presence of properties expressed as atomic propositions.
This work is dedicated to address this problem by first extending PLTL with data and afterwards constructing a corresponding monitoring algorithm that also takes care of obtaining measurements for the included parameters. In order to incorporate data, the atomic propositions in PLTL will be replaced by predicates over an arbritrary, possibly infinite domain. Universally quantified variables ranging over this domain will facilitate the construction and readability of data-aware specifications, lifting PLTL up to a first-order logic called First-order PLTL (FO-PLTL). Then, an online monitoring algorithm for measuring a sysem's behaviour using FO-PLTL specifications is presented, together with the corresponding implementation that shows detailed measuring capabilities.

Acknowledgements

I am very grateful to Prof. Bernd Finkbeiner and Peter Faymonville for the opportunity to work on such an interesting and exciting topic. I want to thank Peter for his support and suggestions during the development of this thesis and the corresponding implementation. I also want to thank Prof. Finkbeiner who always helped in word and deed, not only for this thesis, but even more in his mentoring position in the Honor's Program for Bachelor Students during my entire studies so far.
Furthermore I am thankful to everyone who supported me during my studies and took the load off my back, so I could focus on this thesis. In particular, I want to thank my family and especially my girlfriend who helped me hanging on, even when there were difficulties healthwise.
A special thankyou also goes to Petra Klasmeier and Carsten Hornung for proof-reading this thesis and to my computer science teacher in school, who caused my interest in computer science early on and motivated me to study this subject.

Contents

Introduction

Motivation

With the presence of computer based systems in almost every part of everyday life, an increasingly high degree of reliability is expected. Thus behavioural verification of such systems becomes one of the key focuses in computer science. But with growing complexity and dependency on factors beyond the primal implementation (e.g. networks or user interaction), many of these systems are hard to formalize - if possible at all - in order to apply established verification methods like model checking. Especially for these systems, *runtime verification* is a prominent alternative for formal verification and behavioural analysis, as it needs no information about the implementation or undelying model. So called *monitoring algorithms* (or simply *monitors*) observe a particular execution and verify behaviour by analyzing externally measurable events.
When talking about monitors, we distinguish between two types, 'online' and 'offline' monitors. Whereas the former observes the ongoing execution as it proceeds and thus runs side-by-side with the monitored system, the latter has direct access to all positions in the sequence of observed events, as it is triggered after the execution has already terminated. Although offline monitors can be implemented particularly space-efficient [12], online monitors have the advantage of being able to recognize violations early and intervene e.g. by stopping or restarting the current process. This is particularly advantageous, considering the fact that in order to analyze the entire system's behaviour (instead of just the one execution), a large number of monitored runs is desirable.
Most monitors are based on a formal logic to express the system's desired behaviour. One of the most frequently used formalizations for such specifications is the Linear-time Temporal Logic (LTL, [22]). With growing complexity of software systems and more and more unknown factors (e.g. unknown API implementations, network reliability etc.) though, another part of analysis - besides verification - becomes interesting, namely *measuring of temporal behaviour*. First introduced in [2], the LTL-extension called Parametric Linear-time Temporal Logic (PLTL) adds parametrized bounds to LTL's standard temporal operators in order to add the ability to gather information e.g. about internal execution time. In [12], efficient online and offline monitors are presented that return detailed temporal data by determining optimal values for the parameters included in the specification.
But PLTL still has some restrictions when it comes to the expressiveness of some specifications, as one can see in the following example PLTL-formula

$$\Box(request \rightarrow \Diamond_{\leq k} response)$$

which can be read as "At every point in time, a request is getting a response in at most k steps". With the usage of PLTL monitoring (or model checking) algorithms, the optimal

k can easily be determined and therefore information about the maximal duration between request and response can be gathered. For example, a correct monitor determines for the trace of events $\{request\}\{request\}\{response\}$, that the specification is met for the parameter valuation that assigns 2 to k. This may however not be the desired result, since there is only one response for two – most likely different – requests. The more interesting specification would therefore be "At every point in time, every request is getting a *corresponding* response". In order to express this specification, though, a formalism to include some kind of reference between a response and the respective request is needed. The source of this problem lies in the fact that LTL (and therefore PLTL) is based on *atomic propositions* to express the system's properties. In every event, each of these properties either holds (i.e. the proposition is present) or does not hold (i.e. the proposition is absent). This thesis is dedicated to adding the ability to extend PLTL in a way, that these propositions can carry *data* (e.g. the request could carry a reference to the corresponding request) and therefore express data-aware specifications, like the one above.

Outline

The aim of this work is to develop a PLTL extension that is able to express such data-aware properties and a corresponding online monitoring algorithm. Besides the formal introduction of those two parts in this thesis, there will also be an implementation for the monitor with some optimizations for both time- and space-efficiency.
In order to extend PLTL with "data", we adapt mechanisms from first-order predicate logic. By introducing quantifiers and variables to PLTL, we not only add the ability to include identifiers as needed e.g. for the above specification, but also to handle arbitrary data. The specification "Every request ist getting a corresponding response" will then be formalized as

$$\Box\big(\forall id.\, request(id) \to \Diamond_{\leq k} response(id)\big).$$

Similar to the way it is done in [12], the aim of PLTL monitoring is to determine an *optimal* parameter-valuation under which the specification is met for the given input. Since the introduction of PLTL in [2] does not give any definition of this kind of optimality for such valuations, we will consider a semantically modified variant of PLTL, like it is done for online monitoring in [12]. This variant eliminates the ambiguity of the measuring component of a PLTL monitor, as shown in the first chapter.
An advantage that comes in handy with the use of variables in the specification is the ability to not only determine an overall measuring result, but also collect value-dependent temporal information about the considered system. With the specification above, one could e.g. not only determine a k such that every response follows in at most k steps, but also determine an optimal value for every respective id.
This adds to the enhanced verification capabilities a strong measuring component that can be used to gain detailed temporal information about the monitored trace and therefore system, as well.
The thesis itself is split into three parts. Whereas the first part formally introduces PLTL and its extension – namely First-order PLTL (FO-PLTL) – the second and third part are dedicated to the development of the monitoring algorithm, giving the formal definition in the former and implementation details and results in the latter one.
The first chapter not only introduces First-order PLTL, but also defines the correct measuring behaviour that is needed for the construction of the monitoring algorithm. The

second chapter then presents an automata-based algorithm that can be used for verification and measuring of FO-PLTL specifications. Every component of the algorithm will be introduced and proven formally.
The third chapter then presents some information about the corresponding implementation that is develoed in Scala[1]. It covers important optimizations that are needed for processing large amounts of data. There are also two sections about some results that are obtained from using the monitoring algorithm that also show more information about the detailed measuring capabilities.
At the end of this thesis, there will be a short conclusion of the presented work and a part about some related work in this field.

In order to keep a clear and lucid structure, all formal proofs and lemmas are located in the appendix of this thesis.

[1]see `http://www.scala-lang.org`

Chapter 1

First-order Parametric LTL

In this chapter, we will lay the foundation for the measuring algorithm in the next chapter by introducing the underlying temporal logic, called *First-order Parametric LTL* (FO-PLTL). As we have briefly seen in the last chapter, we can use a temporal logic like LTL and their extensions to formalize specifications that describe e.g. the desired behaviour of a computer system. Such specifications can then be used with monitoring or model-checking algorithms to verify that a system behaves correctly, i.e. in a way that meets the specification. Naturally, each temporal logic has its advantages and disadvantages, may it be in expressiveness or in the efficiency of the ways they can be checked. We will use the basis that Parametric LTL ([2]) provides and develop a logic that has the same capabilities, but is also able to express specifications that reason about *data*.
The chapter will be structured as follows: In the first section, there will be a brief introduction to LTL and PLTL and we will introduce the basic notion that is used in these two formalisms to express specifications. We will present neither their formal syntax nor semantics, but rather focus on the intuitive meaning behind the operators and how they can be used to describe specific behaviour. Afterwards, we will extend PLTL with data by formally introducing First-order Parametric LTL, starting with the syntactic concept and then continuing by giving the semantic definitions. At the end of this chapter, we will define the measuring part of FO-PLTL and how to tweak optimal results.

1.1 A short introduction to LTL and PLTL

The *Linear-time Temporal Logic* (LTL) is a temporal logic that can be used to express future-dependent properties on a discrete time model, whose notion first appeared in [22]. The simplified idea is that at every point in time, some properties "hold". These properties are formally expressed as *atomic propositions* (e.g. $sensor\_on$, $request$, $start$...). At every point in time, each of these properties is either absent or present, altogether describing a system's current state. The atomic propositions also form the lowest-level LTL formulae, i.e. every atomic proposition is also an LTL formula that describes the specification that requires the corresponding property of the system to hold. For example, the LTL formula $sensor\_on$ requires the system's sensor to be activated.
LTL formulae can be constructed using a variety of *operators*. First of all, it allows the usual boolean connectives as negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$) and implication ($\rightarrow$), each being interpreted in the usual manner, for example $\neg request$ describing that

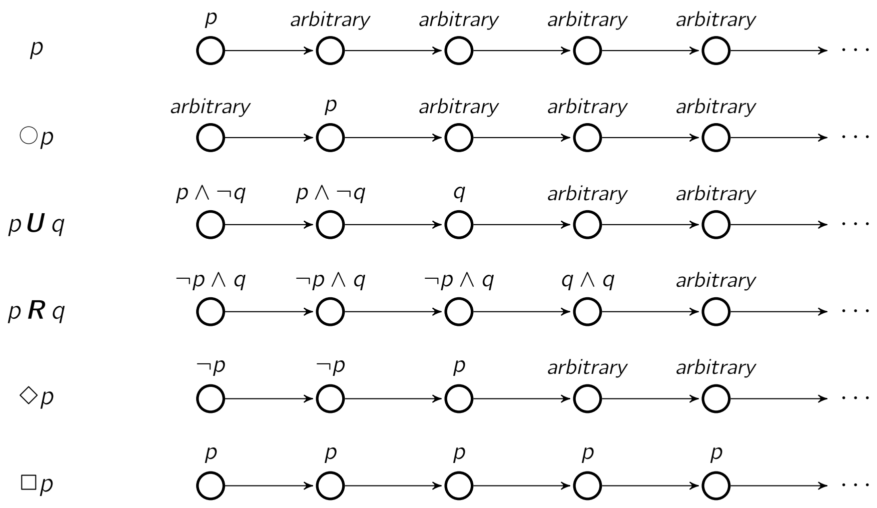

Figure 1.1: LTL operator semantics

there must be no request at the time checked. Besides these operators, LTL uses *temporal* operators to express specifications that describe behaviour in time. There are the unary operators Next ($\bigcirc$), Eventually ($\Diamond$) Globally ($\Box$) and the two binary operators Until ($\boldsymbol{U}$) and Release ($\boldsymbol{R}$). Before we describe their semantic meaning, we first need to define how to interprete LTL specifications in general:
To check if an execution meets an LTL specification, we consider executions as sequences of events, named traces. Each event represents a state at some point in time and is fomalized as a subset of atomic propositions, namely the ones that describe the system's properties at that point in time. We can then check the *satisfaction* of an LTL specification *starting* at a given point in time, since there are no operators that describe past behaviour. An execution meets a LTL specification whenever the entire corresponding trace satisfies the whole LTL formula, i.e. the formula is satisfied at the first position of the trace. As briefly mentioned above, at some point, i.e. regarding a specific event, the LTL formula p (where p is an atomic proposition) is satisfied if and only if p is present in the current event, i.e. the property holds. The boolean connectives are interpreted as usual, whereas the temporal operators have the following meaning:

- $\bigcirc\varphi$ is satisfied iff φ is satisfied in the *next* step.
- $\varphi\,\boldsymbol{U}\,\psi$ is satisfied iff ψ is satisfied now or at some point in the future and *until* then, in every step φ is satisfied.
- $\varphi\,\boldsymbol{R}\,\psi$ is satisfied iff ψ is satisfied until and including the point it is *release*d when φ is satisfied, or always, if no such point exists.
- $\Diamond\varphi$ is satisfied iff φ is *eventually* satisfied, i.e. now or in some later step.
- $\Box\varphi$ is satisfied iff φ is *globally* satisfied, i.e. in every step from now on.

The meaning of these operators is also depicted in Figure 1.1. As already mentioned above, the formal syntax and semantics for LTL will not be introduced here, but they can be found in [3], pp. 231-236 for infinite traces or in [10] for finite traces. It is interesting to know that, in fact, only negation, conjunction, Next and Until are the operators that are essential for the expressiveness of LTL, every other operator can be derived ([3], pp. 231f.).

Example 1.1.1. *Consider the following LTL formula:*

$$\Box(request \rightarrow \Diamond response)$$

which reads "At every point in time it holds that, whenever there is a request, a response will follow", so "Every reqeust eventually gets a response".

Monitoring or model checking LTL focuses on *qualitative* verification of specifications that decribe temporal behaviour, i.e. *if* some property holds or not, rather than being interested in quantitative information about e.g. *how long* it takes until some property holds. In order to easily express more restrictive and detailed specifications, temporally *bounded* operators were introduced in [18] to enhance the standard LTL operators. For example $a\boldsymbol{U}_{\leq 5}b$ is only satisfied if b holds in *at most five steps* and until then a holds. The operators $\Diamond_{\leq 5}a$, $a\boldsymbol{R}_{\leq 5}b$ and $\Box_{\leq 5}b$ work similarly, the latter two stating that b needs to hold at least for the next five steps (unless released by an earlier a).
Is is easy to see that only adding constant bounds to the temporal operators in LTL in this manner does not increase its expressiveness, since they can be replaced by iterations of the $\bigcirc$-operator. Nevertheless, they give advantages both when it comes to constructing specifications and to building efficient model checking and monitoring algorithms.

Seizing on this idea, the *Parametric Linear-time Temporal Logic* (PLTL) was introduced in [2]. It presents a way to not only verify, but also *measure* temporal behaviour by determining such bounds for the LTL operators while checking a specification. It allows replacing the temporal bounds with *parameters* whose values are to be determined by the model checking or monitoring algorithm.
PLTL extends LTL with the formal introduction of the following two operators: $\Diamond_{\leq k}$ and $\Box_{\leq k}$. The semantics correspond to their LTL counterparts by restricting the time interval to k steps, i.e. for $\Diamond_{\leq k}\varphi$, φ needs to be satisfied in at most k steps and for $\Box_{\leq k}\varphi$, it must be satisfied for at least the next k steps. Derived from these two operators, there are parametric variants for $\boldsymbol{U}$ and $\boldsymbol{R}$, as well as the respective counterparts with a lower bound, e.g. $\Diamond_{>k}\varphi$ (φ needs to be satisfied in more than k steps). Their full semantic definition can be found in [2] for infinite and [12] for finite traces.
As mentioned above, the main idea is to measure the temporal behaviour of a system whilst verifying the correctness. This is achieved by determining values for the included parameters. Determining values for each parameter, though, is a tricky task, since every parametric temporal operator behaves monotonically, e.g. if $\Diamond_{\leq k}a$ holds for some k, it holds for every greater k' as well. Defining which value is in fact *optimal* and how to obtain it will be considered in detail later in this chapter.

Example 1.1.2. *Consider the following PLTL formula:*

$$\Box(request \rightarrow \Diamond_{\leq k} response)$$

which is obtained from the one in Example 1.1.1 by replacing the $\Diamond$-operator with its parametric counterpart using a new parameter k. Now checking this property is not only about

$$\underbrace{\{request\} \rightarrow \{\} \rightarrow \{response\}}_{\text{2 steps}} \rightarrow \underbrace{\{request\} \rightarrow \{response\}}_{\text{1 step}} \rightarrow \underbrace{\{request, response\}}_{\text{0 steps}}$$

Figure 1.2: Determining a parameter value in PLTL

verifying *that every request eventually gets a response, but also about determining* how long *it takes at most until a request is followed by a response. As depicted in Figure 1.2, the given trace satisfies the specification for* $k = 2$ *and every greater* k*, because every request is followed by a response in at most* 2 *steps.*

As mentioned in the introduction, we are aiming to develop an extension of PLTL that can handle "data". This basically means that from now on we are interested in specifications that talk about values of some computer system's component. While PLTL and LTL only consider properties that are either present or absent (formalized as atomic propositions), we now want to be able to talk about data-aware properties (e.g. values of variables, measurements, counters etc.). In order to reason about such properties, we first need to formalize them. We will do this using well known concepts of first-order predicate logic and first-order LTL extensions like the one in [7].

1.2 Syntax

To express data-aware properties, we will part from atomic propositions and substitute *predicates*. Every predicate can carry a number of "arguments" that can be of an arbitrary kind of data. For example, we could use the predicate $var_s($"`Hello World!`"$)$ to describe the fact that the program variable s currently stores the string "Hello World!". We can also use predicates to describe more general properties of data, e.g. a predicate $leq(x, y)$ that holds if and only if $x \leq y$. While the former one depends on the current state of the system, the latter is rigid, i.e. does not depend on the considered execution of the system. We will refer to these two kinds of predicates as a priori *uninterpreted* and *interpreted* predicates, respectively.
Formally, we adapt the syntactic concepts introduced in [7] and introduce first-order signatures for First-order Parametric Temporal Logic:

Definition 1.1 (Parametric first-order signature)**.**
A parametric first-order signature $\mathcal{S}$ *is a tuple* $(\mathcal{F}, \mathcal{P}, \mathcal{X}, \mathcal{K})$ *together with an implicit function* arity : $\mathcal{F} \cup \mathcal{P} \rightarrow \mathbb{N}$ *where:*

- $\mathcal{F}$ *is a finite set of* function symbols *with* $\forall f \in \mathcal{F}.\, \text{arity}(f) \geq 0$.
- $\mathcal{P} = \mathcal{P}^I \cup \mathcal{P}^U$ *is a finite set of* predicate symbols *where*
  - $\mathcal{P}^I \cap \mathcal{P}^U = \emptyset$,
  - $\forall p \in \mathcal{P}^U.\, \text{arity}(p) \geq 0$ *and* $\forall r \in \mathcal{P}^I.\, \text{arity}(r) > 0$.
- $\mathcal{X}$ *is a set of* variables.
- $\mathcal{K}$ *is a finite set of* parameters.
- $\mathcal{F}$, $\mathcal{P}$, $\mathcal{X}$ *and* $\mathcal{K}$ *are pairwise disjunct.*

From now on, we call parametric first-order signatures simply signatures. As usual, we refer to 0-ary function symbols as *constants* and 0-ary predicate symbols as *propositions*. The partitioning of the predicate symbols into the sets $\mathcal{P}^I$ and $\mathcal{P}^U$ corresponds to the a priori interpreted and respecively uninterpreted predicate symbols, that have been mentioned above. We use $a, b, c, \ldots$ to denote constants, $f, g, h, \ldots$ as function symbolds, $p, q, r, \ldots$ as predicate symbols and $x, y, z, \ldots$ to name variables.

The parametric first-order signature gives us all components that are needed to construct formulae in First-order PLTL. Since the lowest-level formulae in FO-PLTL are not as easy as atomic propositions like they were for PLTL, we need to define the notion of *terms*:

Definition 1.2 (Terms)**.** Terms *over a parametric first-order signature* $\mathcal{S} = (\mathcal{F}, \mathcal{P}, \mathcal{X}, \mathcal{K})$ *are defined inductively as follows:*

- *For every* $x \in \mathcal{X}$*,* x *is a term and for every* $a \in \mathcal{F}$ *with* $arity(a) = 0$*,* a *is a term.*
- *For every* $f \in \mathcal{F}$ *with* $\text{arity}(f) = n > 0$ *and terms* $t_1, \ldots, t_n$*,* $f(t_1, \ldots, t_n)$ *is a term.*

We denote the set of terms over $\mathcal{S}$ *as* $T_{\mathcal{S}}$*.*

It is easy to see, that terms are defined in the same way as they are for non-temporal first-order predicate logic. As the final step for the syntactic definition of First-order PLTL, we introduce *formulae*.

Definition 1.3 (First-order PLTL fomulae)**.**
First-order PLTL formulae φ *over a parametric first-order signature* $\mathcal{S} = (\mathcal{F}, \mathcal{P}, \mathcal{X}, \mathcal{K})$ *are defined as follows:*

$$\begin{aligned}\varphi \quad ::= \quad & \boldsymbol{true} \;\big|\; p(t_1, \ldots, t_n) \;\big|\; \neg\varphi \;\big|\; \varphi \wedge \varphi \;\big|\; \varphi \vee \varphi \;\big|\; \bigcirc\varphi \;\big|\; \varphi\, \boldsymbol{U}\, \varphi \;\big|\; \varphi\, \boldsymbol{R}\, \varphi \;\big| \\ & \Diamond_{\leq k}\varphi \;\big|\; \Box_{\leq k}\varphi \;\big|\; \forall(x_1, \ldots, x_m){:}r.\, \varphi \;\big|\; \exists(x_1, \ldots, x_m){:}r.\, \varphi\end{aligned}$$

where $k \in \mathcal{K}$*,* $p \in \mathcal{P}$*,* $n = \text{arity}(p)$*,* $t_1, \ldots, t_n \in T_{\mathcal{S}}$*,* $r \in \mathcal{P}^U$*,* $m = \text{arity}(r) > 0$ *and* $x_1, \ldots, x_m \in \mathcal{X}$*. We also allow the following derived operators:*

$$\boldsymbol{false}\ ,\ \rightarrow\ ,\ \Diamond\ ,\ \Box\ ,\ \Diamond_{>k}\ ,\ \Box_{>k}\ ,\ \boldsymbol{U}_{\leq k}\ ,\ \boldsymbol{U}_{>k}\ ,\ \boldsymbol{R}_{\leq k}\ ,\ \boldsymbol{R}_{>k}$$

The operators $\bigcirc$, $\boldsymbol{U}$, $\boldsymbol{R}$, $\Diamond$ *and* $\Box$ *are called* temporal*. Their subscripted variants, e.g.* $\Diamond_{\leq k}$ *are further called* parametric*. To avoid the need of fully-parenthesized formulae, we define the following precedence order on the operators:*

$$\neg = \bigcirc = \Diamond = \Box \quad > \quad \boldsymbol{U} = \boldsymbol{R} \quad > \quad \wedge \quad > \quad \vee \quad > \quad \rightarrow \quad > \quad \forall = \exists$$

where all parametric operators have the same precedence as the corresponding non-temporal operator. We also declare that binary operators with the same precedence bind from right to left, e.g. $\varphi_1\, \boldsymbol{U}\, \varphi_2\, \boldsymbol{R}\, \varphi_3 = \varphi_1\, \boldsymbol{U}\, (\varphi_2\, \boldsymbol{R}\, \varphi_3)$.
Formulae containing a parametric operator are called parametric*, such with a temporal operator as its main connective are called* temporal*. The same way it is done for non-temporal predicate logic, we call* $\boldsymbol{true}$*,* $\boldsymbol{false}$ *and formulae of the type* $p(t_1, \ldots, t_n)$ atomic *and such atomic formulae that contain no variables* atomic sentences.
We refer to formulae in which every variable is bound by a quantifier (i.e. appears within the scope of a quantifier that introduces the variable) as ground *formulae.*

First-order PLTL formulae can be used to express a variety of specifications that describe a temporal behaviour. But before we can talk in more detail about formulae aside from what they look like, we need to give meaning to the syntactic components. For this purpose, we will now introduce the formal semantics of FO-PLTL.

1.3 Semantics

In this section, we will define the semantics for FO-PLTL that our monitor is based on. All in all, the semantics are more or less directly constructed as a combination of the finite trace PLTL semantics (taken from [12]) and standard predicate-logic semantics for quantification. Nevertheless, one can observe in Definition 1.3 that there is a modification of standard quantification syntax. This is, in fact, a restriction to ensure efficient monitoring and will be explained in the following.

1.3.1 Ensuring finite-domain quantification

A problem that arises with every first-order logic is the fact that checking quantifications over infinite domains is hard. For example, consider examples as simple as:

$$\forall x.p(x) \qquad \exists x.\neg p(x)$$

It is nearly impossible for a monitor to check the truth value of these formulae efficiently when there are infinitely many possible values for x. Since we do not want to restrict FO-PLTL to finite domains[1], we need to find another way to resolve this problem. The same way it is done in [7] and for other first-order LTL extensions ([4],[15]), in order to avoid this kind of quantification, we add some restrictions to our FO-PLTL definition.
First of all, we assume that every event is *finite*, i.e. at every point in time, only finitely many properties hold. Since events formally represent a system's state in time which usually consists of finitely many properties, the affect of this restriction on the expressiveness of FO-PLTL is negligible. Nevertheless, it ensures that the *active domain*, i.e. the set that contains all values of the domain that have been "seen" in the trace up to this point, is finite. Now, in order to make sure we can monitor specifications efficiently, we want to ensure that we only need to consider the values in the active domain to check if a quantified property is satisfied. This problem of *domain-independence* has already been extensively studied ([16],[17],[1]). Unfortunately, though, domain-indepence is in general undecidable, even for non-temporal first-order logic ([1]), which means, that we need to find a way to restrict FO-PLTL in a way that our specifications are domain-independent by construction. This is where the "unusual" syntax for quantification, that has been introduced in the last section, originates. Similar to the way presented in [7], in FO-PLTL we only allow quantification of the following kind:

$$\forall x_1 ... \forall x_n .\ p(x_1, ..., x_n) \rightarrow \varphi \qquad\qquad \exists x_1 ... \exists x_n .\ p(x_1, ..., x_n) \wedge \varphi$$

[1] By restricting FO-PLTL to finite domains, we get no more expressiveness than in propositional PLTL, as illustrated in the following: With a finite domain and finitely many predicates, the set of atomic sentences would be finite, as well. By setting AP to the set of all possible atomic sentences and replacing every universal (and existential) quantification with a finite conjunction (and disjunction) over all possible values, we could transform every FO-PLTL formula into an equivalend PLTL formula, that could be monitored using PLTL-monitoring techniques.

Under standard quantification semantics, satisfaction of these formulae only depends on the values $d_1, ..., d_n$ for which $p(d_1, ..., d_n)$ holds, i.e. for which $p(d_1, ..., d_n)$ is in the current event. Since therefore they are in the active domain, we know these formulae to be domain-independent. The predicate p (taken from the set of a priori *uninterpreted* predicates $\mathcal{P}^U$) is called the *guard* of the quantification. To make it clear that p is a guard, we will use the notation

$$\forall(x_1, ..., x_n){:}p.\,\varphi \qquad\qquad \exists(x_1, ..., x_n){:}p.\,\varphi$$

that has already been introduced in the formal FO-PLTL syntax definition.

1.3.2 First-order structures

We will now start giving meaning to the syntactical constructs of First-order PLTL. As already mentioned above, we want variables to range over an arbitrary, possibly infinite, domain. Naturally, the constants and function also need to take values in these domain, and therefore terms, as well. We will start by introducing *first-order structures* that are used to give meaning to the invariable components of a parametric first-order signature, i.e. functions, constants and interpreted predicate symbols:

Definition 1.4 (First-order Structure)**.** *A* first-order structure *over a signature $\mathcal{S}$ where $\mathcal{S} = (\mathcal{F}, \mathcal{P}, \mathcal{X}, \mathcal{K})$, or simply $\mathcal{S}$-structure, is a tuple $\mathbb{S} = (\mathbb{D}, \mathbb{I})$ where $\mathbb{D}$ is a nonempty set called* domain *and $\mathbb{I}$ is an* interpretation *that maps*

- *every constant $a \in \mathcal{F}$ with $\mathrm{arity}(a) = 0$ to a value $a^{\mathbb{I}} \in \mathbb{D}$,*
- *every function symbol $f \in \mathcal{F}$ with $\mathrm{arity}(f) = n > 0$ to a function $f^{\mathbb{I}} : \mathbb{D}^n \to \mathbb{D}$ and*
- *every interpreted predicate symbol $p \in \mathcal{P}^I$ to a countable set $p^{\mathbb{I}} \subseteq \mathbb{D}^{\mathrm{arity}(p)}$.*

We will write $p(d_1, ..., d_n) \in \mathbb{I}$ instead of $(d_1, ..., d_n) \in p^{\mathbb{I}}$ to express the notion that a property $p(d_1, ..., d_n)$ holds. The first-order structure builds the foundation for interpreting FO-PLTL formulae. It states the domain from which values can be taken and defines values for every component of the formulae for which the value or truth value does not change during the execution. For constants and function symbols, this interpretation is straightforward. It defines the semantic meaning by assigning specific values and functions ranging over the given domain. Furthermore it gives meaning to the interpreted predicate symbols whose interpretations do not change in time, either. As mentioned above, the idea behind interpreted predicate symbols is to express properties that do not depend on the computer system's current state, e.g. comparison results between different values in the domain. As an example, the aforementioned binary predicate leq would be interpreted via the domain $\mathbb{D} = \mathbb{Z}$ as $leq^{\mathbb{I}} := \{(a, b) \in \mathbb{Z} \mid a \leq b\}$, which follows the intuitive meaning of the less-or-equal property. It has been shown in [7] that this notion can be extended to interpretations that are not rigid, i.e. change over time, but can still be computed at every point in time. Since this extension does not change anything for the monitoring approach itself, in this thesis, we assume every interpreted predicate to have an invariable meaning. Another part that varies from the first-order LTL semantics defined in [7] is the fact that we do not use "sorted", i.e. typed, signatures and structures. The reason for this is the same as before; it does not change anything for the monitoring approach itself and

can thus, for simplicity reasons, be omitted withouth changing the expressiveness of FO-PLTL. Nevertheless, it shows that without much effort, a type checking component could be integrated into this monitoring algorithm.
So far we have given meaning to every part of FO-PLTL that does not depend on the monitored system's current state. Since FO-PLTL is a *temporal* logic, though, we now address the temporal properties of FO-PLTL, especially the notion of "satisfaction" of First-order PLTL formulae. Because we have already seen the informal understanding of formula satisfaction for LTL and PLTL, we can now focus on the *formal* introduction.

1.3.3 Formula satisfaction

In this section, we will introduce the formal semantics for First-order PLTL. For the rest of this chapter, let a signature $\mathcal{S} = (\mathcal{F}, \mathcal{P}, \mathcal{X}, \mathcal{K})$ and a $\mathcal{S}$-structure $\mathbb{S} = (\mathbb{D}, \mathbb{I})$ be given.
As mentioned in the introduction of LTL, the idea behind a temporal logic like FO-PLTL is to formally express specifications for which we can algorithmically check, if it is met by the monitored system's behaviour. When it comes to monitoring, we consider a specific execution of the monitored system, formalized as a sequence of *events* which describe the system's state at some point in time. For First-order PLTL, we only consider *finite* executions, as the main interest lies in the measuring result which is to be returned after checking the entire execution. Therefore we will define a *finite-trace semantics* for FO-PLTL that captures this idea, similar to the way it is done in [12] for PLTL. This means in particular that at the end of monitoring a finite execution there is no information about the possibility of satisfaction after processing further events. The result only captures *if* the system's behaviour up to this point follows the specification and if so, what are the measured temporal bounds. This supports the idea behind PLTL which prioritizes the measuring task and makes it a subordinate interest *why* a specification is not met, i.e. if there have been violations or there are still unfulfilled obligations after reading a finite trace of events.
We will look into the measuring behaviour, i.e. determining optimal parameter-values, in section 1.3.4. For now, our main focus lies on the question when (depending on given parameter-values) a specification is met and when it is not. For this purpose, we first introduce traces and events formally:

Definition 1.5 (Actions, events and traces)**.**
For any $p \in \mathcal{P}^U$ with arity$(p) = n$ *and $d \in \mathbb{D}^n$, we call the tuple (p, d) an* action. *A finite set of actions is called an* event. *We denote the set of all actions as $Act_{\mathbb{S}}$ and all events as $E_{\mathbb{S}}$.*
A trace *σ is a finite sequence of events, i.e. for some n, a function $\sigma : \{0, ..., n-1\} \to E_{\mathbb{S}}$, where n is called the* length *of the trace and is denoted as $|\sigma|$. Whenever clear from context, the subscript of $Act_{\mathbb{S}}$ and $E_{\mathbb{S}}$ is omitted.*

For a trace σ and $i < |\sigma|$, we will write $p(d_1, ..., d_n) \in \sigma[i]$ instead of $(p, (d_1, ..., d_n)) \in \sigma(i)$ and $\sigma[i...]$ to denote the sub-trace starting at index i. Furthermore we define for every $p \in \mathcal{P}^U$ with arity$(p) = n > 0$ and i, the set $\sigma_p[i] := \{d \in \mathbb{D}^n \mid (p, d) \in \sigma[i]\}$ of all values for which the predicate p is present at point i.
Actions, events and traces give meaning to the a priori uninterpreted predicates and are used to describe the system's state and how it changes over time, e.g. the current value of a program variable.

Satisfaction of a FO-PLTL specification is defined inductively over the structure of the formula, i.e. different subformulae need to be satisfied at different points in time. For this reason, we will define satisfaction depending on the position in the trace. In First-order PLTL, satisfaction does not only depend on the monitored trace, but also on the values of the parameters and free variables in the specification. To capture these values formally, we introduce valuations and assignments:

Definition 1.6 (Valuations and assigments)**.**

1. *A* parameter-valuation *is a, possibly partial, function* $\alpha : \mathcal{K} \to \mathbb{N} \cup \{\infty\}$.
2. *A* variable-assignment *is a, possibly partial, function* $\beta : \mathcal{X} \to \mathbb{D}$.
3. *The* interpretation $\mathbb{I}_\beta : T_\mathcal{S} \to \mathbb{D}$ *of a term* t *under a valuation* β *is defined as follows:*

$$\mathbb{I}_\beta(t) \quad := \quad \begin{cases} \beta(x) & \text{if } t = x \in \mathcal{V} \\ a^{\mathbb{I}} & \text{if } t = a \in \mathcal{F} \text{ with } \mathrm{arity}(a) = 0 \\ f^{\mathbb{I}}\big(\mathbb{I}_\beta(t_1), ..., \mathbb{I}_\beta(t_n)\big) & \text{if } t = f(t_1, ..., t_n) \end{cases}$$

We use variable-assignments and the corresponding term-interpretations to determine the value of a term under the current circumstances. Using $\mathbb{I}_\beta$, we can consider every atomic formula as an atomic sentence and therefore find out whether or not they currently hold. For reasons of simpler notation, we extend term-interpretations to atomic formulae by defining

$$\mathbb{I}_\beta\big(p(t_1, ..., t_n)\big) \quad := \quad p\big(\mathbb{I}_\beta(t_1), ..., \mathbb{I}_\beta(t_n)\big)$$

for every predicate p with $\mathrm{arity}(p) = n$ and terms $t_1, ..., t_n$. By having given meaning to every syntactical component of First-order PLTL, we can now introduce the formal semantics for formula satisfaction:

Definition 1.7 (First-order-PLTL formula satisfaction)**.**
For a FO-PLTL formula φ*, a trace* σ*, a position* i *with* $0 \le i < |\sigma|$*, a parameter-valuation* α *which is total on the set of parameters occuring in* φ *and a variable-assignment* β *which is total on the set of variables occuring freely in* φ*, we inductively define the* satisfaction relation $(\sigma, i, \alpha, \beta) \models \varphi$ *(in words: "*σ *satisfies* φ *at position* i *under* α *and* β*") as follows:*

$$(\sigma, i, \alpha, \beta) \models \boldsymbol{true}$$

$$(\sigma, i, \alpha, \beta) \models p(t_1, ..., t_n) \quad \text{iff} \quad \begin{cases} \mathbb{I}_\beta\big(p(t_1, ..., t_n)\big) \in \sigma[i] & \text{if } p \in \mathcal{P}^U \\ \mathbb{I}_\beta\big(p(t_1, ..., t_n)\big) \in \mathbb{I} & \text{if } p \in \mathcal{P}^I \end{cases}$$

$$(\sigma, i, \alpha, \beta) \models \neg\varphi \quad \text{iff} \quad (\sigma, i, \alpha, \beta) \not\models \varphi$$

$$(\sigma, i, \alpha, \beta) \models (\varphi_1 \wedge \varphi_2) \quad \text{iff} \quad (\sigma, i, \alpha, \beta) \models \varphi_1 \text{ and } (\sigma, i, \alpha, \beta) \models \varphi_2$$

$$(\sigma, i, \alpha, \beta) \models (\varphi_1 \vee \varphi_2) \quad \text{iff} \quad (\sigma, i, \alpha, \beta) \models \varphi_1 \text{ or } (\sigma, i, \alpha, \beta) \models \varphi_2$$

$$(\sigma, i, \alpha, \beta) \models \bigcirc\varphi \quad \text{iff} \quad i + 1 < |\sigma| \text{ and } (\sigma, i + 1, \alpha, \beta) \models \varphi$$

$$(\sigma, i, \alpha, \beta) \models \varphi_1 \,\boldsymbol{U}\, \varphi_2 \quad \text{iff} \quad \text{there exists } j,\ i \le j < |\sigma| \text{ where } (\sigma, j, \alpha, \beta) \models \varphi_2 \text{ and } (\sigma, j', \alpha, \beta) \models \varphi_1 \text{ for each } j',\ i \le j' < j$$

$$(\sigma, i, \alpha, \beta) \models \varphi_1 \,\boldsymbol{R}\, \varphi_2 \quad \text{iff} \quad \text{for each } j,\ i \le j < n, \text{ either } (\sigma, j, \alpha, \beta) \models \varphi_2 \text{ or there exists } j',\ i \le j' < j \text{ where } (\sigma, j', \alpha, \beta) \models \varphi_1$$

$(\sigma, i, \alpha, \beta) \models \Diamond_{\leq k}\varphi$	*iff*	*there exists* j *with* $0 \leq j \leq \alpha(k)$ *and* $i + j < \lvert\sigma\rvert$ *such that* $(\sigma, i + j, \alpha, \beta) \models \varphi$
$(\sigma, i, \alpha, \beta) \models \Box_{\leq k}\varphi$	*iff*	$(\sigma, i + j, \alpha, \beta) \models \varphi$ *for each* j *where* $0 \leq j \leq \alpha(k)$ *and* $i + j < \lvert\sigma\rvert$
$(\sigma, i, \alpha, \beta) \models \forall(x_1, ..., x_n){:}p.\, \varphi$	*iff*	$\big(\sigma, i, \alpha, \beta[{}^{d}/_{(x_1,...,x_n)}]\big) \models \varphi$ *for each* $d \in \sigma_p[i]$
$(\sigma, i, \alpha, \beta) \models \exists(x_1, ..., x_n){:}p.\, \varphi$	*iff*	$\big(\sigma, i, \alpha, \beta[{}^{d}/_{(x_1,...,x_n)}]\big) \models \varphi$ *for some* $d \in \sigma_p[i]$

where $\beta[{}^{d}/_{(x_1,...,x_n)}]$ *denotes the function that for* $d = (d_1, ..., d_n)$ *maps* x_i *to* d_i *(for all* i*) and* x *to* $\beta(x)$ *for every other* x.
The derived operators are assigned the following semantic interpretations:

$$\begin{array}{rclrcl}
\boldsymbol{false} & := & \neg\boldsymbol{true} & \varphi_1 \rightarrow \varphi_2 & := & \neg\varphi_1 \vee \varphi_2 \\
\Diamond\varphi & := & \boldsymbol{true}\,\boldsymbol{U}\,\varphi & \Box\varphi & := & \boldsymbol{false}\,\boldsymbol{R}\,\varphi \\
\Diamond_{>k}\varphi & := & \Box_{\leq k}\Diamond\bigcirc\varphi & \Box_{>k}\varphi & := & \Diamond_{\leq k}\Box\bigcirc\varphi \\
\varphi_1\,\boldsymbol{U}_{\leq k}\varphi_2 & := & (\varphi_1\,\boldsymbol{U}\,\varphi_2) \wedge \Diamond_{\leq k}\varphi_1 & \varphi_1\,\boldsymbol{U}_{>k}\varphi_2 & := & \Box_{\leq k}\big(\varphi_1 \wedge \bigcirc(\varphi_1\,\boldsymbol{U}\,\varphi_2)\big) \\
\varphi_1\,\boldsymbol{R}_{\leq k}\varphi_2 & := & (\varphi_1\,\boldsymbol{R}\,\varphi_2) \vee \Box_{\leq k}\varphi_1 & \varphi_1\,\boldsymbol{R}_{>k}\varphi_2 & := & \Diamond_{\leq k}\big(\varphi_1 \vee \bigcirc(\varphi_1\,\boldsymbol{R}\,\varphi_2)\big)
\end{array}$$

As mentioned above, this finite trace semantics is defined such that a formula is satisfied if and only if there have been no violations in the trace and there are no unfulfilled obligations after reading the entire trace, e.g. $\Diamond\varphi$ where φ was not yet satisfied.
The above satisfaction semantics for the basic LTL operators correspond to the ones in [10], the PLTL operators are defined in the same way as in [12] and the first-order component has been adapted from [7]. The interpretations of the boolean connectives $\neg, \wedge, \vee$ and $\rightarrow$ also directly correspond to the standard predicate logic semantics and the definition of the derived parametric operators supports the intuitive meaning of bounded temporal operators that is introduced in [2].

Theorem 1.1 (Existence of positive normal form)**.** *For every FO-PLTL formula* φ *there exists an equivalent (wrt.* $\models$*) FO-PLTL formula* φ' *in* positive normal form *(PNF), i.e. such that the negation operator* $\neg$ *only appears in front of atomic formulae, there are no double-negations and no derived operators.*

The monitoring approach presented in this thesis only works with FO-PLTL formulae in positive normal form. Altough for the derived operators, the transformation into PNF makes a formula grow exponentially, this has no greater effect on the monitoring algorithm, since every subformula is considered only once. The usage of PNF in the monitoring algorithm is also the reason for explicitly defining the operator-semantics of $\wedge$, $\boldsymbol{R}$ and $\exists$, even though they can be derived from the other operators.
For ground formulae, free variables only appear in quantified subformulae, which means that satisfaction of a ground formula φ by a trace σ under a parameter-valuation α can be defined as follows:

$$(\sigma, \alpha) \models \varphi \qquad := \qquad (\sigma, 0, \alpha, \beta_\emptyset) \models \varphi$$

where $\beta_\emptyset$ denotes the variable-assigment that is undefined for every variable in $\mathcal{X}$.
With the above definition the meaning of every component of the FO-PLTL formulae is formalized. The key feature of FO-PLTL monitoring, though, is not about checking if a specification is satisfied under a given parameter-valuation, but rather about measuring temporal behaviour by determining optimal values for these parameters.

1.3.4 Measuring First-order PLTL

Properties. It is easily derivable from the above semantics, that there is not always a unique parameter-valuation for which the formula is satisfied. In fact, there are multiple factors that contribute to this unambiguity of FO-PLTL.
First of all is every parametric operator *upward* or *downward monotone* ([2]), e.g. the operator $\diamond_{\leq k}$ is upward monotone since if satisfaction holds for $[k \mapsto n]$, it holds for every $[k \mapsto n']$ with a greater n', as well. This follows the intuitive meaning the operator: Whenever some property holds in at most n, steps, it naturally holds in at most "more than n" steps, as well. $\Box_{\leq k}$, though, is downward monotone, since whenever a property holds for the next n steps, it naturally holds for the next $n' < n$ steps, as well. The same is the case for every operator that is derived from one of those two. Negated parametric formulae have respective opposing monotinicity behaviour, as illustrated in A.6 and A.7. From now on, we assume that every formulae is in positive normal form, so we know that $\diamond_{\leq k}$ behaves upward and $\Box_{\leq k}$ behaves downward monotonously.
Because of this property, choosing arbitrarily large values for parameters that appear in the $\diamond_{\leq k}$ operator and 0 for the ones that appear in $\Box_{\leq k}$ is a way to obtain satisfaction (in most cases). In order to get detailed information about the temporal behaviour of our system, though, we are aiming to get *optimal* parameter values. Therefore, for the $\diamond_{\leq k}$ operator, we want to minimize the value for k, such that satisfaction is given and for $\Box_{\leq k}$ we want to maximize the value, respectively.
Restrictions. With the monotinicity of the parametric operators, a problem arises: In both PTLT and FO-PLTL, we must not use the same parameter in both upward and downward monotone operators. Allowing this, we would face the problem that the existence of a parameter-valuation under which the specification is satisfied, becomes *undecidable*, as it was shown in [2]. Furthermore, in order to ensure efficient monitoring, we use a parameter only to measure the satisfaction of one unique subformula, i.e. for every FO-PLTL formulae φ, φ' and parameter k holds:

$$\diamond_{\leq k}\varphi' \text{ subformula of } \varphi \Rightarrow \forall \varphi''. \Box_{\leq k}\varphi'' \text{ not a subformula of } \varphi \tag{1.1}$$

$$\Box_{\leq k}\varphi' \text{ subformula of } \varphi \Rightarrow \forall \varphi''. \diamond_{\leq k}\varphi'' \text{ not a subformula of } \varphi \tag{1.2}$$

$$\diamond_{\leq k}\varphi' \text{ subformula of } \varphi \Rightarrow \forall \varphi''. (\diamond_{\leq k}\varphi'' \text{ subformula of } \varphi \Rightarrow \varphi' = \varphi'') \tag{1.3}$$

$$\Box_{\leq k}\varphi' \text{ subformula of } \varphi \Rightarrow \forall \varphi''. (\Box_{\leq k}\varphi'' \text{ subformula of } \varphi \Rightarrow \varphi' = \varphi'') \tag{1.4}$$

We denote the two disjunct sets of parameters k that occur in a $\diamond_{\leq k}$ or $\Box_{\leq k}$ operator as $\mathcal{K}_{\varphi,\uparrow}$ or $\mathcal{K}_{\varphi,\downarrow}$, respectively. Furthermore, we define the set of parameters occuring in a formula φ altogether as $\mathcal{K}_{\varphi} := \mathcal{K}_{\varphi,\downarrow} \cup \mathcal{K}_{\varphi,\uparrow}$. Additionally, we assume that every parameter-valuation is only defined for the parameters that occur in the specification we currently investigate. Whenever φ is clear from context, it is ommited in the subscipt.
We call a parameter-valuation that is maximized for upward monotone and minimized for downward monotone parameters a *measure*. In order to introduce this measure-property formally, we first define the the *quality* of a parameter-valuation:

Definition 1.8 (Valuation quality). *For any FO-PLTL formula φ and parameter-valuations α_1 and α_2, we define $\sqsupseteq_{\varphi}$ such that $\alpha_1 \sqsupseteq_{\varphi} \alpha_2$ if and only if*

$$\forall k \in \mathcal{K}_{\varphi,\uparrow}.\ \alpha_1(k) \leq \alpha_2(k) \qquad \textit{and} \qquad \forall k \in \mathcal{K}_{\varphi,\downarrow}.\ \alpha_1(k) \geq \alpha_2(k)$$

We further define $\alpha_1 \sqsupset_{\varphi} \alpha_2$ if and only if $\alpha_1 \sqsupseteq_{\varphi} \alpha_2$ and $\alpha_1 \neq \alpha_2$.

$$\begin{tikzcd}
 & \{p(1),p(2),q(1)\} \arrow[dl] & & \{q(2)\} \arrow[d] \\
\Diamond_{\leq 0} q(1) \arrow[d] & & & \Diamond_{\leq 0} q(2) \arrow[dll] \\
\Diamond_{\leq 0}\Diamond_{\leq 0} q(1) \arrow[dr] & \Diamond_{\leq 1}\Diamond_{\leq 0} q(2) \arrow[d] & & \\
 & \forall x : p.\Diamond_{\leq 1}\Diamond_{\leq 0} q(x) \arrow[dr] & & \forall x : p.\Diamond_{\leq 0}\Diamond_{\leq 0} q(x) \arrow[dl] \\
 & & \Box(\forall x : p.\Diamond_{\leq 1}\Diamond_{\leq 0} q(x)) &
\end{tikzcd}$$

$$\begin{tikzcd}
 & \{p(1),p(2),q(1)\} \arrow[dl] & & \{q(2)\} \arrow[dll] \\
\Diamond_{\leq 0} q(1) \arrow[d] & \Diamond_{\leq 1} q(2) \arrow[d] & & \\
\Diamond_{\leq 0}\Diamond_{\leq 0} q(1) \arrow[dr] & \Diamond_{\leq 0}\Diamond_{\leq 1} q(2) \arrow[d] & & \\
 & \forall x : p.\Diamond_{\leq 0}\Diamond_{\leq 1} q(x) \arrow[dr] & & \forall x : p.\Diamond_{\leq 0}\Diamond_{\leq 0} q(x) \arrow[dl] \\
 & & \Box(\forall x : p.\Diamond_{\leq 0}\Diamond_{\leq 1} q(x)) &
\end{tikzcd}$$

Figure 1.3: Ambiguity of measures

By definition, the relations $\sqsupset_\varphi$ and $\sqsupseteq_\varphi$ are in general *partial* orderings on parameter-valuations. Again, Whenever clear from context, the subscript φ is omitted. The quality of a parameter-valuation is now used to describe the measure-property formally:

Definition 1.9 (Measure)**.** *For every FO-PLTL formula φ, trace σ and variable-assignment β, a parameter-valuation α is called a* measure *for φ over σ under β if and only if $(\sigma, 0, \alpha, \beta) \models \varphi$ and $(\sigma, 0, \alpha', \beta) \not\models \varphi$ for all α' where $\alpha' \sqsupset \alpha$. We denote the set of all measures for φ over σ under β as $M_{\varphi,\sigma,\beta}$.*

It is proven that even for propositional PLTL, measuring of a specification is unambiguous, meaning that there is not always a *unique* measure for every satisfiable specification ([12]). This carries over to the first-order case, as well:

Theorem 1.2 (Ambiguity of measures)**.** *There exist φ, σ and β, such that $|M_{\varphi,\sigma,\beta}| > 1$.*

Example 1.3.1. *Let φ be the First-order PLTL formula $\Box(\forall x{:}p.\, \Diamond_{\leq k}\Diamond_{\leq k'} q(x))$ and σ be the trace $\{p(1), p(2), q(1)\}\{q(2)\}$. As depicted in Figure 1.3, for $x \mapsto 1$, $q(x)$ holds in the same step as $p(x)$ and therefore we can minimize both k and k' to 0. But for $x \mapsto 2$, we have two ways. Either we set $k \mapsto 0$ and $k' \mapsto 1$ or we choose $k \mapsto 1$ and $k' \mapsto 0$. Both valuations are measures of φ over σ under $\beta_\emptyset$.*

This example briefly displays another characteristic of measures: Although the subformula $\Diamond_{\leq k}\Diamond_{\leq k'} q(x)$ is satisfied by σ under $[k \mapsto 0, k' \mapsto 0]$ and variable-assignment $[x \mapsto 1]$, it is not a measure of φ. We have to find a valuation, under which satisfaction is given for all instantiations of the quantifier. More generally, we need to consider every "measuring instance" to get a measure for the entire formula, i.e. the measure at every point in time and for every quantifier instantiation, where the parametric subformula needs to be satisfied. The monitoring algorithm in the next chapter measures the instantiations of a quantifier separately. Therefore, we need to define a way to combine parameter-valuations that maintains satisfaction and parameter-quality:

Definition 1.10 (Conjunction on valuations)**.** *For two of parameter-valuations α_1 and α_2, we define the* conjunction *of valuations $\cap$ as follows:*

$$(\alpha_1 \cap \alpha_2)(k) \quad = \quad \begin{cases} \max\{\alpha_1(k), \alpha_2(k)\} & \textit{if } k \in \mathcal{K}_\uparrow \\ \min\{\alpha_1(k), \alpha_2(k)\} & \textit{if } k \in \mathcal{K}_\downarrow \end{cases}$$

1.3.5 Unambiguous semantics

In general, we want to avoid ambiguity of measuring results. Especially in the first-order case when it comes to existential quantification, the number of measures that need to be considered, can explode easily. For this purpose, we aim to modify the measuring semantics of FO-PLTL in such a way that we obtain a *unique* result, whenever a specification is satisfiable. We will present two different ways to reach this goal, the latter one being used in our monitoring algorithm.

Unambiguity through parameter priorities

One way to obtain a unique measure is used in the offline monitoring algorithm in [12] and uses a precedence, under which parameter values need to be optimized primarily. This precedence is given by a total *priority ordering* on the parameters in $\mathcal{K}$. This ordering then induces a total quality-ordering on paramater-valuations (and therefore measures) that is used to determine the optimal measure.

Definition 1.11 (Optimal measure with regard to paramater priority).
Let $\succ$ be a total order over $\mathcal{K}$ and k be the maximal element of $\mathcal{K}$ with respect to $\succ$. We define for every FO-PLTL formula the ordering $\underline{\unrhd}$ over parameter-valuations inductively as follows: For parameter-valuations α_1 and α_2, $\alpha_1 \underline{\unrhd} \alpha_2$ holds if

- *$k \in \mathcal{K}_\uparrow$ and $\alpha_1(k) < \alpha_2(k)$,*
- *$k \in \mathcal{K}_\downarrow$ and $\alpha_1(k) > \alpha_2(k)$ or*
- *$\alpha_1(k) = \alpha_2(k)$ and $\alpha_1 \setminus k \underline{\unrhd} \alpha_2 \setminus k$.*

We further define $\alpha_1 \unrhd \alpha_2$ if and only if $\alpha_1 \underline{\unrhd} \alpha_2$ and $\alpha_1 \neq \alpha_2$. We refer to the maximal measure (with respect to $\unrhd_\varphi$) for a formula φ over a trace σ under a variable-assignment β as the optimal measure *for φ over σ with parameter-priority $\succ$.*

As mentioned above, $\unrhd$ is a *total* ordering on parameter-valuation that extends $\sqsupset$ by using $\succ$ whenever α and α' are incomparable with respect to $\sqsupset$. In particular, this means that for all priorities $\succ$, we have that $\alpha \sqsupset \alpha' \Rightarrow \alpha \unrhd \alpha'$.

Example 1.3.2. *We consider the formula and trace used in Example 1.3.1 and assume that $k \succ k'$ holds. The two determined measures are incomparable wrt. $\sqsupset$, but by definition of $\succ$, we give priority to the optimization of k and obtain $[k \mapsto 0, k' \mapsto 1] \unrhd [k \mapsto 1, k' \mapsto 0]$. We then choose the valuation that is maximal wrt. $\unrhd$, i.e. the measure that has the minimal value for k and therefore obtain the optimal measure $[k \mapsto 0, k' \mapsto 1]$.*

Unambiguous semantics

A different way to obtain unique measures is by modifying the FO-PLTL semantics altogether. As seen in [12], for (propositional) PLTL, it is sufficient to specify the behaviour concerning disjunction, the Until, the Release and the Eventually operator. In FO-PLTL, we also need to alter the semantics for the existential quantification.
When in comes to disjunction, for example, considering a specification $\varphi_1 \vee \varphi_2$, it is easily observable that whenever both φ_1 and φ_2 are satisfied, the chance of obtaining multiple measures is given. Similarly for formulae like $\varphi_1 \boldsymbol{U} \varphi_2$, whenever φ_2 holds multiple

times while φ_1 keeps being satisfied, it may depend on the considered point at which φ holds, how the measure looks like. Per definition, therefore the same holds for the non-parametric $\Diamond$-operator, as you can see in [12] (page 5, section "Unique Measures"). Since guarded existential quantification behaves similar to a finite disjunction, we can get different measures for $\exists x{:}p.\,\varphi$ whenever φ is satisfiable under $\beta[{}^{d}/{}_{x}]$ for multiple d. To tackle this ambiguous measuring behaviour, we introduce an *unambiguous semantics* by adapting the definition introduced in [12] and extending it to cover ambiguity of measuring existential quantifications. To obtain unique measures, we now give priority in these kinds of formulae, i.e. for $\varphi_1 \vee \varphi_2$ we consider φ_2 only if φ_1 cannot be satisfied for any parameter-valuation. For $\varphi_1 \, \boldsymbol{U} \, \varphi_2$, we measure to the *first* occurrence where φ_2 can be satisfied and similarly for $\varphi_1 \, \boldsymbol{R} \, \varphi_2$ the first position where φ_1 and φ_2 can. For $\exists x{:}p.\,\varphi$, we use the approach introduced on the previous page and determine the optimal d by looking at the (unique) measures for $\varphi[{}^{d}/{}_{x}]$ and choosing the one that is optimal with respect to a given priority of the parameters that occur in φ. To express the notion of "can or cannot be satisfied" that we use in this approach, we define the FO-LTL-abstraction (see also [12]):

Definition 1.12 (FO-LTL-abstraction)**.** *For a FO-PLTL formula φ, the* FO-LTL-abstraction *$[\varphi]$ is defined inductively as follows:*

$$\begin{array}{rcl@{\qquad}rcl}
[\boldsymbol{true}] & = & \boldsymbol{true} & [\varphi_1 \, \boldsymbol{U} \, \varphi_2] & = & [\varphi_1] \, \boldsymbol{U} \, [\varphi_2] \\
[p(t_1,\dots,t_n)] & = & p(t_1,\dots,t_n) & [\varphi_1 \, \boldsymbol{R} \, \varphi_2] & = & [\varphi_1] \, \boldsymbol{R} \, [\varphi_2] \\
[\neg p(t_1,\dots,t_n)] & = & \neg p(t_1,\dots,t_n) & [\Diamond_{\le k}\varphi] & = & \Diamond[\varphi] \\
[\varphi_1 \wedge \varphi_2] & = & [\varphi_1] \wedge [\varphi_2] & [\Box_{\le l}\varphi] & = & [\varphi] \\
[\varphi_1 \vee \varphi_2] & = & [\varphi_1] \vee [\varphi_2] & [\forall(x_1,\dots,x_n){:}p.\,\varphi] & = & \forall(x_1,\dots,x_n){:}p.\,[\varphi] \\
[\bigcirc\varphi] & = & \bigcirc[\varphi] & [\exists(x_1,\dots,x_n){:}p.\,\varphi] & = & \exists(x_1,\dots,x_n){:}p.\,[\varphi]
\end{array}$$

The FO-LTL-abstraction of a formula φ eliminates all parametric operators and substitutes in a way, such that $[\varphi]$ is satisfied if and only if there exists a parameter-valuation, under which φ is satisfied:

Theorem 1.3 (Satisfiability of the FO-LTL abstraction)**.** *For every φ, σ and β holds:*

$$\forall\alpha.(\sigma,0,\alpha,\beta) \models [\varphi] \qquad \textit{iff} \qquad \exists\alpha.(\sigma,0,\alpha,\beta) \models \varphi$$

Since $[\varphi]$ is parameter-free by construction, we know that satisfaction does not depend on the choice of α. In particular, this means that if $\neg[\varphi]$ is satisfied under some valuation, φ cannot be satisfied under any valuation. We now use this abstraction to formally define the unambiguous semantics that is described above.

Definition 1.13 (Unambiguous FO-PLTL formula satisfaction)**.** *Let a total ordering $\succ$ on $\mathcal{K}$ be given. We define the satisfaction relation $\models$* under unambiguous semantics *by adapting the definition of $\models$ from FO-PLTL formula satisfaction and modifying only the semantics for the following operators:*

$(\sigma,i,\alpha,\beta) \models (\varphi_1 \vee \varphi_2)$	*iff*	$(\sigma,i,\alpha,\beta) \models \varphi_1$ *or* $(\sigma,i,\alpha,\beta) \models \neg[\varphi_1] \wedge \varphi_2$
$(\sigma,i,\alpha,\beta) \models \varphi_1 \, \boldsymbol{U} \, \varphi_2$	*iff*	*there exists j, $i \le j < \lvert\sigma\rvert$, such that $(\sigma,j,\alpha,\beta) \models \varphi_2$ and $(\sigma,j',\alpha,\beta) \models \varphi_1 \wedge \neg[\varphi_2]$ for each j', $i \le j' < j$*
$(\sigma,i,\alpha,\beta) \models \varphi_1 \, \boldsymbol{R} \, \varphi_2$	*iff*	*for each i', $i \le i' < n$. $(\sigma,i',\alpha,\beta) \models \neg[\varphi_1] \wedge \varphi_2$ or there exists j, $i \le j < n$ such that $(\sigma,j,\alpha,\beta) \models \varphi_1 \wedge \varphi_2$ and $(\sigma,j',\alpha,\beta) \models \neg[\varphi_1] \wedge \varphi_2$ for each j', $i \le j' < j$*
$(\sigma,i,\alpha,\beta) \models \Diamond_{\le k}\varphi$	*iff*	*there exists j with $0 \le j \le \alpha(k)$ and $i + j < \lvert\sigma\rvert$, such that $(\sigma,i+j,\alpha,\beta) \models \varphi$ and $(\sigma,i+j',\alpha,\beta) \models \neg[\varphi]$ for each j', $0 \le j' < j$*

$(\sigma, i, \alpha, \beta) \models \exists(x_1, ..., x_n){:}p.\,\varphi$ *iff* *for some* $d \in \sigma_p[i]$, $\big(\sigma, i, \alpha, \beta[{}^{d}/{}_{(x_1,\dots,x_n)}]\big) \models \varphi$ *and for each* α' *where* $\alpha' \trianglerighteq \alpha$ *holds that* $\big(\sigma, i, \alpha', \beta[{}^{d'}/{}_{(x_1,\dots,x_n)}]\big) \not\models \varphi$ *for every* $d' \in \sigma_p[i]$

Whenever necessary, we use $\models^{\succ}$ to make clear that we consider satisfaction under the unambiguous semantics.
It is important to notice that a measure under unambiguous semantics may not be a measure under the standard semantics from Definition 1.7, as is illustrated in [12] (p.15) for propositional PLTL. Nevertheless, we can show that the restrictions in the unambiguous semantics only modify the *measuring* behaviour, not satisfaction in general, i.e. whenever we can achieve satisfaction under the standard FO-PLTL semantics, we can find a parameter-valuation which is satisfying under the unambiguous semantics:

Theorem 1.4 (Existence of a satisfying valuation under unambiguous semantics)**.** *For every FO-PLTL formula φ, trace σ, variable-assignment β and total ordering $\succ$ on $\mathcal{K}$*

$$\exists\alpha.\,(\sigma, 0, \alpha, \beta) \models \varphi \qquad \textit{iff} \qquad \exists\alpha'.\,(\sigma, 0, \alpha', \beta) \models^{\succ} \varphi$$

The purpose of the modifications we made in the unambiguous semantics compared to the standard FO-PLTL semantics is to obtain an unanbiguous measuring behaviour. Although the notion is already briefly explained above, we still need to prove that measures under the unambiguous semantics are, in fact, *unique*:

Theorem 1.5 (Uniqueness of the measure under unambiguous semantics)**.** *For every First-order PLTL formula φ, trace σ, variable-assignment β and total ordering $\succ$ on $\mathcal{K}$ holds: If α is measure for φ over σ under β, then it is unique, i.e. there is no different parameter-valuation $\alpha' \neq \alpha$ that is also a measure.*

If this unique measure exists, we will refer to it as the *unambiguous measure* with respect to $\succ$ for φ over σ under β and denote it as $\alpha^{\succ}_{\varphi,\sigma,\beta}$.

Summary

In this chapter, we have introduced First-order Parametric LTL, a temporal logic with measuring capabilities that can express data-aware specifications for monitoring or model-checking purposes. We have stated some semantic restrictions – e.g. guarded quantification or the requirement every parameter must not appear in both upward and downward monotone operators – that are neccessary to ensure efficient monitoring.
We have also seen that in general, there is not a unique measuring result for a given specification and trace and further information (e.g. parameter priorities) or even semantic modifications (see unambiguous semantics) are needed to obtain an explicit measure.
By presenting the formal definition of First-order Parametric LTL, we have laid the foundation of the monitoring algorithm and can therefore start its construction in the following chapter.

Chapter 2

A monitor for First-order PLTL

We will adapt the online-monitoring algorithms presented in [12] and [7] to construct a monitor for First-order Parametric LTL, since both use a similar approach. The monitor will be based on a special kind of *automata* that accepts an input – i.e. a trace – if and only if the trace satisfies the formula. In [12], so called *measuring automata* were introduced, which maintain counter variables for the parameters to be measured. The algorithm in [7] uses *spawning automata* to handle the first-order component of the logic, by spawning a new automaton for every instantiation of a quantification and modifying the accepting condition accordingly.
Our monitoring algorithm uses *atoms* (in [7] they are called *complete subsets* of the closure), i.e. sets of subformulae, as states for the automata. Intuitively speaking, such an atom contains every subformula that is satisfied at the current position in the trace. By defining *consistency* conditions on such atoms and constructing the transition relation accordingly, violations with respect to the satisfaction semantics are ruled out and therefore every atom represents a possibly satisfying future behaviour. Since there is in general no unique way of satisfying a specification in LTL or PLTL, these automata are nondeterministic. For the unambiguous semantics of PLTL, a deterministic measuring automaton can used, whose states are sets of such atoms. ([12]) In First-order LTL, a nondeterministic transition relation is used, meaning that in every step, possibly different successor states can be reached and the existence of an accepting run is checked. ([7])
Each run maintains counters that are used to determine a measure for the given specification. Since the meausuring behaviour is in general ambiguous for the standard FO-PLTL semantics, we will develop an algorithm that computes the *unambiguous measure* for a given specification and input-sequence. As mentioned above, we will use a combination of the two automata presented in [12] and [7] and introduce a new kind of automaton.
We will slightly modify the notation introduced in the original algorithms and automata constructions to provide optimal readability in this thesis. Nevertheless, we will present every aspect of the monitoring algorithm formally and prove its correctness.

2.1 Measuring-spawning automata

In this section, we will formalize the type of automaton that is used for monitoring FO-PLTL. We adapt the definition from spawning automata (see [7]) and measuring automata (see [12]). For an arbitraty set X, let $\mathcal{B}^+(X)$ denote the set of all positive boolean formulae over X, i.e. conjunctions and disjunctions of elements of X. For a subset $Y \subseteq X$ and $b \in \mathcal{B}^+(X)$, we write $Y \Vdash b$ if and only if the truth assignment that assigns true to all elements in Y and false to all elements in $X \setminus Y$, satisfies b under the standard boolean semantics. (see also [7])

Definition 2.1 (Measuring-spawning automaton)**.** *A* measuring-spawning automaton *(ms-automaton) $\mathcal{A}$ is a tuple $(\Sigma, l, Q, q_0, P, \theta, \delta, \gamma, \lambda, F)$ where*

- Σ *is an* input alphabet,
- $l \in \mathbb{N}$ *is a* level,
- Q *is a set of* states, $q_0 \in Q$ *the* initial state,
- P *is a set of* measuring variables, $\theta : P \rightarrow \mathbb{N}$ *is an* initial measuring-assignment,
- $\delta : Q \times \Sigma \rightarrow 2^Q$ *is a* transition relation,
- $\gamma : Q \times Q \rightarrow (P \rightarrow \mathbb{N}) \rightarrow (P \rightarrow \mathbb{N})$ *is an* update function,
- $\lambda : \Sigma \times Q \rightarrow \mathcal{B}^+(\mathcal{A}^{<l})$ *is a* spawning function,
- $F \subseteq Q$ *is a set of* final states,

where $\mathcal{A}^{<l}$ denotes the set of spawning measuring automata with a level smaller than l.

Definition 2.2 (Run of a spawning measuring automaton)**.** *A* run *of a spawning measuring automaton $\mathcal{A} = (\Sigma, l, Q, q_0, P, \theta, \delta, \gamma, \lambda, F)$ over an input sequence $\sigma = \sigma_0\sigma_1...\sigma_{n-1} \in \Sigma^*$ is a sequence $\pi = (s_0, \eta_0)(s_1, \eta_1)...(s_n, \eta_n)$ of configuations, where each configuration is a pair (s_i, η_i) consisting of a state $s_i \in Q$ and a measuring-assignment $\eta_i : P \rightarrow \mathbb{N}$, such that*

- $s_0 = q_0$, $\eta_0 = \theta$,
- $s_{i+1} \in \delta(s_i, \sigma_i)$ *for* $i = 0, ..., n-1$ *and*
- $\eta_{i+1} = \gamma(s_i, s_{i+1})(\eta_i)$ *for* $i = 0, ..., n-1$.

We call $\lambda(\sigma_i, s_{i+1})$ the (spawning) obligation *at point i. π is called* locally accepting, *if $s_n \in F$. If $l = 0$, π is called* accepting *if it is locally accepting. If $l > 0$, it is called accepting if for each $i = 0, ..., n-1$ there exists a set $Y \subseteq A^{<l}$ such that $Y \Vdash \lambda(\sigma_i, s_{i+1})$ and all $\mathcal{A}' \in Y$ have an accepting run π' over $\sigma_i...\sigma_{n-1}$. The set of all accepting runs π of $\mathcal{A}$ over σ is denoted as $\Pi_{\mathcal{A}}(\sigma)$.*

2.2 Atoms and successors

Atoms. As it has been briefly mentioned in the introduction of this chapter, we will use sets of subformulae of the given specification, so called *atoms*, as states in the monitoring automata. These sets will contain the subformulae that are satisfied at the time they are visited or are assumed to be satisfied by some future behaviour. To handle the case of quantified formulae, we use the spawning capabilities of the ms-automata to spawn a new automaton for every instantiation of the quantification, e.g. whenever a formula $\forall x{:}p.\varphi$ is assumed to be satisfied, we spawn a new automaton that measures φ under $\beta[^d/_x]$ for

every $d \in \sigma_p[i]$ and then use the acceptance condition of ms-automata to make sure that every such "assumption" of satisfaction is, in fact, the case. Therefore, we do not need to consider the subformulae of φ in the "parent" automaton.
The measuring component of the ms-automaton uses this invariant to update the counters for every parameter in the specification, whenever some parametric formula is present in the current atom. In general, we make the assumption that whenever an atom does not contain a subformula, it contains its negation – also in PNF. Since we need to avoid measuring negations of parametric subformula, i.e. we do not measure k in $\Box_{\leq k}\neg\varphi$ whenever $\Diamond_{\leq k}\varphi$ is not in the atom, we consider only the negations of non-parametric subformulae. We are, in order to follow the unambiguous semantics, reliant upon the FO-LTL abstraction and therefore we consider every subformula of the specification's abstraction and its respective negation, as well. The set of all such considered subformulae is called the *closure*.
In the following, we will denote the given specification that is to be monitored as Φ and use φ for its subformulae.

Definition 2.3 (Closure)**.** *For a First-order PLTL formula Φ, we define the* closure $cl(\Phi)$ *of Φ to be the set that contains every subformula of Φ or $[\Phi]$ that does not occur inside the scope of a quantifier and the respective negation of every non-parametric such formula.*

As mentioned before, we will treat formulae that start with a quantifier as if they were propositions, i.e. either absent or present. We will now define *atoms* so that they follow the FO-PLTL satisfaction semantics:

Definition 2.4 (Atom)**.** *An* atom *of a First-order PLTL formula Φ is a set $A \subseteq cl(\Phi)$ that has the following properties:*

1. Local consistency with respect to propositional logic, *i.e.:*

$$(\varphi_1 \wedge \varphi_2) \in A \Leftrightarrow (\varphi_1 \in A) \wedge (\varphi_2 \in A) \tag{At.1.1}$$
$$(\varphi_1 \vee \varphi_2) \in A \Leftrightarrow (\varphi_1 \in A) \vee (\varphi_2 \in A) \tag{At.1.2}$$
$$\varphi \in A \Rightarrow \neg\varphi \notin A \tag{At.1.3}$$
$$\boldsymbol{true} \in cl(\Phi) \Rightarrow \boldsymbol{true} \in A \tag{At.1.4}$$

2. Local consistency with respect to the Until operator, *i.e.:*
   For each $(\varphi_1 \,\boldsymbol{U}\, \varphi_2) \in cl(\Phi)$ holds:

$$\varphi_2 \in A \Rightarrow (\varphi_1 \,\boldsymbol{U}\, \varphi_2) \in A \quad \textit{and} \quad \big((\varphi_1 \,\boldsymbol{U}\, \varphi_2) \in A \ \wedge\ \varphi_2 \notin A\big) \Rightarrow \varphi_1 \in A \tag{At.2}$$

3. Local consistency with respect to the Release operator, *i.e.:*
   For each $(\varphi_1 \,\boldsymbol{R}\, \varphi_2) \in cl(\Phi)$ holds:

$$(\varphi_1 \,\boldsymbol{R}\, \varphi_2) \in A \Rightarrow \varphi_2 \in A \quad \textit{and} \quad \big(\varphi_1 \in A \ \wedge\ \varphi_2 \in A\big) \Rightarrow (\varphi_1 \,\boldsymbol{R}\, \varphi_2) \in A \tag{At.3}$$

4. Local consistency with respect to the parametric Globally operator, *i.e.:*
   For each $\Box_{\leq k}\varphi \in cl(\Phi)$ holds:

$$\Box_{\leq k}\varphi \in A \Rightarrow \varphi \in A \tag{At.4}$$

5. Unambiguity with respect to disjunction, *i.e.:*
   For each $(\varphi_1 \vee \varphi_2) \in A$ *holds:*

$$[\varphi_1] \in A \wedge \varphi_1 \in A \qquad \textit{or} \qquad \neg[\varphi_1] \in A \wedge \varphi_2 \in A \tag{At.5}$$

6. Unambiguity with respect to the Until operator, *i.e.:*
   For each $(\varphi_1 \, \boldsymbol{U} \, \varphi_2) \in A$ *holds:*

$$\varphi_1 \in A \wedge \neg[\varphi_2] \in A \qquad \textit{or} \qquad \varphi_2 \in A \tag{At.6}$$

7. Unambiguity with respect to the Release operator, *i.e.:*
   For each $(\varphi_1 \, \boldsymbol{R} \, \varphi_2) \in A$ *holds:*

$$\neg[\varphi_1] \in A \wedge \varphi_2 \in A \qquad \textit{or} \qquad \varphi_1 \in A \wedge \varphi_2 \in A \tag{At.7}$$

8. Maximality, *i.e. for every non-parametric* $\varphi \in cl(\Phi)$*:*

$$\varphi \in A \qquad \textit{or} \qquad \neg\varphi \in A \tag{At.8}$$

We will write At_Φ *to denote the set of atoms of* Φ.

Successors. Our monitor processes subsequent events step by step to determine violations and update measurements. To describe the transitions between atoms, we need a successor relation that processes an event corresponding to the semantic definition of the considered subformulae. Thereby, the invariant mentioned above saying that the current atom contains every subformula which is satisfied at the time it is "visited", is maintained:

Definition 2.5 (Successor relation)**.** *For each* Φ *and* β*, let* $\rightarrow_\beta \subseteq At_\Phi \times E \times At_\Phi$ *be a succesor relation between atoms of* Φ*. For each* $\tau, \tau' \in At_\Phi$ *and* $e \in E$*, we have* $\tau \xrightarrow{e}_\beta \tau'$ *if* τ' *is the* smallest *set such that the following conditions hold:*

$$\forall . p(t_1, ..., t_n) \in At_\Phi \text{ where } p \in \mathcal{P}^U.\ p(t_1, ..., t_n) \in \tau' \Leftrightarrow \mathbb{I}_\beta\big(p(t_1, ..., t_n)\big) \in e \tag{S.1}$$

$$\forall . p(t_1, ..., t_n) \in At_\Phi \text{ where } p \in \mathcal{P}^I.\ p(t_1, ..., t_n) \in \tau' \Leftrightarrow \mathbb{I}_\beta\big(p(t_1, ..., t_n)\big) \in \mathbb{I} \tag{S.2}$$

$$\bigcirc\varphi \in \tau \quad \Rightarrow \quad \varphi \in \tau' \tag{S.3}$$

$$(\varphi_1 \, \boldsymbol{U} \, \varphi_2) \in \tau \quad \Rightarrow \quad (\varphi_2 \in \tau) \vee \big(\varphi_1 \in \tau \wedge (\varphi_1 \, \boldsymbol{U} \, \varphi_2) \in \tau'\big) \tag{S.4}$$

$$(\varphi_1 \, \boldsymbol{R} \, \varphi_2) \in \tau \quad \Rightarrow \quad (\varphi_2 \in \tau) \wedge \big(\varphi_1 \in \tau \vee (\varphi_1 \, \boldsymbol{R} \, \varphi_2) \in \tau'\big) \tag{S.5}$$

$$\Diamond_{\leq k}\varphi \in \tau \quad \Rightarrow \quad (\varphi \in \tau) \vee (\Diamond_{\leq k}\varphi \in \tau') \tag{S.6}$$

The definition of this successor relation corresponds to the expansion laws for the LTL operators that can be found in [3] (p. 248) or [14] and has already been introduced similarly in [12] and [7]. We will drop the subscript β whenever clear from context.

2.3 Construction of the monitoring automaton

We now begin the construction of the automata that are used to monitor a FO-PLTL specification. As mentioned above, we treat a quantification just like an atomic proposition and check satisfaction using the spawned sub-automata. It is easy to see that formulae containing no quantification can be monitored using the same approach as presented in [12].
We now define the monitoring automaton for a FO-PLTL formula Φ under a variable assignment β, formally $\mathcal{A}_{\Phi,\beta} := (\Sigma, I, Q, q_0, P, \theta, \delta, \gamma, \lambda, F)$. The automaton takes care of different parts of the monitoring tasks: it checks satisfaction, measures optimal results and spawns sub-automata. Each of these components will be defined in the following:

LTL Component

The LTL component of the monitor takes care of checking *satisfiability* of the specification, i.e. whether or not some parameter-valuation exists, under which satisfaction is given. Since our monitor processes events, we first of all set the input alphabet to the set of possible events, i.e.

$$\Sigma \quad := \quad E \tag{$\mathcal{A}$.1}$$

As mentioned above, we use the atoms as states of the automaton, aiming to get the invariant that an atom is visited at some point in an accepting run whenever all contained formulae are satisfied at that point in the trace. Therefore, similar to [12], we use a dedicated initial state which is not an atom:

$$Q \quad := \quad At_\Phi \cup \{q_0\} \tag{$\mathcal{A}$.2}$$

The transition relation δ is now defined in such a way that the above invariant is maintained. For this purpose, we use the successor-relation that we defined above. Since the initial state is not an atom, we define its successors to be the atoms that contain the monitored formula φ and that are consistent with the first event of the trace. In fact, altough q_0 is not an atom, it behaves just like an atom that contained only $\bigcirc\Phi$. Formally, for every $\tau \in Q$ and $e \in E$

$$\delta(\tau, e) := \begin{cases} \{\tau' \in At_\Phi \mid \Phi \in \tau' \text{ and conditions S.1, S.2 hold}\} & \text{if } \tau = q_0 \\ \{\tau' \in At_\Phi \mid t \overset{e}{\to} \tau'\} & \text{otherwise} \end{cases} \tag{$\mathcal{A}$.3}$$

Since satisfaction at some point may depend on future behaviour, every run makes assumptions about the future behaviour that is later satisfied during the monitoring process or not. In order to correllate with the semantics of FO-PLTL, we must define the accepting states such that no more unfulfilled temporal obligations are contained, i.e.:

$$F := \{\tau \in At_\Phi \mid \bigcirc\varphi \notin \tau \;\wedge\; \Diamond_{\leq k}\varphi \in \tau \Rightarrow \varphi \in \tau \;\wedge\; (\varphi_1 \,\boldsymbol{U}\, \varphi_2) \in \tau \Rightarrow \varphi_2 \in \tau\} \tag{$\mathcal{A}$.4}$$

Measuring Component

We want our monitor to calculate measures for the given formula over the input trace. In order to calculate these measures, we maintain counters for every parameter $k \in \mathcal{K}_\varphi$ that are updated in every step:

Example for $k \in \mathcal{P}_\uparrow$:

	$\cdots$	$\Diamond_{\leq k} a$, $\neg a$	$\Diamond_{\leq k} a$, $\neg a$	$\Diamond_{\leq k} a$, a	a	$\Diamond_{\leq k} a$, $\neg a$	$\Diamond_{\leq k} a$, a	$\cdots$
n_k	0	1	2	0	0	1	0	$\cdots$
m_k	0	0	0	2	2	2	2	$\cdots$

Example for $k \in \mathcal{P}_\downarrow$:

	$\cdots$	$\Box_{\leq k} a$, a	a	$\Box_{\leq k} a$, a	a	a	$\neg a$	$\Box_{\leq k} a$, a	a	$\neg a$	a	$\cdots$
a_k	0	1	1	1	1	1	0	1	1	0	0	$\cdots$
n_k	0	0	1	0	1	2	0	0	1	0	0	$\cdots$
m_k	∞	∞	∞	∞	∞	∞	2	2	2	1	1	$\cdots$

Figure 2.1: The update function

For parameters k occuring in an upward-monotone operator, namely $\Diamond_{\leq k}$, we need two measuring variables m_k and n_k. The latter one is a counter used to determine the number of steps it takes from the first assumption that $\Diamond_{\leq k}\varphi$ holds to the first position where φ holds. m_k is then used to store the maximum of these values for each iteration of $\Diamond_{\leq k}$. Intuitively, we update the counters as follows:
As long as $\Diamond_{\leq k}\varphi$ is present in the visited atom, meaning that is assumed to be satisfied later, we increment n_k in every step. At the first position where φ is present, we compute the maximum of the previously measured instance (namely m_k) and the current instance n_k and store it in m_k.
For parameters k occuring in a downward-monotone operator, namely $\Box_{\leq k}$, we need three measuring variables m_k, n_k and a_k. The first two are used in a similar fashion as for upward-monotone k, with n_k counting the number of steps until φ does not hold anymore, instead. a_k is used to store whether the counter is currently *active* or not. We activate the counter for k at the first position where $\Box_{\leq k}\varphi$ is present and start incrementing n_k as long as φ is present, as well. Whenever a new instance of $\Box_{\leq k}\varphi$ is started, i.e. is present again in the current atom, we need to reset n_k to zero, since we need to make sure that the counted value is satisfying for *every* instance of the formula. The first time where φ (and by At.4 therefore $\Box_{\leq k}\varphi$) is not present anymore, we take the minimum of n_k and the previously measured value m_k and store it in m_k. Assuming initially that there is no position where φ is not present anymore, we set k to ∞ in the initial assignment.[1]
The workings of the update function are illustrated in Figure 2.1. It is important to understand the need of the activity flag a_k for the downward monotonous parameters. They are neccessary to know whether or not n_k needs to be incremented when a is present. Formally, we define the following:

$$P \quad := \quad \{m_k, n_k \mid k \in \mathcal{K}\} \cup \{a_k \mid k \in \mathcal{K}_\downarrow\} \tag{$\mathcal{A}$.5}$$

$$\theta(\nu) \quad := \quad \begin{cases} \infty & \nu = m_k \text{ for some } k \in \mathcal{K}_\downarrow \\ 0 & \text{otherwise} \end{cases} \tag{$\mathcal{A}$.6}$$

We then define the update function γ, such that for each $\tau, \tau' \in At_\Phi$ and $\eta : p \to \mathbb{N}$ holds: $\gamma(\tau, \tau')(\eta) = \eta'$ where

[1] Technically, ∞ is not a member $\mathbb{N}$, making the initial assignment ill-formed by definition. But since it can be chosen arbitrarily large, we assume that it is not reachable by any counter.

for each $(\Diamond_{\leq k}\varphi) \in cl(\Phi)$: $(\mathcal{A}.7)$

```
if $(\Diamond_{\leq k}\varphi) \in \tau'$ and $\varphi \in \tau'$ then $\eta'(m_k) = \max\{\eta(m_k), \eta(n_k)\}$ ; $\eta'(n_k) = 0$
else if $(\Diamond_{\leq k}\varphi) \in \tau'$ then $\eta'(m_k) = \eta(m_k)$ ; $\eta'(n_k) = \eta(n_k) + 1$
else $\eta'(m_k) = \eta(m_k)$ ; $\eta'(n_k) = \eta(n_k)$
```

and for each $(\Box_{\leq k}\varphi) \in cl(\Phi)$: $(\mathcal{A}.8)$

```
If $\eta(a_k) = 1$
    if $(\Box_{\leq k}\varphi) \in \tau'$ then $\eta'(m_k) = \eta(m_k)$ ; $\eta'(n_k) = 0$ ; $\eta'(a_k) = 1$
    else if $\varphi \in \tau'$ then $\eta'(m_k) = \eta(m_k)$ ; $\eta'(n_k) = \eta(n_k) + 1$ ; $\eta'(a_k) = 1$
    else $\eta'(m_k) = \min\{\eta(m_k), \eta(n_k)\}$ ; $\eta'(n_k) = 0$ ; $\eta'(a_k) = 0$
else
    if $(\Box_{\leq k}\varphi) \in \tau'$ then $\eta'(m_k) = \eta(m_k)$ ; $\eta'(n_k) = \eta(n_k) + 1$ ; $\eta'(a_k) = 1$
    else $\eta'(m_k) = \min\{\eta(m_k), \eta(n_k)\}$ ; $\eta'(n_k) = 0$ ; $\eta'(a_k) = 0$
```

Spawning Component

To measure quantified subformulae of the specification, we spawn sub-automata for every instantiation of the quantifier. More precisely, whenever a quantified subformula $\forall x{:}p.\varphi$ (or $\exists x{:}p.\,\varphi$) is assumed to be satisfied in the visited atom, we spawn sub-automata $\mathcal{A}_{\varphi,\beta[d/x]}$ for every neccessary d and require acceptance of all (or one) of these automata over the sub-trace starting at the spawning-position. The monitoring automaton for a quantifier-free FO-PLTL specification Φ will have level 0 and therefore no spawning capabilities. To determine the level of a monitoring automaton for an arbitrary FO-PLTL formula, we define the *quantifier-depth* (see [7]):

Definition 2.6 (Quantifier-depth)**.** *For a FO-PLTL formula, we define the* quantifier-depth *inductively as follows:*

$$\begin{aligned}
dep(\boldsymbol{true}) &= dep(p(t_1, ..., t_n)) \\
&= dep(\neg p(t_1, ..., t_n)) && := 0 \\
dep(\bigcirc\varphi) &= dep(\Box_{\leq l}\varphi) \\
&= dep(\Diamond_{\leq k}\varphi) && := dep(\varphi) \\
dep(\varphi_1 \vee \varphi_2) &= dep(\varphi_1 \wedge \varphi_2) \\
&= dep(\varphi_1 \,\boldsymbol{U}\, \varphi_2) \\
&= dep(\varphi_1 \,\boldsymbol{R}\, \varphi_2) && := \max\{dep(\varphi_1), dep(\varphi_2)\} \\
dep\big(\forall(x_1, ..., x_n){:}p.\,\varphi\big) &= dep\big(\exists(x_1, ..., x_n){:}p.\,\varphi\big) && := 1 + dep(\varphi)
\end{aligned}$$

We then use this definition in the construction of our monitoring automaton to determine the level:

$$l \quad := \quad dep(\Phi) \qquad (\mathcal{A}.9)$$

We spawn sub-automata to verify the satisfaction of quantified subformulae. Correlating to the semantic definition of the respective quantifier, for universal quantification we require every sub-automaton to be accepting and for existential quantification only one suffices. This is captured in the spawning function which uses conjunction for universal and disjunction for existential quantification, similarly to the way it is done in [7]. Consequently, for every event e and destination atom τ', we have:

$$\lambda(e,\tau') := \left(\bigwedge_{(\forall x:p.\,\varphi)\in\tau'} \left(\bigwedge_{p(d)\in e} \mathcal{A}_{\varphi,\beta[d/x]} \right) \right) \wedge \left(\bigwedge_{(\exists x:p.\,\varphi)\in\tau'} \left(\bigvee_{p(d)\in e} \mathcal{A}_{\varphi,\beta[d/x]} \right) \right) \qquad (\mathcal{A}.10)$$

Although this definition does not capture the unambiguity of the measure for existential quantification, we know by Theorem 1.4 that the acceptance condition is chosen correctly. In order to make sure we compute the correct measure, we have to define the *result* accordingly.

Measuring result

Before we define how the correct result is determined, we show the following property.

Theorem 2.1 (Uniqueness of the accepting run)**.** *For every FO-PLTL formula Φ, trace σ and variable-assignment β holds:*

$$\left|\Pi_{\mathcal{A}_{\Phi,\beta}}(\sigma)\right| \quad \leq 1$$

This means that there is at most one accepting run for every specification and variable-assignment. Whenever existent, we refer to it as $\pi_{\Phi,\sigma,\beta}$. The measure of Φ over σ is then obtained by evaluating the measuring-assigment at the end of an accepting run, taking results of the spawned sub-automata into account.

Definition 2.7 (Result of the measuring automaton)**.** *For an accepting run $\pi \in \Pi_{\mathcal{A}_{\Phi,\beta}}(\sigma)$ where $\pi = (\tau_0,\eta_0)\cdots(\tau_n,\eta_n)$, we define the* local result *for every $k \in \mathcal{K}$ as follows:*

$$res_{loc}(\pi)(k) \quad := \quad \eta_n(m_k)$$

We then define the result *of a spawning measuring automaton mutually recursively as follows:*

$$res(\mathcal{A}_{\Phi,\beta},\sigma) \quad := \quad res(\pi_{\Phi,\sigma,\beta})$$

where for every $\pi = (\tau_0,\eta_0)\cdots(\tau_n,\eta_n) \in \Pi_{\mathcal{A}_{\Phi,\beta}}(\sigma)$:

$$\begin{aligned}
res(\pi) \quad &:= \quad \begin{cases} res_{loc}(\pi) & \text{if } dep(\Phi) = 0 \\ res_{loc}(\pi) \bigcap \left[\bigcap_{i=1}^{n} res(\pi_i) \right] & \text{otherwise} \end{cases} \\
res(\pi_i) \quad &:= \quad \left(\bigcap_{(\forall x:p.\,\varphi)\in\tau_i} \left(\bigcap_{d\in\sigma_p[i-1]} res\big(\mathcal{A}_{\varphi,\beta[d/x]}, \sigma[i-1...]\big) \right) \right) \bigcap \\
&\qquad \left(\bigcap_{(\exists x:p.\,\varphi)\in\tau_i} \max_{d\in\sigma_p[i-1]} \left\{ res\big(\mathcal{A}_{\varphi,\beta[d/x]}, \sigma[i-1...]\big) \right) \right)
\end{aligned}$$

where $\cap$ is the valuation conjunction and max *is determined wrt. $\trianglerighteq$, considering only existing results.*

This definition captures in particular the unambiguous semantics for the existential quantification by choosing the best result under all accepting instantiations. Now that we have formalized the measuring automaton and procedure to obtain measures from accepting runs, it remains to show the correctness of our algorithm.

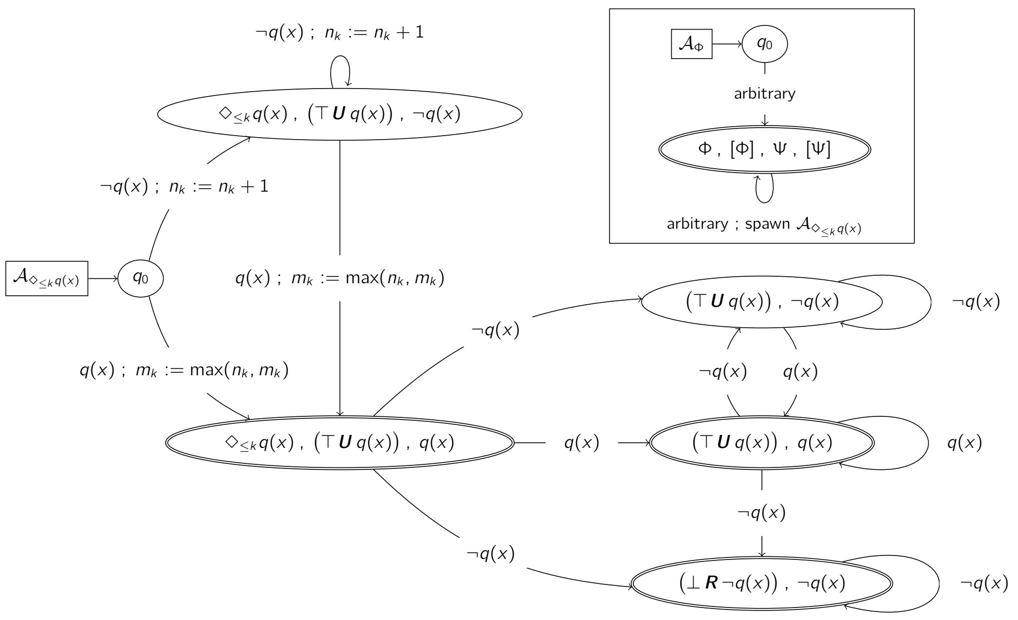
$\neg q(x)$; $n_k := n_k + 1$
$\Diamond_{\leq k} q(x)$, $(\top \, \boldsymbol{U} \, q(x))$, $\neg q(x)$
$\neg q(x)$; $n_k := n_k + 1$
$\mathcal{A}_{\Diamond_{\leq k} q(x)}$
q_0
$q(x)$; $m_k := \max(n_k, m_k)$
$q(x)$; $m_k := \max(n_k, m_k)$
$\Diamond_{\leq k} q(x)$, $(\top \, \boldsymbol{U} \, q(x))$, $q(x)$
$\neg q(x)$
$q(x)$
$\neg q(x)$
$\mathcal{A}_\Phi$
q_0
arbitrary
Φ , $[\Phi]$, Ψ , $[\Psi]$
arbitrary ; spawn $\mathcal{A}_{\Diamond_{\leq k} q(x)}$
$(\top \, \boldsymbol{U} \, q(x))$, $\neg q(x)$
$\neg q(x)$
$\neg q(x)$
$q(x)$
$(\top \, \boldsymbol{U} \, q(x))$, $q(x)$
$q(x)$
$\neg q(x)$
$(\bot \, \boldsymbol{R} \, \neg q(x))$, $\neg q(x)$
$\neg q(x)$

Figure 2.2: Examples of monitoring automata

Theorem 2.2 (Correctness of $\mathcal{A}_{\Phi,\beta}$)**.** *For every First-order PLTL formula Φ, trace σ and variable-assignment β holds that – whenever existing –*

$$res(\mathcal{A}_{\Phi,\beta}, \sigma) \quad = \quad \alpha^{\succ}_{\Phi,\sigma,\beta}.$$

Example 2.3.1. *An example of ms-automata for monitoring $\Phi = \Box(\forall x{:}p.\, \Diamond_{\leq k} q(x))$ can be found in Figure 2.2. The upper right automaton is the toplevel-automaton that spawns an automaton $\mathcal{A}_{\Diamond_{\leq k} q(x)}$ for every instantiation of the quantification $\Psi = \forall x{:}p.\, \Diamond_{\leq k} q(x)$. $\mathcal{A}_\Phi$ uses default-transitions, since $cl(\Phi)$ does not contain any atomic formulae besides $\top := \boldsymbol{true}$ and $\bot := \boldsymbol{false}$. Since $\Box\Psi$ is only an abbrevation of $\boldsymbol{false}\, \boldsymbol{R}\, \Psi$, by S.5 and the fact that $\boldsymbol{false}$ is absent in every atom, we need to assume that Ψ holds at every point in time. Therefore we only have one atom[2], namely the one which contains both Φ, Ψ and their respective abstractions, and one default transition that spawns the corresponding obligation in every step.*
One can easily observe that $\mathcal{A}_{\Diamond_{\leq k} q(x)}$ measures only one instance of the parametric operator. As soon as $q(x)$ is present for the first time, we don't need to measure $\Diamond_{\leq k}$ anymore. In fact, this is where the condition for the successor relation, that τ' needs to be the smallest *respective set (see Definition 2.5), has its roots.*

[2]Actually, we have more than this one atom as states in the automaton, but they are either not reachable or can be ommited since they cannot appear in an accepting run (see 3.1.2)

Summary

In this chapter we have developed an automaton for measuring a FO-PLTL specification under unambiguous semantics. We use sets of subformulae of the specification as states to verify the described behaviour and various counters and other variables to obtain a measuring result. The monitor is constructed in such a way that, whenever a specification is satisfied by an input-trace, there exists exactly one accepting run which can be used to determine the optimal paramever-valuation, i.e. the measure under the unambiguous semantics. This automaton construction is the foundation for the implementation of the monitoring algorithm which will be considered at in the following chapter.

Chapter 3

Implementation and results

This chapter is dedicated to the implementation of the monitoring automaton developed in the last chapter. We will present implementation details and optimizations, and discuss some results afterwards.

3.1 Implementation details

The implementation (fopltlmon[1]) is written in Scala[2], a functional and object-oriented programming language that combines functional and imperative programming paradigms and is therefore optimal for the implementation of the formal notions in the last chapter, using imperative methods to implement the first-order component. As Scala runs on a Java Virtual Machine, it can be used cross-platform.

fopltlmon supports both strings and integers as data types, coming with the standard comparison predicates. One can use $=$ and $\neq$ for every two values; for integers, the standard comparison operators $<, >, \leq, \geq$ are implemented as interpreted predicates, as well.

3.1.1 The basic approach

We maintain possible runs for every automaton that is considered. Each run carries the current state (intially q_0), the current measuring-assignment (initially θ) and the obligations that have been spawned up to this point in this run (initially none). When processing an event e, we eliminate runs for which the state has no successor and create new runs when a state has more than one successor. Depending on the destination state, we update the measuring-assignment and add obligations to the run. After processing the last event, we consider the (unique) accepting run and compute the result using the meausuring-assignment and the sub-results of the spawning obligations. Whenever there is no run anymore, a violation of the specification is reported.

[1]fopltlmon can be found here: `https://github.com/KaiHornung/fopltlmon`
[2]see `http://www.scala-lang.org`

3.1.2 Optimizations

Automaton Prototypes

The problem with monitoring a first-order logic is always how to handle a great number of quantifier instantiations. In our monitoring approach, we need to consider a possibly large number of spawned sub-automata at every position of the trace. It is clear that we cannot create an entirely new instance of the sub-automaton with every spawning. Therefore, we use *automaton prototypes*. These prototypes are constructed independently of the variable-assignment an can be used for every β. The transitions of the prototype are therefore not labelled with events, but with sub-sets of the closure that contain every atomic formula which is an uninterpreted predicate or its respective negation.
Therefore, when it comes to spawning a new automaton, we only need to store the valuation and references to the current state in the corresponding prototype for every run. To process an event, we then determine which of the $\mathbb{I}_\beta(\varphi)$ (where φ is an uninterpreted predicate formula) is present in the event and take the corresponding prototype transitions that are labelled with these φ. Since the computation of all atoms can be costly, we precompute all possibly needed prototypes before processing a trace.
The usage of automaton prototypes eliminates the need to both reinstantiate and recalculate every state and transition for every instantiation of quantified subformulae. In fact, the example in Figure 2.2 shows the prototype $\mathcal{A}_{\Diamond_{\leq k} q(x)}$ that can be used for every $\mathcal{A}_{\Diamond_{\leq k} q(x),\beta}$.

Early detection of acceptance and violations

As mentioned in the introduction, an advantage of online monitors is that violations can be detected early on. Therefore, we can eliminate non-satisfiable sub-automata – and correspondingly, runs that contain these sub-automata as obligations – during the processing of the trace. By minimalizing the measuring automaton, the algorithm detects when an automaton is accepting and will stay accepting for every possible upcoming event and stops updating them in every step. To save memory, the respective result is stored in a database-like environment and the automaton itself is deallocated.

Abstractions and spawnings

As mentioned above, we maintain the invariant that in an accepting run every visited atom contains exactly those subformulae which are satisfied at that point in the trace. In particular, this means that an atom which contains φ and $\neg[\varphi]$ – altough possibly well-defined as an atom – cannot be part of such an accepting run, since φ and $[\varphi]$ are equally satisfiable by Theorem 1.3. To decrease the number of states, we therefore add the following restriction which needs to hold for every atom τ and formula φ.

$$\text{If } \varphi \in \tau, \text{ then } [\varphi] \in \tau.$$

It is easy to see that the need of the FO-LTL abstraction can also drastically increase the number of spawnings per step, because it is possible that e.g. for $\varphi = \forall x{:}p.\, \varphi'$ both φ and $[\varphi]$ are contained in the current atom. To avoid the use of unneccessary spawnings, we add the following: Let φ be of the form $\forall x{:}p.\, \varphi'$ or $\exists x{:}p.\, \varphi'$ and τ be an atom.

$$\text{If } \varphi \in \tau,\ [\varphi] \in \tau \text{ and } \varphi \neq [\varphi], \text{ then don't spawn an obligation for } [\varphi].$$

	3 processes		5 processes		10 processes	
trace length	time	mem	time	mem	time	mem
10k events	3081	85	3169	85	3212	85
100k events	11211	85	11767	85	13210	85
1M events	86674	85	95065	85	115823	85

Figure 3.1: Run-time and memory result for the mutex experiment

Since $[\varphi]$ is non-parametric and therefore has no measuring capabilities, it is sufficient to consider φ in the spawning obligations to compute the correct measure without violating the acceptance condition by Theorem 1.3.

3.2 Experiments

To test the efficiency of this implementation, two benchmarks were generated. The system that ran the tests was an 3.4GHz Intel Core i5 Quad Core with 8GB of available memory. The implementation (version 0.1) has been run on a Java 1.6 runtime environment, reading traces from and writing results to an internal solid state drive. Each test generated a number of results that will partially be presented in this section:

Mutex experiment

The first measuring experiment analyzed a number of processes trying to enter a critical section. The specification was chosen in such a way such that verification of the mutual exclusion and measuring of the time between requesting access to the critical section and being granted access are covered. Each process is first active in the non-critical region, then waiting for access to the critical section, then active in the critial section and then leaves the critical section to start over. We considered the corresponding update actions $wait(i)$, $enter(i)$ and $exit(i)$ for a process with the id i. The FO-PLTL input specification was therefore defined as follows:

$$\Box\left(\underbrace{\forall x{:}enter.\bigcirc\big(exit(x)\ \boldsymbol{R}\ (\forall y{:}enter.\,\boldsymbol{false})\big)}_{\text{Veryfing mutual exclusion}} \wedge \underbrace{\forall z{:}wait.\,\Diamond_{\leq k} enter(z)}_{\text{measuring waiting time}}\right)$$

The tested input traces have been generated by randomly chosing a process and updating its status to the next one (if possible). We have tested trace lengths of 10.000 to 1.000.000 events for up to 10 processes. The run-times in milliseconds and needed memory in MB is shown in Figure 3.1.

It is important to know that the figure shows only the run-time of the actual trace processing, the time needed to construct the prototype automata is not included. For the specification above, 8 prototypes have been constructed, altogether in about 380ms. The most significant observation with the above numbers, though, is that there is no measurable difference in the memory usage regarding different domain sizes. This displays an enormous advantage to prospotional PLTL, where each added process would result in growth of the formula and therefore the measuring automaton.

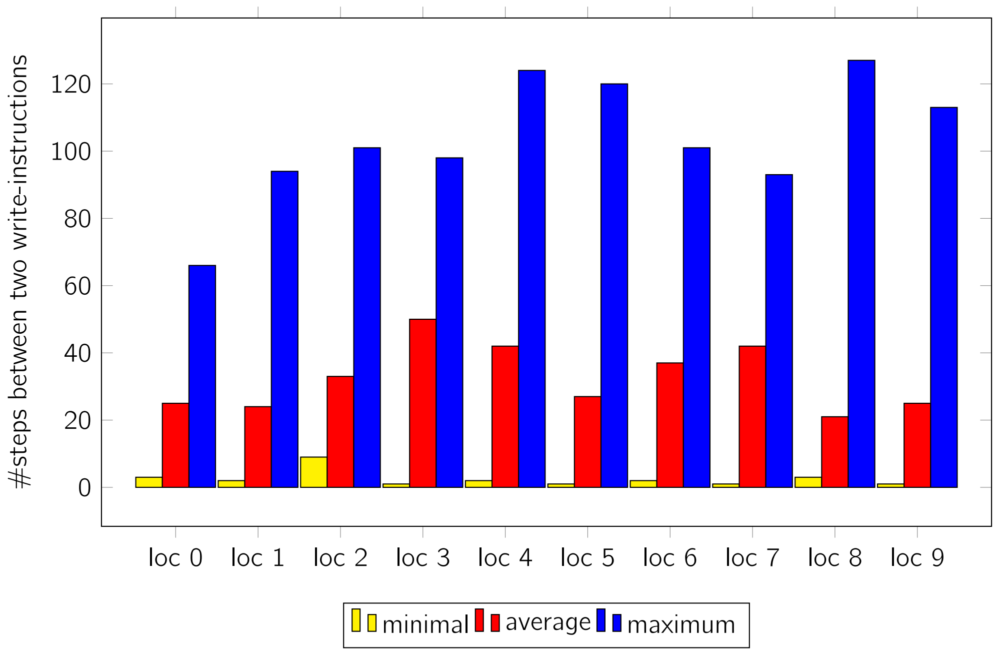

Figure 3.2: Example results of the memory controller experiment

Memory controller experiment

In this experiment, we have used the FO-PLTL monitoring algorithm to measure the retention period of a memory cell. The same simulator as used in the memory controller experiment in [12] has been used to generate the input traces. The input specifiaction is of greater size and can be found inside the downloadable implementation. Figure 3.2 shows a graphical presentation of the measured data, depending on the location (loc) in the memory. It shows minimal, maximal and an average of all measured times between two write instructions to the same memory location as the number of processor steps. As in the mutex experiment, we have measured traces of lengths between 10.000 and 1.000.000 events, considering up to 10 different memory locations.

Summary

We have seen a number of example results of the implementation of the monitoring algorithm presented in the last chapter. This implementation not only determines the (overall) unambiguous measure, but also collects and stores detailed measuring results which can be used to analyze a system's temporal behaviour.
Due to optimizations in the implementation, the algorithm can be used scalably in the size of the considered active domain, in particular when it comes to memory consumption. This gives a great advantage to propositional approaches that need to cope with growing specification size when considering a larger amount of considered objects. A larger specification means in particular that the monitoring automata explode in size, wherease the FO-PLTL measuring automaton remains constant with growing domain size.

Conclusion and related work

We have presented a framework for runtime analysis of computer systems, covering both verification of correct behaviour and the collection of detailed temporal information about a given execution. By providing FO-PLTL with quantified variables and predicates, we created a powerful specification language with the ability to reason about data-dependent properties. Future work could include extending this work to an infinite-trace semantics and a more general way to obtain domain-independent specifications.
Formal methods for runtime verification of temporal properties have already been extensively studied. A variety of different approaches to verify LTL properties has been developed, either automata based like in this thesis ([13],[8],[24]) or using different approaches ([14],[23]). Many ways to express data-aware properties using first-order temporal logic and corresponding monitoring algorithms have been developed. While [20] only covers finite domains, [4] and [5] even cover verification of traces with infinite events over an arbitrary domain, using database-approaches based on [9]. Different approaches for first-order monitoring can be found in [11] and [21]. The first-order extension in this thesis is mainly based on the results presented in [7], whose development also aims to develop a runtime verification framework for software systems like the Android operating systems and respective applications ([6],[19]).
Altough first-order monitoring has already been studied, this thesis presents the first approach for not only rutime verification, but also "measuring" using first-order properties. The basis for PLTL can be found in [2] and corresponding monitoring approaches are presented in [12].

List of Definitions

List of Theorems

List of Figures

Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu, editors. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1995.

[2] Rajeev Alur, Kousha Etessami, Salvatore La Torre, and Doron Peled. Parametric temporal logic for “model measuring”. In Jiří Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming*, volume 1644 of *Lecture Notes in Computer Science*, pages 159–168. Springer Berlin Heidelberg, 1999.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[4] David Basin, Felix Klaedtke, Samuel Müller, and Birgit Pfitzmann. Runtime Monitoring of Metric First-order Temporal Properties. In Ramesh Hariharan, Madhavan Mukund, and V Vinay, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 2 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49–60, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[5] David A. Basin, Felix Klaedtke, Srdjan Marinovic, and Eugen Zalinescu. Monitoring of temporal first-order properties with aggregations. In *Runtime Verification - 4th International Conference, RV 2013, Rennes, France, September 24-27, 2013. Proceedings*, pages 40–58, 2013.

[6] Andreas Bauer and Jan-Christoph Küster. Runtime verification meets Android security. In Alwyn Goodloe and Suzette Person, editors, *Proceedings of the 4th NASA Formal Methods Symposium (NFM)*, Lecture Notes in Computer Science, Berlin, Heidelberg, April 2012. Springer-Verlag.

[7] Andreas Bauer, Jan-Christoph Küster, and Gil Vegliach. From propositional to first-order monitoring. In Axel Legay and Saddek Bensalem, editors, *Runtime Verification*, volume 8174 of *Lecture Notes in Computer Science*, pages 59–75. Springer Berlin Heidelberg, 2013.

[8] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for ltl and tltl. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, September 2011.

[9] Jan Chomicki, David Toman, and Michael H. Böhlen. Querying atsql databases with temporal logic. *ACM Trans. Database Syst.*, 26(2):145–178, 2001.

[10] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 854–860. AAAI Press, 2013.

[11] Normann Decker, Martin Leucker, and Daniel Thoma. Monitoring modulo theories. In *TACAS'14*, pages 341–356, 2014.

[12] Peter Faymonville, Bernd Finkbeiner, and Doron Peled. Monitoring parametric temporal logic. In *VMCAI*, pages 357–375, 2014.

[13] Bernd Finkbeiner and Henny Sipma. Checking finite traces using alternating automata. *Formal Methods in System Design*, 24(2):101–127, March 2004.

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

[15] Sylvain Hallé and Roger Villemaire. Runtime monitoring of message-based workflows with data. In *EDOC*, pages 63–72. IEEE Computer Society, 2008.

[16] Richard Hull and Jianwen Su. Domain independence and the relational calculus. *Acta Informatica*, 31:513–524, 1993.

[17] Volha Kerhet and Enrico Franconi. On checking domain independence. In Francesca A. Lisi, editor, *CILC*, volume 857 of *CEUR Workshop Proceedings*, pages 246–250. CEUR-WS.org, 2012.

[18] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Syst.*, 2(4):255–299, October 1990.

[19] Jan-Christoph Küster and Andreas Bauer. Platform-centric android monitoring—modular and efficient. Computing Research Repository (CoRR) arXiv:1406.2041, Association for Computing Machinery (ACM), June 2014.

[20] Riccardo De Masellis and Jianwen Su. Runtime enforcement of first-order ltl properties on data-aware business processes. In Samik Basu, Cesare Pautasso, Liang Zhang, and Xiang Fu, editors, *ICSOC*, volume 8274 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2013.

[21] Ramy Medhat, Yogi Joshi, Borzoo Bonakdarpour, and Sebastian Fischmeister. Parallelized runtime verification of first-order ltl specifications. 2014.

[22] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[23] Grigore Roşu, Feng Chen, and Thomas Ball. Runtime verification. chapter Synthesizing Monitors for Safety Properties: This Time with Calls and Returns, pages 51–68. Springer-Verlag, Berlin, Heidelberg, 2008.

[24] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata, volume 1043 of Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, 1996.

Appendix A

Detailed proofs

Proof of Theorem 1.1

PNF is reached by first resolving every occurence of derived operators using their definition from Definition 1.7. Afterwards, we push negations downward using the following equivalences for non-atomic formulae:

$$\neg(\varphi_1 \wedge \varphi_2) \equiv \neg\varphi_1 \vee \neg\varphi_2 \tag{A.1}$$
$$\neg(\varphi_1 \vee \varphi_2) \equiv \neg\varphi_1 \wedge \neg\varphi_2 \tag{A.2}$$
$$\neg\bigcirc\varphi \equiv \bigcirc\neg\varphi \tag{A.3}$$
$$\neg(\varphi_1 \boldsymbol{U}\, \varphi_2) \equiv \neg\varphi_1 \boldsymbol{R}\, \neg\varphi_2 \tag{A.4}$$
$$\neg(\varphi_1 \boldsymbol{R}\, \varphi_2) \equiv \neg\varphi_1 \boldsymbol{U}\, \neg\varphi_2 \tag{A.5}$$
$$\neg\Diamond_{\leq k}\varphi \equiv \Box_{\leq k}\neg\varphi \tag{A.6}$$
$$\neg\Box_{\leq k}\varphi \equiv \Diamond_{\leq k}\neg\varphi \tag{A.7}$$
$$\neg\forall(x_1, ..., x_n) : p.\varphi \equiv \exists(x_1, ..., x_n) : p.\neg\varphi \tag{A.8}$$
$$\neg\exists(x_1, ..., x_n) : p.\varphi \equiv \forall(x_1, ..., x_n) : p.\neg\varphi \tag{A.9}$$

A.1 and A.2 correspond to the De-Morgan laws, the rewrite rules for A.3, A.4 and A.5 can be found in [3], A.6 and A.7 in [2]. Equivalences A.8 and A.9 also follow directly from the definition of $\models$ and of the De-Morgan laws. Afterwards, double negations can easily be eliminated using the equivlalence

$$\neg\neg\varphi \equiv \varphi \tag{A.10}$$

that follows directly from the semantic interpretation of $\neg$. ■

Proof of Theorem 1.2

An example of such a formula and trace under $\beta_\emptyset$ can be found in Example 1.3.1 and is illustrated in Figure 1.3. ■

Proof of Theorem 1.3

We show the claim by induction over the structure of φ for the stronger claim where the position is not 0, but an arbitrary i. The base cases and the cases for the non-parametric connectives are either trivial or follow directly from the inductive hypothesis. Therefore we only consider the following two cases:

- case $\varphi = \Diamond_{\leq k}\varphi'$:

  Assume $\forall\alpha.(\sigma, i.\alpha, \beta) \models [\varphi]$. Therefore by definition of $[\cdot]$, for all α exists $j \geq i$ s.t. $(\sigma, j, \alpha, \beta) \models [\varphi']$. Since satisfaction of $[\varphi']$ does not depend on α, we know that there is a j such that for all α, $(\sigma, j, \alpha, \beta) \models [\varphi']$. By the inductive hypothesis exists a j such that there is an α' s.t. $(\sigma, j, \alpha', \beta) \models \varphi'$. By definition of $\models$ and the fact that $k \notin \mathcal{K}_{\varphi'}$, we obtain $(\sigma, i, \alpha'[j/k], \beta) \models \varphi$.

  Assume there is an α s.t. $(\sigma, i, \alpha, \beta) \models \varphi$. Therefore there is a $\leq j \leq \alpha(k)$ s.t. $(\sigma, j, \alpha, \beta) \models \varphi'$. By the inductive hypothesis, we get that there is a j such that for all α', $(\sigma, j, \alpha', \beta) \models [\varphi']$. Therefore for all α' holds that there is a j such that $(\sigma, j, \alpha', \beta) \models [\varphi']$, so by definition of $[\cdot]$ and $\models$, for all α', $(\sigma, i, \alpha', \beta) \models [\varphi]$.

- case $\varphi = \Box_{\leq k}\varphi'$:

  Assume $\forall\alpha.(\sigma, i.\alpha, \beta) \models [\varphi]$. Therefore by definition of $[\cdot]$, $\forall\alpha.(\sigma, i.\alpha, \beta) \models [\varphi']$. By the inductive hypothesis we know that there is an α' s.t. $(\sigma, i.\alpha', \beta) \models \varphi'$. By definition of $\models$, we get $(\sigma, i.\alpha'[0/k], \beta) \models \varphi$.

  Assume there is an α s.t. $(\sigma, i, \alpha, \beta) \models \varphi$. Therefore, in particular $(\sigma, i, \alpha, \beta) \models \varphi'$. By the inductive hypothesis, we get that for all α' that $(\sigma, i, \alpha', \beta) \models [\varphi']$. Therefore, by definition of $[\cdot]$, for all α', $(\sigma, i, \alpha', \beta) \models [\varphi]$.

■

Proof of Theorem 1.4

We show this theorem by induction over the structure of φ where the position is not 0, but an arbitrary i. By definition of the unambiguous semantics we can omit the cases $\varphi = p(\cdots)$, $\varphi = \neg p(\cdots)$, $\varphi = \varphi_1 \wedge \varphi_2$, $\varphi = \bigcirc\varphi'$ and $\forall x{:}p.\,\varphi'$, since they have the same definition for satisfaction. For the other cases, we have the following inductive hypothesis: For every subformula φ' of φ, position i, σ and β holds:

$$\exists\alpha.\,(\sigma, i, \alpha, \beta) \models \varphi' \qquad \text{iff} \qquad \exists\alpha'.\,(\sigma, i, \alpha', \beta) \models^{\succ} \varphi' \tag{IH}$$

Since the unambiguous semantics is stronger than the respective counterpart for every considered operator, the "if"-direction follows easily. Therefore, we focus on the "only if"-part. We use $\alpha \models \varphi'$ to abbreviate $(\sigma, i, \alpha, \beta) \models \varphi'$ and $j, \alpha \models \varphi'$ to abbreviate $(\sigma, j, \alpha, \beta) \models \varphi'$.

- case $\varphi = \varphi_1 \vee \varphi_2$:

  Let α s.t. $\alpha \models \varphi$. Therefore, by Def. $\models$, we have $\alpha \models \varphi_1$ or $\alpha \models \varphi_2$. We consider two cases:

  If $\alpha \models \varphi_1$, by IH, we know that there is an α' such that $\alpha' \models^{\succ} \varphi_1$. Therefore, by Def. $\models^{\succ}$, we have that $\exists\alpha'.\,\alpha' \models \varphi$.

Otherwise, assume $\alpha \models \varphi_2$, we consider two sub-cases. If $\alpha \models \neg[\varphi_1]$, we know by Theorem 1.3 that $\forall\alpha'.\alpha' \not\models \varphi_1$. By IH follows $\forall\alpha'.\alpha' \not\models^{\succ} \varphi_1$. Since by IH, there is an α'_0 with $\alpha'_0 \models^{\succ} \varphi_2$, $\alpha'_0 \models^{\succ} \varphi$ follows directly. If $\alpha \models [\varphi_1]$, we know by Theorem 1.3 that there is an α' s.t. $\alpha' \models \varphi_1$. With the same argumenation as above, the existence of a satisfying valuation under unambiguous semantics follows.

- case $\varphi = \varphi_1 \boldsymbol{U} \varphi_2$:

  Let α s.t. $\alpha \models \varphi$. Therefore, by Def. $\models$, there is a j s.t. $j, \alpha \models \varphi_2$ and $\forall j' < j. j', \alpha \models \varphi_1$. If $\alpha \models \neg[\varphi_2]$ for all such j', $\alpha \models^{\succ} \varphi$ follows by Def. $\models^{\succ}$. Otherwise, set j_0 to the minimum j' s.t. $j', \alpha \models [\varphi_2]$.

  Therefore we know by the fact that $[\varphi_2]$ is non-parametric and Theorem 1.3 that $\exists\alpha_0. j_0, \alpha_0 \models \varphi_2$ and for each $j' < j_0$ and $\alpha_{j'}. j', \alpha_{j'} \models \neg[\varphi_2]$ $(*)$.

  By existence of α, α_0 and IH, we know that there is an α'_0 s.t. $j_0, \alpha'_0 \models^{\succ} \varphi_2$ and for every $j' < j_0$ exists $\alpha'_{j'}$ s.t. $j', \alpha'_{j'} \models^{\succ} \varphi_1$.

  Let α' be the conjunction of α'_0 and these $\alpha'_{j'}$. Then we have $j_0, \alpha' \models^{\succ} \varphi_2$ and for each $j' < j_0$, $\alpha' \models^{\succ} \varphi_1$. By $(*)$, we get $j_0, \alpha' \models^{\succ} \varphi_2$ and for each $j' < j_0$, $\alpha' \models^{\succ} \varphi_1 \wedge \neg[\varphi_2]$, which is $\alpha' \models^{\succ} \varphi$ by Definition of $\models^{\succ}$.

- case $\varphi = \varphi_1 \boldsymbol{R} \varphi_2$ follows almost directy the same argumentation as the Until-case.

- case $\varphi = \Diamond_{\leq k}\varphi'$:

  Let α s.t. $\alpha \models \varphi$. Therefore, by Def. $\models$, there is a $j \leq \alpha(k)$ s.t. $j, \alpha \models \varphi'$. Let j_0 be the minimal $j \geq i$ s.t. $j, \alpha \models [\varphi']$.

  Then we know by Theorem 1.3 that there is a α_0 s.t. $j_0, \alpha_0 \models \varphi'$ and for each $j' < j_0$ and α'_0, $j', \alpha'_0 \models \neg[\varphi']$. By IH, we get that there is an α'_0 s.t. $j_0, \alpha_0 \models^{\succ} \varphi'$ and for each $j' < j_0$, $\alpha'_0 \models \neg[\varphi']$. By setting $\alpha' := \alpha'_0[{}^{j_0}/_{k}]$, we know since $k \notin \mathcal{K}_{\varphi'}$ that $\alpha' \models^{\succ} \varphi$.

- case $\varphi = \exists x{:}p.\,\varphi'$:

  Let α s.t. $\alpha \models \varphi$. Therefore, by Def. $\models$, there is a $d \in \sigma_p[i]$ s.t. $\alpha, \beta[{}^{d}/_{x}] \models \varphi'$. By IH, we have the existence of an α_0, s.t. $\alpha_0, \beta[{}^{d}/_{x}] \models^{\succ} \varphi'$. Therefore the set $A := \{\alpha_0 \mid \exists d \in \sigma_p[i].\, \alpha_0, \beta[{}^{d}/_{x}] \models \varphi'\}$ is non-empty. By setting α' to the maximum of A with respect to $\trianglerighteq$, we obtain $\alpha' \models^{\succ} \varphi$.

■

Proof of Theorem 1.5

For the proof of this theorem, let $\alpha \not\sqsupseteq\!\!\not\sqsubseteq \alpha'$ denote the fact that α and α' are incomparable with respect to $\sqsupseteq$, i.e. neither $\alpha \sqsupseteq \alpha'$ nor $\alpha' \sqsupseteq \alpha$ holds.
In this proof, we need some properties of this incomparability which are stated in the following lemma. As above, we use $\alpha \models \varphi'$ to abbreviate $(\sigma, i, \alpha, \beta) \models \varphi'$ and $j, \alpha \models \varphi'$ to abbreviate $(\sigma, j, \alpha, \beta) \models \varphi'$.:

Lemma A.1. *For every α, α', α'' and α_i with $i \in I$ holds:*

1. *If $\alpha \not\sqsupseteq\!\!\not\sqsubseteq \alpha'$ and $\alpha'' \sqsupseteq \alpha'$, then $\alpha \not\sqsupseteq\!\!\not\sqsubseteq \alpha''$.*

2. *For all $i \in I$, $\alpha_i \sqsupseteq \bigcap_{i \in I} \alpha_i$.*

3. a. *If $\alpha \sqsupseteq \bigcap_{i \in I} \alpha_i$, then there is a $j \in I$ s.t. $\alpha \sqsupseteq \alpha_j \vee \alpha \not\sqsubseteq\!\!\not\sqsupseteq \alpha_j$.*

   b. *If $\alpha \sqsupset \bigcap_{i \in I} \alpha_i$, then there is a $j \in I$ s.t. $\alpha \sqsupset \alpha_j \vee \alpha \not\sqsubseteq\!\!\not\sqsupseteq \alpha_j$.*

   c. *If $\alpha \not\sqsubseteq\!\!\not\sqsupseteq \bigcap_{i \in I} \alpha_i$, then there is a $j \in I$ s.t. $\alpha \not\sqsubseteq\!\!\not\sqsupseteq \alpha_i$.*

4. *For every $\alpha, \alpha' \in M_{\varphi,\sigma,\beta}$: If $\alpha \neq \alpha'$, then $\alpha \not\sqsubseteq\!\!\not\sqsupseteq \alpha'$.*

5. *If $\alpha \in M_{\varphi,\sigma,\beta}$, $\alpha \not\sqsubseteq\!\!\not\sqsupseteq \alpha'$ and $\alpha' \models \varphi$, then there is an $\alpha'' \in M_{\varphi,\sigma,\beta}$ s.t. $\alpha \neq \alpha''$.*

Proof. All claims follow almost directly from their respective definition. □

We now show Theorem 1.5 by showing that if there is an α_0 s.t. $(\sigma, i, \alpha_0, \beta) \models \varphi$, then there is a unique measure for φ over σ under β. The base cases follow directy from the fact that they are non-parametric. We have the following inductive hypothesis: for every φ' subformula of φ, position i' and assignment β' holds:

$$\text{If } \exists \alpha_0'. (\sigma, i', \alpha_0', \beta') \models \varphi', \text{ then there is a unique measure } \alpha^{\succ}_{\varphi',\sigma[i...],\beta'}. \tag{IH}$$

Assume that $\exists \alpha_0. \alpha_0 \models \varphi$. $(*)$

- case $\varphi = \varphi_1 \wedge \varphi_2$:
  By $(*)$ and Def. $\models$, we know that $\alpha_0 \models \varphi_1$ and $\alpha_0 \models \varphi_2$. By IH, we have the existence of $\alpha_1 = \alpha^{\succ}_{\varphi_1,\sigma[i...],\beta}$ and $\alpha_2 = \alpha^{\succ}_{\varphi_2,\sigma[i...],\beta}$. Let α_1 and α_2 be fix. Set $\alpha := \alpha_1 \cap \alpha_2$.

  Claim: α is the unique measure for φ over $\sigma[i...]$ under β.

  $\alpha \models \varphi$ follows directly from the definition of $\cap$, remains to show that there is no satisfying $\alpha' \sqsupseteq \alpha$. Assume there is and let it be fix. By Def. $\models$, we have $\alpha' \models \varphi_1$ and $\alpha' \models \varphi_2$. By Lemma A.1.3a and the definition of α, we know that there is a $i \in \{1, 2\}$ s.t. $\alpha' \sqsupset \alpha_i \vee \alpha \not\sqsubseteq\!\!\not\sqsupseteq \alpha_i$. Let such an i be fix. If $\alpha' \sqsupset \alpha_i$, then α_i is not a measure, otherwise if $\alpha' \not\sqsubseteq\!\!\not\sqsupseteq \alpha_i$, then by Lemma A.1.5, α_i is not unique. Both cases contradict the assumption. Therefore we know that α is a measure.

  To show that it is unique, assume that there is an $\alpha' \neq \alpha$ that is a measure. By Lemma A.1.4, therefore $\alpha' \not\sqsubseteq\!\!\not\sqsupseteq \alpha$. By Lemma A.1.3c, we have $\alpha' \not\sqsubseteq\!\!\not\sqsupseteq \alpha_1 \vee \alpha' \not\sqsubseteq\!\!\not\sqsupseteq \alpha_2$. In both cases, since $\alpha' \models \alpha_i$, by Lemma A.1.6, we get that α_i is not unique which contradicts the assumptiom.

- case $\varphi = \varphi_1 \vee \varphi_2$:
  By $(*)$ and Def. $\models$, we have $\alpha_0 \models \varphi_1$ or $\alpha_0 \models \varphi_2 \wedge \neg[\varphi_1]$.

  - If $\alpha_0 \models \varphi_1$, then by IH there is $\alpha = \alpha^{\succ}_{\varphi_1,\sigma[i...],\beta}$. Let such α be fix.
    Claim: α is the unique measure for φ over $\sigma[i...]$ under β.
    $\alpha \models \varphi$ follows directly from Def. $\models$. Similar, to above, both for the measure-property and the uniqueness, assuming the opposite directly leads to contradictions to the fact that α is the unique measure for φ_1.

– If $\alpha_0 \models \varphi_2 \wedge \neg[\varphi_1]$, then by IH and the properties of the FO-LTL abstraction, there is a $\alpha = \alpha^{\succ}_{\varphi_2,\sigma[i\ldots],\beta}$ and $\forall \alpha'.\, \alpha' \not\models \varphi_1$. $(**)$
  Claim: α is the unique measure for φ over $\sigma[i\ldots]$ under β.
  $\alpha \models \varphi$ follows from $(**)$ and the definition of $\models$. To show the measure property, assume there is an $\alpha' \sqsupset$ and $\alpha' \models \varphi$. Therefore by $(**)$ $\alpha' \models \varphi_2 \wedge \neg[\varphi_1]$ and therefore $\alpha' \models \varphi_2$, which contradicts the fact that α is a measure for φ_2. To show uniqueness, assume there is a different measure α' for φ. Again, by $(**)$, we know that it is also a measure for φ_2. This contradicts the fact that α is the unique measure for φ_2.

- case $\varphi = \bigcirc\varphi'$:
  Since for all α', $\alpha' \models \varphi$ iff. $i+1, \alpha' \models \varphi'$, we know that $M_{\varphi,\sigma[i\ldots],\beta} = M_{\varphi',\sigma[i+1\ldots],\beta}$, the claim follows directly from the inductive hypothesis.

- case $\varphi = \varphi_1 \,\boldsymbol{U}\, \varphi_2$:
  By $(*)$, we know that there is a j s.t. $j, \alpha_0 \models \varphi_2$ and for all $j' < j$, $j' \models \varphi_1 \wedge \neg[\varphi_2]$. Let such j be fix. By the property of $[\cdot]$, we know that this position j is unique. $(**)$ By IH we know there are $\alpha_i, \ldots, \alpha_j$ s.t. $\alpha_j = \alpha^{\succ}_{\varphi_2,\sigma[j\ldots],\beta}$ and for each j', $i \leq j' < j$, $\alpha_{j'} = \alpha^{\succ}_{\varphi_1,\sigma[j'\ldots],\beta}$.
  Set $\alpha := \bigcap_{i'=i\ldots j} \alpha_{i'}$. Claim: α is the unique measure for φ.
  The claim is proven similarly to the cases above by assuming the opposite and applying Lemma A.1 and $(**)$.

- case $\varphi = \varphi_1 \,\boldsymbol{R}\, \varphi_2$ is proven almost identically to the Until-case, since whenever existent, the release-position is unique.

- case $\varphi = \Diamond_{\leq k}\varphi'$:
  By $(*)$, we know that there is a $j \leq \alpha(k)$ s.t. $j, \alpha_0 \models \varphi'$ and for all $j' < j$, $j' \models \neg[\varphi']$. Let such j be fix. By the property of $[\cdot]$, we know that this position j is unique. By IH we know there is $\alpha_j = \alpha^{\succ}_{\varphi',\sigma[j\ldots],\beta}$. Let $\alpha := \alpha[{}^{j}\!/\!{}_{k}]$.
  The claim that α is the unique measure for φ follows directly from the fact that j is the minimal satisfiable position for φ' by properties of $[\cdot]$ and the same argumentation as for the other cases.

- case $\varphi = \Box_{\leq k}\varphi'$:
  Set $j := \max\{j \mid \forall j' \leq j.\, i+j', \alpha_0 \models [\varphi']\}$. Therefore, for all $i > j$, $i, \alpha_0 \models \neg[\varphi']$. $(**)$ By IH we have $\alpha_i, \ldots, \alpha_{i+j}$ s.t. $\alpha_{j'} = \alpha^{\succ}_{\varphi',\sigma[j'\ldots],\beta}$ for each $j' = i, \ldots, i+j$. Let $\alpha = \bigcap_{j'=i,\ldots,i+j} \alpha_{j'}$.
  The claim that α is the unique measure for φ follows directly from $(**)$ and the same argumentation as for the other cases.

- case $\varphi = \forall x{:}p.\, \varphi'$:
  By $(*)$ and the IH, we have $\alpha_d := \alpha^{\succ}_{\varphi',\sigma[i\ldots],\beta[{}^{d}\!/\!{}_{x}]}$ for every $d \in \sigma_p[i]$. It follows directly from Lemma A.1 that $\alpha = \bigcap_{d\in\sigma_p[i]} \alpha_d$ is the unique measure for φ.

- case $\varphi = \exists x{:}p.\, \varphi'$ follows directly from the definition of $\models$. ■

Proof of Theorem 2.1

We show this theorem using three lemmas. The second and third then directly imply the claim. For every atom τ, let $[\tau]$ denote the subset of τ that contains only the non-parametric subformulae.

Lemma A.2. *For every $\tau, \tau_1, \tau_2 \in Q$ and $e \in E$ holds:*

$$\text{If } \tau \xrightarrow{e} \tau_1 \wedge \tau \xrightarrow{e} \tau_2 \wedge [\tau_1] = [\tau_2], \text{ then } \tau_1 = \tau_2.$$

Proof. Assume all conditions hold, but $\tau_1 \neq \tau_2$. Wlog., let $\varphi \in \tau_1 \setminus \tau_2$. Since $[\tau_1] = [\tau_2]$ by assumption, we know that φ is parametric. Because of the minimality condition of $\xrightarrow{e}$, we know that there is a condition from Definition 2.5 that requires φ to be present. Since all these conditions depend on τ alone, the same holds for τ_2 and therefore $\varphi \in \tau_2$ which contradicts the assumption. □

Lemma A.3. *For every two accepting runs $\pi, \pi' \in \Pi_{\mathcal{A}_{\Phi,\beta}(\sigma)}$ where $\pi = (\tau_0, \eta_0)\cdots(\tau_n, \eta_n)$ and $\pi' = (\tau'_0, \eta'_0)\cdots(\tau'_n, \eta'_n)$ holds:*

$$\text{If } \tau_{n-1} = \tau'_{n-1}, \text{ then } \tau_n = \tau'_n.$$

Proof. Let such π and π' be fix. Assume $\tau_{n-1} = \tau'_{n-1}$. By Lemma A.2, it suffices to show that $[\tau_n] = [\tau'_n]$ We show only that for every $\varphi \in [\tau_n]$, $\varphi \in [\tau'_n]$ holds, the other direction is identical. Let $\varphi \in [\tau_n]$. The base cases for the induction follow directly from S.1 and S.2. The case $\varphi = \bigcirc\varphi'$ can be ommited by definition of F.

- $\underline{\text{case } \varphi = \varphi_1 \wedge \varphi_2}$: Therefore by At.1.1, $\varphi_1 \in \tau_n$ and $\varphi_2 \in \tau_n$. By IH, therefore $\varphi_1 \in \tau'_n$ and $\varphi_2 \in \tau'_n$. So by At.1.1, $\varphi \in \tau'_n$.
- $\underline{\text{case } \varphi = \varphi_1 \wedge \varphi_2}$: Therefore by At.1.2, $\varphi_1 \in \tau_n$ or $\varphi_2 \in \tau_n$. Wlog. $\varphi_1 \in \tau_n$. By IH, therefore $\varphi_1 \in \tau'_n$. So by At.1.2, $\varphi \in \tau'_n$.
- $\underline{\text{case } \varphi = \varphi_1 \boldsymbol{U} \varphi_2}$: By definition of F, we know that $\varphi_2 \in \tau_n$. By IH, we get $\varphi_2 \in \tau'_n$ and therefory by At.2, $\varphi \in \tau'_n$.
- $\underline{\text{case } \varphi = \varphi_1 \boldsymbol{R} \varphi_2}$: Assume $\varphi \notin \tau'_n$. Therefore $\neg\varphi = \neg\varphi_1 \boldsymbol{U} \neg\varphi_2 \in \tau'_n$. Therefore by definition of F, $\neg\varphi_2 \in \tau'_n$, so $\varphi_2 \notin \tau'_n$ by At.8 and At.1.3. By IH, therefore $\varphi_2 \notin \tau_n$, which contradicts the consistency condition At.3.
- $\underline{\text{case } \varphi = \forall x{:}p.\,\varphi'}$ and $\underline{\text{case } \varphi = \exists x{:}p.\,\varphi'}$ follow from the definition of accepting runs and the fact that $\mathcal{A}_{\varphi',\beta}$ and $\mathcal{A}_{\neg\varphi',\beta}$ cannot be both accepting on the same run.

□

Lemma A.4. *For every two accepting runs $\pi, \pi' \in \Pi_{\mathcal{A}_{\Phi,\beta}(\sigma)}$ where $\pi = (\tau_0, \eta_0)\cdots(\tau_n, \eta_n)$ and $\pi' = (\tau'_0, \eta'_0)\cdots(\tau'_n, \eta'_n)$ and every i, $0 \leq i < n$ holds:*

$$\text{If } \tau_{i+1} = \tau'_{i+1}, \text{ then } \tau_i = \tau'_i.$$

Proof. This lemma is shown similarly to the last one, the only difference is that the Next-case is obligatory. The base cases and the cases for conjunction, disjunction and quantifiers are the same as in Lemma A.3, the other cases are shown here:

- case $\varphi = \bigcirc\varphi'$. Assume $\varphi \notin \tau'_i$, therefore $\neg\varphi = \bigcirc\neg\varphi' \in \tau'_i$. Therefore by S.3, both $\varphi' \in \tau_{i+1} = \tau'_{i+1}$ and $\neg\varphi' \in \tau'_{i+1} = \tau_{i+1}$ which contradicts At.1.3.
- case $\varphi = \varphi_1 \boldsymbol{U} \varphi_2$: If $\varphi_2 \in \tau_i$, by IH, we know that $\varphi_2 \in \tau'_i$. Therefore by At.3, $\neg\varphi_1 \boldsymbol{R} \neq \varphi_2 = \neg\varphi \notin \tau'_i$. By At.8, $\varphi \in \tau'_i$ follows.

  Otherwise, if $\varphi_2 \notin \tau_i$, by S.4, we know that $\varphi_1 \in \tau_i$ and $\varphi \in \tau_{i+1}$. By IH, we know that $\varphi_1 \in \tau'_i$. Assume $\varphi \notin \tau_i$, then $\neg\varphi_1 \boldsymbol{R} \neg\varphi_2 = \neg\varphi \in \tau'_i$. Therefore, by S.5, we know that $\neg\varphi \in \tau'_i$. This contradicts At.1.3, since $\varphi \in \tau'_i$, as well.
- case $\varphi = \varphi_1 \boldsymbol{R} \varphi_2$: If $\varphi_1 \in \tau_i$, we know by At.3 and IH that $\varphi_1 \in \tau'_i$ and $\varphi_2 \in \tau'_i$. By At.3, therefore $\varphi \in \tau'_i$.

  Otherwise, we know that $\neg\varphi_1 \in \tau_i$. Therefore, by S.5 $\varphi \in \tau_{i+1}$. Assume that $\varphi \notin \tau'_i$. Therefore $\neg\varphi_1 \boldsymbol{U} \neg\varphi_2 = \neg\varphi \in \tau'_i$. Since $\varphi_2 \in \tau_i$ by At.3, $\varphi_2 \in \tau'_i$ follows by IH. Therefore, by S.4, $\neg\varphi \in \tau'_{i+1} = \tau_{i+1}$ wich contradicts At.1.3.

The non-parametric cases can, again, be omitted since it suffices to show $[\tau_i] = [\tau'_i]$. □

Lemma A.2 and Lemma A.3, together with the fact that every accepting run starts in the same state (namely q_0) and the fact that the η_i only depend on the τ_i show the uniqueness of the accepting run. ■

Proof of Theorem 2.2

Since we know by Theorem 2.1 that there is at most one accepting run and by Theorem 1.5 that there is at most one measure, it remains to show that whenever $\alpha_{\Phi,\sigma,\beta}$ exists, there is an accepting run with result $\alpha_{\Phi,\sigma,\beta}$. First, we construct a state-sequence and then show that the corresponding measuring-assignments and updates compute the right values for every parameter.
Assume that $\alpha_{\Phi,\sigma,\beta}$ exists and let $\alpha := \alpha_{\Phi,\sigma,\beta}$. Let $n := |\sigma|$.
We define the state-sequence $\tau_0, ..., \tau_n$ with $\tau_0 = q_0$ and for every i, $1 \leq i \leq n$:

$$\tau_i \quad := \quad \{\varphi \in cl(\Phi) \mid (\sigma, i-1, \alpha, \beta) \models \varphi\}$$

It is easy to show that every τ_i is a well-defined atom and that for every i, $\tau_{i-1} \xrightarrow{\sigma[i-1]} \tau_i$.
It remains to show that the corresponding measuring computes the result α.
For this purpose, let $\pi := (\tau_0, \eta_0) \cdots (\tau_n, \eta_n)$ such that for every i and $\varphi \in cl(\Phi)$. $\varphi \in \tau_i$ iff $(\sigma, i-1, \alpha, \beta) \models \varphi'$. $(*)$ The fact that π is accepting follows directly from its construction and the definition of F. Remains to show that $res(\pi) = \alpha$.
We show this claim by induction over the level of $\mathcal{A}_{\Phi,\beta}$, but first we need some properties of the local result. Let $\mathcal{K}^*$ denote the set of all parameters in Φ that don't occur inside the scope of a quantifier.

Lemma A.5.

1. $\forall k \in \mathcal{K}_\uparrow.\ res_{loc}(\pi)(k) = \max\{j - i \mid \Diamond_{\leq k}\varphi \in \tau_i \wedge \varphi \in \tau_j \wedge \forall j', i \leq j' < j. \varphi \notin \tau_{j'}\}$
2. $\forall k \in \mathcal{K}_\downarrow.\ res_{loc}(\pi)(k) = \min\{j - i + 1 \mid \Box_{\leq k}\varphi \in \tau_i \wedge \varphi \notin \tau_j\}$

Proof. Follows directly from the construction of the update function. □

Lemma A.6.
For every accepting run $\pi = (\tau_0, \eta_0) \cdots (\tau_n, \eta_n)$ where $\varphi \in \tau_i$ iff. $(\sigma, i-1, \alpha^{\succ}_{\varphi,\sigma,\beta}, \beta) \models \varphi$ holds:

$$\alpha^{\succ}_{\Phi,\sigma,\beta}|_{\mathcal{K}^*} \quad = \quad res_{loc}(\pi)$$

Proof. Let $\alpha := \alpha^{\succ}_{\varphi,\sigma,\beta}$, $res := res_{loc}(\pi)$ and $\varphi \in \tau_i$ iff. $(\sigma, i-1, \alpha, \beta) \models \varphi$. $(*)$
Assume that there is a $k \in \mathcal{K}^*$ s.t. $\alpha(k) \neq res(k)$.

- case $k \in \mathcal{K}_\uparrow$:
  - case $\alpha(k) < res(k)$: Therefore there are j, i s.t. $j - i = res(k)$, $\Diamond_{\leq k}\varphi \in \tau_i$, $\varphi \in \tau_j$ and for each j', $i \leq j' < j$. $\varphi \notin \tau_{j'}$ by Lemma A.5.1. Let i, j be fix.

    Since $\Diamond_{\leq k}\varphi \in \tau_i$, we know by $(*)$ that $(\sigma, i-1, \alpha, \beta) \models \Diamond_{\leq k}\varphi$. By definition of $\models$ this means that there is a $i' \leq \alpha(k)$ s.t. $(\sigma, i+i'-1, \alpha, \beta) \models \varphi$. Therefore by $(*)$, $\varphi \in \tau_{i+i'}$. Since $i' \leq \alpha(k) < res(k) = j - i$, we know that there is a j' with $i \leq j' < j$ with $\varphi \in \tau_{j'}$. Contradiction!
  - case $\alpha(k) > res(k)$: By Lemma A.6.1, we know that for every i where $\Diamond_{\leq k}\varphi \in \tau_i$ holds that $\alpha(k) > \min\{j \mid \varphi \in \tau_{i+j}\}$. Let i such that $\Diamond_{\leq k} \in \tau_i$ be arbitrary, but fix.

    Therefore there is a $j < \alpha(k)$ with $\varphi \in \tau_{i+j}$, i.e. by $(*)$, there is a $j \leq \alpha(k) - 1$ s.t. $(\sigma, i+j-1, \alpha, \beta) \models \varphi$. Therefore $(\sigma, i+j-1, \alpha[{}^{\alpha(k)-1}/_k], \beta) \models \Diamond_{\leq k}\varphi$. Since i is chosen arbitrarily and $\alpha[{}^{\alpha(k)-1}/_k] \sqsupset \alpha$ by definition, we know that α cannot be a measure. Contradiction!
- case $k \in \mathcal{K}_\downarrow$ is shown almost identically using Lemma A.5.2

□

Lemma A.6 directly shows the base case for the induction over the level of the measuring automaton. Because of the restrictions for parameters 1.3 and 1.4, it also shows the inductive step for every parameter not occuring inside the scope of a quantifier. Remains to show that parameters occuring inside the scope of a quantifier have the correct value. Assume there is a k that appears inside the scope of a quantified subformula and that $res(\pi)(k) \neq \alpha(k)$. We consider only the case that $k \in \mathcal{K}_\uparrow$, the other case is alsmost identical.

- case k is in the scope of $\forall x{:}p.\,\varphi$.

  If $res(\pi)(k) > \alpha(k)$, by IH this means that there is a position i and a $d \in \sigma_p[i]$ s.t. $\forall x{:}p.\,\varphi \in \tau_{i+1}$ and $\alpha^{\succ}_{\varphi,\sigma[i\ldots],\beta}(k) = res(\pi)(k)$. Since by $(*)$ therefore $(\sigma, i, \alpha, \beta) \models \forall x{:}p.\,\varphi$, we know that $\alpha^{\succ}_{\varphi,\sigma[i\ldots],\beta}$ is not optimal for k and therefore no measure. Contradiction!

  Otherwise, if $res(\pi)(k) > \alpha(k)$, it can be easily shown similarly to the proof of Lemma A.6 that the specification is also satisfied under $\alpha[{}^{\alpha(k)-1}/_k]$ which means that α is not a measure. Contradiction!
- case k is in the scope of $\exists x{:}p.\,\varphi$ follows from the inductive hypothesis and the construction of the result, since every existential obligation takes the optimal sub-result wrt. $\trianglerighteq$ which follows directly the semantical definition of the existential quantification.

■