Informatics

Real-Time Logics and Reasoning

Answerset Semantik für Metric Temporal Logic

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Nikolaus Manes Funk, BSc
Matrikelnummer 01425838

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Prof. Dr. Thomas Eiter

Wien, 24. Juni 2021

Nikolaus Manes Funk	Thomas Eiter

Informatics

Real-Time Logics and Reasoning

Answer Set Semantics for Metric Temporal Logic

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Nikolaus Manes Funk, BSc
Registration Number 01425838

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr. Thomas Eiter

Vienna, 24th June, 2021

Nikolaus Manes Funk   Thomas Eiter

Erklärung zur Verfassung der Arbeit

Nikolaus Manes Funk, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 24. Juni 2021

Nikolaus Manes Funk

Abstract

Stream Reasoning is a young field of study that focuses on the high level evaluation of data streams. The LARS framework was created for this task utilizing a logic based approach, modelling data streams and programs in Answer Set Programming fashion. We introduce a new formalism for the LARS framework by translating the LARS logic programming language to Metric Temporal Logic. This logic expresses formulas more succinctly than others and uses timed state sequences instead of streams as semantic structures, which distinguish between system- and real-time. By translating LARS to MTL and back we create a new MTL fragment, called Metric LARS, which describes the LARS-translatable forumlas, and provide answer set semantics for timed state sequences and Metric LARS programs. Timed state sequences are explored in detail on their own and we give various options to translate them to LARS streams and back, with methods that can be chosen depending on the semantics of the data to be modelled. We further create fragments of Metric LARS for Metric Interval Temporal Logic, Signal Temporal Logic and their respective semantic structures and have a look at Timed Propositional Temporal Logic and its relation with Metric LARS.

Kurzfassung

Stream Reasoning ist ein neues Forschungsfeld, das sich auf die semantische Auswertung von Datenstreams konzentriert. Für dieses Feld wurde das LARS Framework geschaffen; es benutzt Logik Programmierung und Datenströme der Answerset Semantik. In dieser Arbeit führen wir neue Formalismen für das LARS Framework ein, indem wir die LARS Programmiersprache in Metric Temporal Logic (MTL) übersetzen. Mit dieser Logik lassen sich Formeln sehr prägnant ausdrücken und sie wertet diese Formeln auf Timed State Sequences anstatt auf Datenströmen aus. Diese Timed State Sequences unterscheiden zwischen einer System- und einer Echtzeit. Durch die Übersetzung von LARS auf MTL erhalten wir ein neues MTL Fragment, das wir Metric LARS nennen. Dieses Fragment umfasst alle Formeln, die sich in LARS übersetzen lassen. Für Metric LARS und Timed State Sequences wird auch Answerset Semantik eingeführt. Timed State Sequences werden weiters auch genauer behandelt und wir beschäftigen uns mit verschiedenen Möglichkeiten diese in Datenströme zu übersetzen. Hier gibt es mehrere Optionen, jenachdem wie die temporalen Daten modelliert werden sollen. Wir werfen außerdem noch einen Blick auf andere mit MTL verwandte Real-Time Logics wie Metric Interval Temporal Logic, Signal Temporal Logic und deren semantischen Strukturen, sowie Timed Propositional Temporal Logic, welche die Mutter dieser Familie der Logik ist.

Contents

CHAPTER 1

Introduction

1.1 Background

Stream Reasoning is a novel field of research that is concerned with the monitoring, evaluation and reasoning of data streams. Although a clear definition cannot be made, Stream Reasoning distinguishes itself from Stream Processing by being a more high-level and abstract approach to working with temporal data. As there are currently more data streams around the world than ever before in the history of computer science, Stream Reasoning is of growing interest and will be a key research topic in the coming years. The key features of stream reasoning include dropping and filtering data to cope with the excessive amount of information contained in a stream and that evaluation of programs has to be done incrementally, continuously and repeatedly because of newer information constantly flowing in during the run-time of a stream and possibly invalidating assumptions of the past. This interest was already monitored and observed by Dell'Agilo et al. in 2017 [DDVvHB17].

With this new field of research emerging, there are many possibilities to choose from, but none is as perfectly suited for this task as logic. While there are many formalisms available to model data with, Beck et al. chose 2018 in their paper "LARS: A logic-based framework for Analytic Reasoning over Streams" [BDTE18] to work with temporal logic using Answer Set Semantics. This has many great properties to reason over streaming data.

Temporal Logic [BvBW06] is a form of Modal Logic [BvBW06] that models time as a structure where each point in time has a set of logic formulas that describes its state. Many such logics exist, but the most prominent one is Linear Temporal Logic [Pnu77], which uses a structure of linear, subsequent time points. Since data streams are not a lot different in that they usually consist of a linear sequence of data points – although different representations may consist of parallel streams or include technical details such as transmission delays – it was a logical option to use for Stream Reasoning.

The LARS framework utilizes linear temporal logic and semantics of Answer Set Programming [BET11][Lif19][RN02]. This is a formalism for reasoning over sets of logic inference rules called programs. Rules consist of a body that has to be fulfilled so that the head of the rule can be inferred. This technique has many great properties for reasoning tasks and can be adapted for most problem instances.

1.2 Problem Statement

As was said, the LARS framework only uses semantic structures of linear temporal logic, but it is unclear how it is related to other temporal logics. The goal of this thesis is to introduce a different type of logic to Answer Set Semantics for Stream Reasoning and investigate its relationship with LARS: Real-Time logic, first and foremost Metric Temporal Logic. This logic is special in that its semantic structures, called Timed State Sequences, have a way to distinguish between system time and real time. It was first introduced in 1993 by Alur and Henzinger [AH93] and was intended to be a more succinct fragment of Timed Propositional Temporal Logic, the first real-time logic devised in 1989 [AH89]. Furthermore we want to show ways to model data streams as timed state sequences and align it with various other semantic structures of other real-time logics.

1.3 Overview of Results

In this thesis we enrich the LARS framework with an additional form of reasoning using Metric Temporal Logic and fully translate it into the LARS logic language. With this we also introduce Answer Set Semantics for MTL and Timed State Sequences. We further explore the relationship of the resulting MTL logic programs with other real-time logics, such as Metric Interval Temporal Logic [AFH96], Signal Temporal Logic [MN04] and Timed Propositional Logic [AH89]. We also examine the relationships of timed state sequences with the structures underlying these logics and describe how to use their respective intended semantics with timed state sequences.

1.4 Related Work

There have been multiple research papers aiming to provide Metric Temporal Logic with Answer Set Semantics. One of the earliest was Datalog MITL by Brandt et al. 2017 [BKK$^+$17], which focuses on the Metric Interval Temporal Logic fragment of MTL, which will be explained in the later chapters of this thesis. There have already been applications of MITL used in reasoning as seen in the work of Fu and Topcu 2015 [FT15]. A different form of datalog for MTL was proposed by Wałęga et al. in 2019 [WKG19], called Forward Propagating DatalogMTL. The authors decided to omit changes to the past by the logical inference done on the logic rules, hence the name. This is again a datalog type that is based on MITL rather than MTL itself. This logic is not evaluated on timed state sequences, but rather a rational timeline, with a mapping function that assigns sets of propositions to points in the timeline.

In a recent paper, Wałęga et al. [TCWCGK21] did further work on DatalogMTL and extended it for stratified datalog programs with MTL temporal operators. Here they proposed negation as failure as it is defined in most answer set formalisms. This means that facts that are not known to be true are assumed to be false. They also progressed the understanding of their language by researching the reasoning complexity of negation-free DatalogMTL programs, which was found to be PSPACE-complete in data complexity and EXPSPACE for combined complexity. This extension of DatalogMTL is again evaluated on a rational timeline.

Stream reasoning has features that make it amenable for ASP; the LARS framework and its success is one of the witnesses of this view. The framework was proposed by Beck et al. in 2018 [BDTE18] and is the basis for this thesis. Another example of ASP for stream reasoning is Temporal Datalog by Ronca et al. [RKG$^+$18]. Here the authors thoroughly examine the possibilities of stream reasoning in datalog that is extended with a temporal sort and successor function. They also examine new problems in stream reasoning, such as the Definitive Time Point problem and define their characteristics, as well as their complexity requirements. For instance DTP is PSPACE-complete in data complexity. These problems were designed to challenge existing stream reasoning frameworks and inspire the design of future systems. They found that logic-based stream reasoning is feasible for non-recursive Temporal Datalog queries.

Recently there has been more interest sparking in MTL answer sets, as Cabalar et al. [CDSS20] show in their 2020 paper "Towards Metric Temporal Answer Set Programming". This thesis is not influenced by their work on Answer Set Semantics for MTL, as the approaches are radically different in nature and were developed independently at roughly the same time. The biggest difference is found in the authors advancing the logic of Here-and-There with an MTL extension and base it on Metric Equilibrium Logic. The results are the fragments of MHT and MEL. While using the logic of Here-and-There is a valid idea for the establishment of answer set semantics, it is not really suited for the application in stream reasoning. This was found by Beck et al. in 2016 [BDTE16]. In this paper the authors proposed a form of LARS that included the HT semantics, but they found that the notion of windows cannot be properly expressed in Here-and-There logic.

1.5 Organization

In the following we will explain the content to come in the following chapters.

- In Chapter 2 we introduce the various preliminary topics this thesis is built upon. We start with the answer set semantics that will later make up the core contribution of the thesis. Next we have a look at Modal Logic and the Possible World Semantics, focusing on temporal logic. From this we start with stream reasoning and the LARS framework. This includes an overview of stream reasoning, as well as a first quick look at the syntax and semantics of LARS, that will get more in depth in the later chapters.
- Chapter 3 is dedicated solely to Metric Temporal Logic. We provide an in-depth explanation of the syntax and semantics of this logic, as well as proofs for expressivity and complexity of this logic. This serves as the foundation of all findings of this thesis.
- In Chapter 4, the main contributions are presented. We first show the translation of the Plain LARS fragment, which omits most of the challenging features of LARS. To properly translate all LARS formulas we introduce a new normal form, called Nested Window Normal Form. This helps with the succinct expression of MTL formulas compared to LARS. From there we provide a translation of general LARS and MTL formulas. In the next step we generalize the translation to window functions of unknown complexity, by introducing Oracle Intervals. With this all done, we have a look at translations for LARS streams to timed state sequences and back. We also define the subset operator for timed state sequences, that creates a partial order on the set of all timed state sequences. This is needed to define the Answer State Semantics of Metric LARS, the set of all LARS-translatable MTL formulas. We finish this chapter with semantic properties of answer state sequences and a small note to the size increase of the translation.
- Chapter 5 is concerned with the relationship of Metric LARS with other real-time logics such as Metric Interval Temporal Logic, Signal Temporal Logic and Timed Propositional Temporal Logic. For this reason we provide translations for the semantic structures of these logics and a Metric LARS fragment for STL and MITL. As TPTL is more expressive than MTL, we only mention it for future research.
- In Chapter 6 we summarize our findings and contributions and have a final outlook on what is yet to come. This includes the obvious lack of a working reasoner and translator for Metric LARS, but also some research on the complexity and expressivity of Metric LARS.

CHAPTER 2

Preliminaries

In this chapter we have a look at the work previously done in the field of stream reasoning and all accompanying technologies necessary to gain a solid foundation to understand the thesis at hand.

In the first section, we will look at answer set programming and the Datalog syntax for first order logic. Next we have a look at modal logics and linear temporal logic in particular. In the last section the LARS framework and stream reasoning will be explained. If not explicitly said, the information found in this section originate from *Artificial Intelligence: A modern Approach* by Russel and Norvig [RN02].

2.1 Answer Set Programming

Answer Set Programming is a certain type of logic programming, where models are defined as so-called *Answer Sets*, centered around the so-called *stable model semantics*. This is the foundation of a lot of work done in the field of automated reasoning and will be the basis for the semantics defined in this thesis.

2.1.1 Syntax

A given extended answer set program $\mathfrak{P}$ consists of a language Σ that is made up of a set P of predicate names, a set C of constants and a set V of variables. Predicates have a positive arity assigned to them determining the number of parameters that they accept. These parameters are called *terms*, which can be either variables or constants. The structure of a predicate name with parameters is called an atom and typically of the form

$$predicate\_name/n(term_1, term_2, ...term_n) \quad n \geq 0 \tag{2.1}$$

If there are no variables in the vector of terms, the atom is called *ground*. Atoms can be negated in two different fashions: strong or classical negation and default negation, also called *negation as failure*. *Literals* are atoms that are either strongly negated or positive. Any positive or negative literal can also be default negative negated. We will not consider the special semantics of strong negation in the following, as this can be emulated by creating two different mutually exclusive atoms.

Definition 1 *An answer set program $\mathfrak{P}$ consists of a finite set of inference rules of first order logic. The most general case is the extended rule, which is of the following form:*

$$a_1 \vee a_2 \vee ... \vee a_m \leftarrow b_1 \wedge b_2 \wedge ... \wedge b_k \wedge not\ b_{k+1} \wedge ... \wedge not\ b_n. \qquad k \leq n \tag{2.2}$$

where we denote the left-hand-side of the rule by $head(r) = \{a_j \mid 1 \leq j \leq m\}$, also called the rule head and the right-hand-side $body(r) = \{b_1, ..., b_k,\ not\ b_{k+1}, ...,\ not\ b_n\}$ is called the rule body. We further discriminate between the positive and the negative body of a rule r. The positive body is defined as $body^+(r) = \{b_i \mid 1 \leq i \leq k\}$, while the negative body consists of all atoms $body^-(r) = \{b_i \mid k < i \leq n\}$.

The type of rule is based on the following conventions:

- fact: a rule is called a fact iff $m = 1$ and $n = 0$, so a rule with just one atom in the head and no body.
- strict: a rule is proper iff $n = k$ and $m \geq 1$
- positive: a rule is positive iff it includes no negated atoms.
- disjunctive: a rule is disjunctive iff $m > 1$, so there are multiple atoms occurring in the head.
- normal: a rule is normal iff it is proper and not disjunctive.
- Horn: a rule is Horn iff $m = 1$, $k = n$ and it is positive.
- extended: a rule is called extended iff it is disjunctive and includes default negation.
- safe: a rule is called safe iff every term in the rule occurring in a default negated atom also occurs in a positive atom.

- constraint: a rule is a constraint iff $m = 0$, so a rule without a head. In this case the rule infers $\bot$.

If each rule in a program has a certain property, the program adopts the attribute for itself. For instance, if every rule in a program is strict, the program is also called strict.

Example 1 *Here we present some rules with their properties:*

- $p \leftarrow q$. *this rule is proper, ground, normal, positive, safe and Horn.*
- $a \leftarrow$. *this rule is a fact.*
- $\leftarrow not\ a$. *this is a constraint. The atom a is default negated.*
- $a_1 \lor a_2 \leftarrow b_1, not\ b_2$. *is extended.*
- $a(X) \leftarrow b(X), not\ c(X)$. *is safe.*
- $a(X) \leftarrow not\ b(X),\ not\ c(X)$. *is not safe.*

2.1.2 Semantics

After defining the syntax of an answer set program, next we want to define the semantics of it.

The semantics of an answer set program $\mathfrak{P}$ is based on a structure composed of a set C of constants, a set P of predicate names and their arity and a set V of variables. The so-called *Herbrand Universe* of program $\mathfrak{P}$ is defined as all the terms occurring in it:

$$HU(\mathfrak{P}) = C \tag{2.3}$$

The *Herbrand Base* of a program $\mathfrak{P}$ is defined as:

$$HB(\mathfrak{P}) = \{p/n(\bar{c}) \mid p/n \in P,\ \bar{c} \subseteq HB(P),\ |\bar{c}| = n,\ n \in N\} \tag{2.4}$$

Atoms can be *ground*, which means that they do not include any variables. Similarly, a rule that only consists of ground atoms is also called ground. The semantics of answer set programs are defined for ground programs, composed entirely of ground rules.

A given set of ground atoms is called *interpretation*. An interpretation I of a program $\mathfrak{P}$ is a subset of Herbrand Base $HB(\mathfrak{P})$. Intuitively an interpretation fulfills an atom if it is in the interpretation, similarly, a ground rule is fulfilled if the positive body is part of the interpretation and the negative part is not. When this is the case, the head is inferred, meaning it has to be in the interpretation as well. This is defined as follows:

$$\{b_1, ..., b_k\} \subseteq I \land \{b_{k+1}, ..., b_n\} \cap I = \emptyset \to \{a_1, ..., a_m\} \cap I \neq \emptyset \tag{2.5}$$

When Condition (2.5) holds for a ground rule, the interpretation is called a classical model of the given rule. If it is a classical model of every rule in a program, the interpretation is a classical model of the program.

In the case of a non-ground program $\mathfrak{P}$, we first derive the ground version of it *grnd($\mathfrak{P}$)*. This process is called *grounding* and uses the substitution function θ. The application of θ on a rule substitutes the variables occurring in its atoms with the constants of the program, which is denoted by *grnd(r)*, where r is the given rule. This creates all possible ground instances of a rule:

$$\theta : V(r) \mapsto HU(\mathfrak{P}),\ r \in \mathfrak{P} \tag{2.6}$$

$$grnd(\mathfrak{P}) = \bigcup_{r \in \mathfrak{P}} grnd(r) \tag{2.7}$$

where $V(r)$ is the set of variables occurring in rule r.

More important than such classical models are stable models for the purpose of defining semantics. Stable models are generated with the Gelfond-Lifschitz Reduct $\mathfrak{P}^I$ of a ground program $\mathfrak{P}$ and an interpretation I [GL88]:

$$\mathfrak{P}^I = \{head(r) \leftarrow body^+(r) \mid r \in \mathfrak{P},\ body^-(r) \cap I = \emptyset\} \tag{2.8}$$

If the interpretation I is a subset-minimal classical model of $\mathfrak{P}^I$, it is called an *Answer Set* of program $\mathfrak{P}$. Because of language features like disjunctive rules and default negation, there may be more than just a single answer set (or conversely no answer set at all). For this purpose, the set of all answer sets of a program $\mathfrak{P}$ is denoted by $AS(\mathfrak{P})$. Since every answer set is minimal, different answer sets are incomparable and the following condition holds:

$$A_1, A_2 \in AS(\mathfrak{P}) \land A_1 \subseteq A_2 \to A_1 = A_2 \tag{2.9}$$

Non-ground programs first have to be grounded to be evaluated by applying the substitution with θ. For these programs, all answer sets that are found for the ground version, are also answer sets for the non-ground programs.

$$AS(\mathfrak{P}) = AS(grnd(\mathfrak{P})) \tag{2.10}$$

Example 2 *Consider the program*

$$\begin{aligned}\mathfrak{P} = \{\ & a.\\ & b.\\ & c \leftarrow b.\\ & d \leftarrow a,\ not\ b.\ \}\end{aligned}$$

Program $\mathfrak{P}$ is ground already, so we do not need to substitute any variables. Instead we proceed to apply the GL Reduct for the interpretation $I = \{a, b, c\}$:

$$\begin{aligned}\mathfrak{P}^I = \{\ & a.\\ & b.\\ & c \leftarrow b.\ \}\end{aligned}$$

As it can be seen, the last rule can be applied and I is in fact an answer set of $\mathfrak{P}$. If we change the program, not to include the fact b., the answer set changes:

$$\begin{aligned}\mathfrak{Q} = \{\ & a.\\ & c \leftarrow b.\\ & d \leftarrow a,\ not\ b.\ \}\end{aligned}$$

Here we use the interpretation $J = \{a, c, d\}$. With this we apply the reduct to receive:

$$\begin{aligned}\mathfrak{Q}^J = \{\ & a.\\ & c \leftarrow b.\\ & d \leftarrow a.\ \}\end{aligned}$$

We can clearly see that J is a classical model of $\mathfrak{Q}$, but can not qualify for being an answer set of $\mathfrak{Q}$. This is because there exists another interpretation $J' = \{a, d\}$, that is a classical model of the program and subsumes the interpretation J.

2.2 Linear Temporal Logic

In this section we will have a look at the temporal family of modal logic with the example of linear time logic. For this we have a look at modal logic and the possible world semantics, again based on [RN02] first and proceed with LTL [Pnu77].

2.2.1 Modal Logic

Compared to classical first-order logic, modal logic is an extension concerned with multiple modalities at once. Whereas a regular formula of logic can adopt a global truth value in the interpretation structure, in modal logic there exists a notion of different localities. Developed by various philosophers in the past, this logic features a semantics using so called *possible worlds* and their interactions inside a structure of worlds. Depending on the type of modal logic in use, they might use different names and properties. Syntactically, two new operators are introduced:

- The possibility operator or diamond operator: $\Diamond\,\phi$
- The necessity or box operator: $\Box\,\phi$
- Both can be expressed in terms of the other as a property of the formal definition: intuitively $\Diamond\,\phi = \neg\Box\neg\phi$ and $\Box\,\phi = \neg\Diamond\neg\phi$

Where ϕ is any formula of modal logic. This creates the following general syntax of modal logic:

$$\phi ::= p \mid \bot \mid \top \mid \neg\,\phi \mid \phi_1 \rightarrow \phi_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \Diamond\,\phi \mid \Box\,\phi \tag{2.11}$$

where p is a propositional atom. Each type of modal logic can however define new operators and may have its own name for the box and diamond operators.

2.2.2 Possible World Semantics

A *Kripke Structure*, named after its inventor Saul Kripke, is an abstract structure that can be visualized as a form of directed graph, where the nodes correspond to worlds and the connections between them as accessibility relations among worlds.

Definition 2 *A Kripke Structure is a tuple $\langle\mathfrak{W}, \mathfrak{R}\rangle$, where $\mathfrak{W}$ is the set of worlds and $\mathfrak{R} = \{xRy \,\|\, x, y \in \mathfrak{W}\}$ is the set of connections between such worlds. Each world $w \in \mathfrak{W}$ itself is a set of modal logic formulas.*

This can be for instance a group of people knowing each other, commandments in philosophy, or a structure of timed events happening subsequently. The semantics of the modal logic in use highly depends on the type of structure they are evaluated against. Classical logical formulas can now be true or false in each world individually, meaning that a formula ϕ can be true in one world, but false in another. The additional operators as seen above define logical properties of the relations between the worlds. The diamond operator $\Diamond\,\phi$ expresses that ϕ is true in one or more connected worlds. On the other hand, the box operator $\Box\,\phi$ requires that ϕ is true in all connected worlds to evaluate to true. The semantics for each formula can be defined as follows and are evaluated on one world $w \in \mathfrak{W}$ inside the Kripke Structure:

$$\begin{array}{lll}
\langle\mathfrak{W},\mathfrak{R}\rangle, w \models p & \textit{iff} & p \in w \\
\langle\mathfrak{W},\mathfrak{R}\rangle, w \models \neg\,\phi & \textit{iff} & \langle\mathfrak{W},\mathfrak{R}\rangle, w \not\models \phi \\
\langle\mathfrak{W},\mathfrak{R}\rangle, w \models \phi_1 \wedge \phi_2 & \textit{iff} & \langle\mathfrak{W},\mathfrak{R}\rangle, w \models \phi_1 \wedge \langle\mathfrak{W},\mathfrak{R}\rangle, w \models \phi_2 \\
\langle\mathfrak{W},\mathfrak{R}\rangle, w \models \phi_1 \vee \phi_2 & \textit{iff} & \langle\mathfrak{W},\mathfrak{R}\rangle, w \models \phi_1 \vee \langle\mathfrak{W},\mathfrak{R}\rangle, w \models \phi_2 \\
\langle\mathfrak{W},\mathfrak{R}\rangle, w \models \phi_1 \to \phi_2 & \textit{iff} & \langle\mathfrak{W},\mathfrak{R}\rangle, w \not\models \phi_1 \vee \langle\mathfrak{W},\mathfrak{R}\rangle, w \models \phi_2 \\
\langle\mathfrak{W},\mathfrak{R}\rangle, w \models \Diamond\,\phi & \textit{iff} & \exists v : wRv \wedge \langle\mathfrak{W},\mathfrak{R}\rangle, v \models \phi \quad v \in \mathfrak{W} \\
\langle\mathfrak{W},\mathfrak{R}\rangle, w \models \Box\,\phi & \textit{iff} & \forall v : wRv \to \langle\mathfrak{W},\mathfrak{R}\rangle, v \models \phi \quad v \in \mathfrak{W}
\end{array} \tag{2.12}$$

If a formula is satisfied on every world of a structure, the entire structure is then said to entail the formula.

Example 3 *Assume a Kripke Structure with worlds $\mathfrak{W} = \{w_1 = \{p\}, w_2 = \{q\}, w_3 = \{q\}\}$ and relations $\mathfrak{R} = \{(w_1, w_2), (w_2, w_1), (w_1, w_3), (w_3, w_1)\}$. We now want to check whether the structure is cyclic regarding proposition p. This means, that every world satisfying p and being connected to another world, also requires that these worlds are connected back in a directed cycle. We can model this property with the formula $p \to \Box\Diamond p$. Since only world w_1 satisfies p, w_2 and w_3 trivially satisfy the formula. The right hand side of the formula now has to be evaluated on w_1. Intuitively the meaning of the subformula is that every reachable world has at least one path that reaches a world that entails p. The world w_1 reaches both remaining worlds w_1Rw_2 and w_1Rw_3. This means the subformula $\Diamond p$ has to hold on w_2 and w_3. Since in both cases there exists a relation w_2Rw_1 and w_3Rw_1, the formula holds. Since this formula holds on every world in the structure $\langle\mathfrak{W},\mathfrak{R}\rangle$, it is cyclic regarding proposition p.*

2.2.3 Linear Temporal Logic

The most important type of modal logic for this thesis is the group of temporal modal logics. These modal logics model different views on how timed processes can be expressed using logic. There are many different approaches to this question but only some will be mentioned here. The most important distinction is which worlds – in this case we talk about one world being a point in time – should be linear or, if it is possible to consider multiple time lines. In this thesis we will only look at forms of logic using the first kind of structure. This understanding of time is used in Linear Time Logic as described by Amir Pnueli [Pnu77]. In his paper various new operators were introduced to provide syntax and semantics for properties of linear time sequences. These include

- The Next operator: $\oplus\phi$ or $X\phi$
- The Eventually operator: $\Diamond\phi$ or $F\phi$
- The Globally operator: $\Box\phi$ or $G\phi$
- The binary Until operator: $\phi_1 \mathcal{U}\,\phi_2$
- The binary Releases operator: $\phi_1 \mathcal{R}\,\phi_2$

where ϕ and ψ are formulas of linear time logic. As in other modal logics, a world is a set of formulas true at this point. In the context of time, each world gets a time point t assigned, where $t \in T$, the (possibly infinite) time domain of the structure. With the operation $\leq$ we can create a total order of time points, corresponding to the set of relations $\mathfrak{R}$ in other modal logics which is therefore given implicitly by the set of worlds. The semantics for the other operators is defined as follows:

$$\begin{aligned}
\mathfrak{W}, t &\models \oplus\phi && \textit{iff} \quad \mathfrak{W}, t+1 \models \phi \\
\mathfrak{W}, t &\models \Diamond\phi && \textit{iff} \quad \exists t' : \mathfrak{W}, t' \models \phi, \quad t < t' \\
\mathfrak{W}, t &\models \Box\phi && \textit{iff} \quad \forall t' : \mathfrak{W}, t' \models \phi, \quad t < t' \\
\mathfrak{W}, t &\models \phi_1 \mathcal{U}\,\phi_2 && \textit{iff} \quad \exists t' : \mathfrak{W}, t' \models \phi_2 \wedge \forall t'' : \mathfrak{W}, t'' \models \phi_1, \quad t \leq t'' < t' \\
\mathfrak{W}, t &\models \phi_1 \mathcal{R}\,\phi_2 && \textit{iff} \quad \mathfrak{W}, t \not\models \neg\phi_1 \mathcal{U} \neg\phi_2
\end{aligned} \tag{2.13}$$

Over the years, many extensions for different applications were made, most importantly PLTL, the Past Time Linear Time Logic, where all of the operators have a paired operator expressing processes in the past. It is however important to note that the expressiveness does not change with the introduction of new operators but is exponentially more succinct as shown by Markey [Mar03].

2.3 Stream Reasoning and the LARS Framework

In recent years the need for more abstract approaches for stream processing gave rise to a new area called stream reasoning [DDVvHB17]. Even though a clear hierarchy between the two can not be established as either the more general or more abstract version can be seen as leading the two to name the united field, they are closely related fields dedicated to work with streams of data. One such approach to stream reasoning is the LARS framework [BDTE18], which is a generic reasoner for linear temporal logic. In the following we will have a close look at first the field of stream reasoning and then the technology behind LARS.

2.3.1 Stream Reasoning

Stream Reasoning was established more than ten years ago, when the world got more and more reliant on data-heavy streams [DDVvHB17]. Ever since then, data streams got even bigger and low level processing is not sufficient anymore. For this, reasoning systems for ontologies or logic programming were tried and tested to faster process the highly semantic information passed through the wires of the world. Due to the highly interdisciplinary field, various ideas get exchanged and influence work of research in the area. Because of this not only logic programming but also a mix with statistical methods is used inside processing pipelines, making artificial intelligence tasks ever more efficient and precise. Future inventions in this area will have a big emphasis on scalability of systems and streams, as the trend towards an more and more connected world develops faster than ever [DEHLP18].

2.3.2 LARS Framework

In 2015 Beck et. al. proposed a new framework for stream reasoning using stable semantics and the advances in reasoning in answer set programming. The *Logic-based Framework for Analytic Reasoning over Streams*, short LARS, is a stream reasoner making use of a fragment of past linear time logic in the fashion of Datalog rules and inference systems. This is done treating time as an order $(\mathbb{N}, \leq)$ as the time ontology. The most prominent feature of the framework is the notion of custom and built-in window functions.

As it was seen in the semantics of PLTL, a formula quantified with the box or diamond operator gets evaluated over the entire stream – i.e. every point in time in the corresponding Kripke Structure. Since an indefinite stream can not be feasibly evaluated, a window function serves as an operation to truncate the stream in a given time interval. Such function can be defined inside the framework, with some specific functions predefined by the authors. These include the time-based window $\boxplus^{[i,j]}\phi$, the tuple-based window $\boxplus^{\#n}\phi$ and the partition-based window $\boxplus^{p}\phi$.

- The time-based window truncates the stream in the interval $[t-i, t+j]$ and evaluates the sub-formula solely in this section.
- The tuple-based window cuts the stream at the n-th last atom. This also means that there may be time points at the border of this interval, where only some of the tuples are taken.
- The partition-based window looks for the occurrence of the given predicate symbol p and creates a sub-stream between the current time and the last occurrence of the symbol in the stream.

Since all of these window functions define some interval of time or tuples, that is evaluated based on the current time of the system, they are called sliding window functions. On the other hand there are also filter windows, that do not truncate the data stream but rather omit any but some specified atoms. The discussed window function are all called sliding windows, which means that the resulting data stream is different at each time point. On the other hand there are also tumbling windows. These split the stream into parts of some specified length and are the same for all time points in their respective part.

For the case of nested windows, there is also a new reset operator $\triangleright\phi$ that resets the sub-stream to the input stream. The $@_t\phi$ operator checks whether formula ϕ holds at a specific point in time t.

The syntax for a LARS rule can then be defined as follows:

$$\begin{gathered}\phi_0 \leftarrow \phi_1, ..., \phi_k, not\ \phi_{k+1}, ..., not\ \phi_n. \\ \phi_i ::= p \mid \neg\,\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid \boxplus^w\,\phi \mid @_t\,\phi \mid \triangleright\,\phi \mid \Box\,\phi \mid \Diamond\,\phi\end{gathered} \tag{2.14}$$

where p is an atom as seen previously in the section on answer set programming and ϕ is a sub-formula of the same kind. As in ASP, the left-hand-side of the rule is called the head of the rule $Head(r)$ and the right side is called the body $Body(r)$. On a side note, there have been fragments of this syntax proposed, which we will have a look at later in this thesis.

Example 4 *Here we can see how rules can look like in LARS in the context of modelling the traffic of a city:*

- *Assume we want to model the participants in the flowing traffic, so noone who is currently in a queue. We can express this with the following rule:* $traffic(X, Time) \leftarrow @_{Time}\,car(X), not\ in\_queue(X, Time).$

- *Observing a bus at some time in a destination, that has the same ID as another destination, we may infer that there is a bus route between them:*
  $@_T route(X, Y, N) \leftarrow destination(X, I), destination(Y, I), @_T spotting(N, X), bus(N).$
- *This rule has no inteded semantic meaning other than showing what rules are possible in the very free syntax of LARS rules:*
  $p \leftarrow \boxplus^{\#5} \Diamond(p \rightarrow \boxplus^{\#2} \Box q), @_{T-2} p, cur(T).$

Rules in LARS have to be evaluated against a set of window functions W, which is assumed to be fixed $\boxplus^W \equiv W$, and a stream S, defined as follows:

Definition 3 ***LARS Stream.*** *A stream is a tuple $S = (T, \nu)$, where T is a closed interval of $\mathbb{N}$ and $\nu : \mathbb{N} \mapsto 2^G$, where G is the set of ground atoms. The function ν returns $\emptyset$ for all $n \in \mathbb{N} \backslash T$ and returns a subset of the set of ground atoms of the corresponding logic program otherwise.*
A sub-stream of a stream $S = (T, \nu)$ is a stream $S' = (T', \nu')$, where $T' \subseteq T$ and $\nu'(t') \subseteq \nu(t')$ for all $t' \in T$.

The semantics for each operation on a structure $M = \langle S^*, W, B \rangle$, where S^* is a stream, W is the set of windows and $B \subseteq G$ is the set of background data for a sub-stream S of S^* at time point $t \in T$ is given as follows:

$$\begin{array}{lll}
M, S, t \Vdash p & \textit{iff} & p \in \nu(t) \vee p \in B \\
M, S, t \Vdash \neg\phi & \textit{iff} & M, S, t \nVdash \phi \\
M, S, t \Vdash \boxplus^w \phi & \textit{iff} & M, S', t \Vdash \phi, \quad S' = w(S, t) \\
M, S, t \Vdash @_{t'}\, \phi & \textit{iff} & M, S, t' \Vdash \phi, \quad t' \in T \\
M, S, t \Vdash \triangleright\phi & \textit{iff} & M, S^*, t \Vdash \phi \\
M, S, t \Vdash \Box\, \phi & \textit{iff} & \forall t' : M, S, t' \Vdash \phi, \quad t' \in T \\
M, S, t \Vdash \Diamond\, \phi & \textit{iff} & \exists t' : M, S, t' \Vdash \phi, \quad t' \in T
\end{array} \tag{2.15}$$

A LARS program then consists of a set of LARS rules.

Definition 4 ***Answer Stream Semantics.*** *Consider a data stream D and a LARS program $\mathfrak{P}$. We define the interpretation stream I, with $D \subseteq I$, so that a structure is then called an interpretation for D of the form $M = \langle I, W, B \rangle$. An interpretation is then a model of a rule $r \in \mathfrak{P}$ at time t, denoted as $M, t \models r$, if $M, t \models Body(r) \rightarrow Head(r)$. The interpretation is a model of $\mathfrak{P}$ at time t, if $M, t \models r$ for each rule $r \in \mathfrak{P}$. It is then called a minimal model of $\mathfrak{P}$ for D, if there is no other model $M' = \langle I', W, B \rangle$ at time point t so that the relations $I' \subset I$ and $D \subseteq I'$ holds.*

This model is now called an *Answer Stream* if it is a minimal model of the reduct $P^{M,t}$ for D at time t. This reduct is based on the FLP-reduct as opposed to the Gelfond-Lifschitz reduct seen earlier. Intuitively, we disregard any rules that cannot fire with respect to the interpretation. Further M has to be minimal so that the conclusions made by the firing rules are always supported. This means that all intensional atoms, i.e. those that occur in some rule head, are based on some chains of rules consisting of extensional atoms, which are all those that do not occur in any rule head. All the facts that cannot be derived are assumed to be false, which corresponds to the closed world assumption.

Example 5 *Consider the propositional LARS program*

$$P = \{ \quad \begin{aligned} & a \leftarrow \boxplus^{[2,2]}\Box b,\ \boxplus^{[1,0]}\Diamond c. \\ & d \leftarrow \boxplus^{[0,3]}\Diamond c. \end{aligned} \quad \}$$

where all window functions are of the sliding interval type and the data stream D

t	*Propositions*
$0 \mapsto$	$\{b\}$
$1 \mapsto$	$\{b, c\}$
$2 \mapsto$	$\{b\}$
$3 \mapsto$	$\{c\}$
$4 \mapsto$	$\{c\}$

Now assume an interpretation stream I of the form

t	*Propositions*
$0 \mapsto$	$\{b, d\}$
$1 \mapsto$	$\{b, c\}$
$2 \mapsto$	$\{b\}$
$3 \mapsto$	$\{c\}$
$4 \mapsto$	$\{c\}$

This interpretation stream is then an answer stream of the program P for D at time point $t = 0$. This can be seen as the reduct $P^{M,0}$ includes only the second rule at said time point:

$$P^{M,0} = \{ \quad d \leftarrow \boxplus^{[0,3]}\Diamond c. \quad \}$$

We now need to evaluate the window function $\boxplus^{[0,3]}$ at time point 0. As it was explained, this function changes the evaluation of the formula on the stream to a substream as *defined in the window. This means we have to evaluate the formula* $\Diamond c$ *on the stream* S'

t	*Propositions*
$0 \mapsto$	$\{b\}$
$1 \mapsto$	$\{b, c\}$
$2 \mapsto$	$\{b\}$
$3 \mapsto$	$\{c\}$

As we can see, the formula $\Diamond c$ *is fulfilled as there is at least one occurrence in the stream.*

The complexity of evaluating LARS programs depends a lot on the window functions that are being used and whether nesting depth is bounded or not. Beck et. al. considered the problems of model checking and satisfiability in their paper. Model checking receives a given model structure I and a program $\mathfrak{P}$ to check whether it is an answer stream of the program, while satisfiability asks if there exists an answer stream satisfying the given program and data stream. The results are as follows [BDTE18]:

- Model Checking:
  - With bounded nesting depth: P for formulas and co-NP for programs
  - Without bounded nesting depth: PSPACE for formulas and PSPACE for programs
- Satisfiability:
  - With bounded nesting depth: NP for formulas and Σ_2^P for programs
  - Without bounded nesting depth: PSPACE for formulas and PSPACE for programs

CHAPTER 3

Metric Temporal Logic

In this section, we will describe the original work of Alur and Henzinger [AH93] in more detail concerning *Metric Temporal Logic* (in short *MTL*). Intuitively, we want to express state changes at arbitrary time steps as opposed to the Linear Temporal Logic, where each world is defined at each passing time point. In this case it is a time sequence with $t_{i+1} - t_i = 1 \; \forall i \in \mathbb{N}$, while in Metric Temporal Logic the difference can not be predicted.

3.1 Syntax

Metric Temporal Logic has some unique properties and operators that make it possible to express formulas with varying temporal distances. There is a big emphasis in this logic on the operators *„Since"* $\phi_1 \, \mathcal{S}_{[i,j]} \, \phi_2$ and *„Until"* $\phi_1 \, \mathcal{U}_{[i,j]} \, \phi_2$. According to Alur and Henzinger the syntax of MTL is given by:

$$\phi ::= p \mid \bot \mid \phi_1 \to \phi_2 \mid \; \oplus_{[i,j]} \phi \mid \; \ominus_{[i,j]} \phi \mid \phi_1 \, \mathcal{U}_{[i,j]} \, \phi_2 \mid \phi_1 \, \mathcal{S}_{[i,j]} \, \phi_2 \tag{3.1}$$

In Equation 3.1, p is an atom, ϕ_i are MTL formulas and $\oplus_{[i,j]} \phi$ is the Next operator, while $\ominus_{[i,j]} \phi$ denotes the Previous operator. This syntax subsumes full Boolean algebra, as shown in the following:

$$\top = \bot \to \bot \tag{3.2}$$

$$\begin{aligned} \neg \, \phi &= \neg\phi \vee \bot \\ &= \phi \to \bot \end{aligned} \tag{3.3}$$

$$\begin{aligned} \phi_1 \vee \phi_2 &= \neg\neg \, \phi_1 \vee \phi_2 \\ &= \neg \, \phi_1 \to \phi_2 \\ &= (\phi_1 \to \bot) \to \phi_2 \end{aligned} \tag{3.4}$$

$$\begin{aligned} \phi_1 \wedge \phi_2 &= \neg(\neg\phi_1 \vee \neg\phi_2) \\ &= \neg(\phi_1 \rightarrow \neg\phi_2) \\ &= (\phi_1 \rightarrow \neg\phi_2) \rightarrow \bot \\ &= (\phi_1 \rightarrow (\phi_2 \rightarrow \bot)) \rightarrow \bot \end{aligned} \tag{3.5}$$

The intervals of some operators may be changed to congruence relations $\equiv_d c$. Based on this syntax for formulas, we will define four more important operators, that are oriented on the possibility and necessity operator of classical modal logic. These operators are $\Diamond^{+}_{[i,j]}\,\phi$ and $\Diamond^{-}_{[i,j]}\,\phi$, describing the occurrence of an event in the at time point $t+i$ to $t+j$ (respectively $t-i$ to $t-j$ in the negative version of this operator).

Further there are additionally the $\boxplus_{[i,j]}\phi$ and $\boxminus_{[i,j]}\phi$ operators that describe that in the time interval starting at $t+i$ and ending in $t+j$ in every single passing time unit, formula ϕ is true (respectively $t-i$ until $t-j$). Syntactically these operators can be built with the Until and Since operators:

$$\begin{aligned} \Diamond^{+}_{[i,j]}\,\phi &= \top\,\mathcal{U}_{[i,j]}\,\phi \\ \boxplus_{[i,j]}\,\phi &= \neg\,\Diamond^{+}_{[i,j]}\,\neg\phi \\ \boxplus_{[i,j]}\,\phi &= \neg(\top\,\mathcal{U}_{[i,j]}\,\neg\phi) \end{aligned} \tag{3.6}$$

$$\begin{aligned} \Diamond^{-}_{[i,j]}\,\phi &= \top\,\mathcal{S}_{[i,j]}\,\phi \\ \boxminus_{[i,j]}\,\phi &= \neg\,\Diamond^{-}_{[i,j]}\,\neg\phi \\ \boxminus_{[i,j]}\,\phi &= \neg(\top\,\mathcal{S}_{[i,j]}\,\neg\phi) \end{aligned} \tag{3.7}$$

The operators as shown in Equation 3.7 are only available in the Past Time Extension of Metric Temporal Logic, denoted as MTL$_P$. Implicitly the MTL$_P$ extension is assumed to be given when referring to MTL in this work.

3.2 Semantics

The semantics of MTL relies on a data stream for state changes called a *Timed State Sequence*, denoted as S. This structure is defined over a set P of finite propositions and a time domain T. More precisely, we define a timed state sequence as follows:

Definition 5 ***Timed State Sequence.*** *A Timed State Sequence S over a time domain T and set of propositions P is a pair $S = \langle \sigma, \tau \rangle$ consisting of an ordered list of states $\sigma = \langle \sigma_1, \sigma_2, ... \rangle$, where every state $\sigma_i \in \sigma$ is a subset of P, and an injective function $\tau : \mathbb{N} \mapsto T$. This function maps each index in the sequence of states σ to a point in time in the time domain T.*

It is important to note that this function is not necessarily surjective, so the function $\tau^{-1} : T \mapsto \mathbb{N}$ is not defined on all points in T. Also pay attention to the fact that a time domain may be infinite. For the sake of complexity we will further assume a finite time domain.

To check whether a formula ϕ is true in the given timed state sequence S at time point t, we define the following semantics for each formula as it was written in Equation 3.1:

$$
\begin{array}{lll}
S,t \models \bot & \textit{iff} & S,t \not\models \bot \\
S,t \models p & \textit{iff} & p \in \sigma_i \wedge \tau(i) = t \\
S,t \models \phi_1 \rightarrow \phi_2 & \textit{iff} & S,t \not\models \phi_1 \;\vee\; S,t \models \phi_2 \\
S,t \models \oplus_{[i,j]}\phi & \textit{iff} & S,\tau(i) \models \phi \wedge \tau(i) \in [t+i, t+j], \\
 & & s.t.\ \tau(i-1) \leq t < \tau(i) \\
S,t \models \ominus_{[i,j]}\phi & \textit{iff} & S,\tau(i) \models \phi \wedge \tau(i) \in [t-j, t-i], \\
 & & s.t.\ \tau(i) < t \leq \tau(i+1) \\
S,t \models \phi_1\, \mathcal{U}_{[i,j]}\, \phi_2 & \textit{iff} & \exists x :\ S,\tau(x) \models \phi_2 \wedge \forall y :\ S,\tau(y) \models \phi_1, \\
 & & \tau(x) \in [t+i, t+j],\ \tau(y) \in [t+i, \tau(x)] \\
S,t \models \phi_1\, \mathcal{S}_{[i,j]}\, \phi_2 & \textit{iff} & \exists x :\ S,\tau(x) \models \phi_2 \;\wedge\; \forall y :\ S,\tau(y) \models \phi_1, \\
 & & \tau(x) \in [t-j, t-i],\ \tau(y) \in [\tau(x), t-i]
\end{array}
\tag{3.8}
$$

Subsequently, we can also define the semantics of the derived MTL operators:

$$\begin{aligned}
S,t &\models \diamondplus_{[i,j]}\ \phi && \textit{iff} \quad S,t \models \top\, \mathcal{U}_{[i,j]}\, \phi && (3.9)\\
S,t &\models \diamondminus_{[i,j]}\ \phi && \textit{iff} \quad S,t \models \top\, \mathcal{S}_{[i,j]}\, \phi && \\
S,t &\models \boxplus_{[i,j]}\, \phi && \textit{iff} \quad S,t \models \neg\ \diamondplus_{[i,j]}\, \neg\, \phi && (3.10)\\
& && \textit{iff} \quad \exists x:\ S,\tau(x) \not\models \phi \rightarrow \exists y:\ S,\tau(y) \models \bot, && \\
& && \qquad \tau(x) \in [t+i, t+j], \tau(y) \in [t+i, \tau(x)] && \\
S,t &\models \boxminus_{[i,j]}\, \phi && \textit{iff} \quad S,t \models \neg\ \diamondminus_{[i,j]}\, \neg\, \phi && (3.11)\\
& && \textit{iff} \quad \exists x:\ S,\tau(x) \not\models \phi\ \rightarrow\ \exists y:\ S,\tau(y) \models \bot, && \\
& && \qquad \tau(x) \in [t-j, t-i], \tau(y) \in [\tau(x), t-i] &&
\end{aligned}$$

As we have seen previously, the mapping τ does not cover every point in time of the time domain T. This means that if a formula has to be evaluated over a time point $t \in T$ that is not corresponding to a state in the timed state sequence, the world at t is considered as being empty, i.e. devoid of any given information. Since this property was explained by Alur and Henzinger [AH93] we call it the *Intended Semantics of Metric Temporal Logic*. For operators using congruence instead of regular intervals, the formulas hold only in the special sets of time points defined by the relation. The relation $\equiv_2 1$ for instance would only check for the timed states at time $t \in \{2n+1\}_{n \geq 0}$.

Example 6 *Consider the MTL formula $\oplus p$. Given a structure $S = \langle \{\{p\}, \{p\}\},$ $\{\{0 \mapsto 0\}, \{1 \mapsto 2\}\} \rangle$, the formula evaluated at $t = 0$ is false: $S, 0 \not\models \oplus p$, since $S, 1 \not\models p$. This is due to the fact that the time point 1 has no information concerning the existence of fact p, as the state with index 1 is occurring at time point 2. The formula is therefore true at time point $t = 1$: $S, 1 \models \oplus p$, since $S, 2 \models p$.*

3.3 Expressivity and Complexity

In their paper Alur and Henzinger showed that the validity problems is for MTL and for MTL$_P$ EXPTIME-complete. The upper bounds were found with size constraints on Tableau procedures, similar to those done in [Pnu77], which will be defined in the following. This section serves as a summary of the work in [AH93], but contains a full complexity proof that was omitted in the original paper.

3.3.1 Deciding MTL$_P$

In the following we will construct a tableau based proof of the decision procedure for MTL$_P$. First we start with the observation that a finite tableau can always be constructed in MTL$_P$. For this we use the term *State Requirement* that denotes a formula or subformula that creates a constraint on a certain state in the time sequence.

Lemma 1 *A given MTL$_P$ formula can only produce a finite number of state requirements.*

Proof. Assume a formula of the form $\diamondplus_{[0,j]}\ \phi$. For the formula to be true in some structure at some time point t, it means that it has to be fulfilled either now – i.e. at t – or at any other point in the interval $[t, t+j] \in T$. Since time passes at the maximum of one state per unit we can now see that the formula is true iff ϕ is true at the current time or when $\diamondplus_{[1,j]}\ \phi$ holds. We can now see that when this is done j times, we end up with a formula that can be evaluated at each timed state in the future until j units have passed. Since this number of states is finite the formula can only produce a finite number of requirements. This is true for all temporal operators defined above. □

Theorem 1 *Deciding the validity of MTL$_P$ formulas is feasible in deterministic exponential time in $O(C \cdot N)$, where $N - 1$ is the number of boolean and temporal connectives and $C - 1$ is the largest constant that occurs as an interval end point.*

Proof. Given a MTL$_P$ formula ϕ, we want to determine whether ϕ is valid. This means that we have to check whether its negation $\neg\phi$ is satisfiable. Let $Closure(\neg\phi)$ define the closure set of $\neg\phi$, which is the smallest set containing $\neg\phi$ that is closed under the Sub

operation:

$$\begin{aligned}
Sub(\psi \to \chi) &= \{\psi, \chi\} \\
Sub(\oplus\, \psi) &= \{\psi\} \\
Sub(\ominus\, \psi) &= \{\psi\} \\
Sub(\psi\, \mathcal{U}_I\, \chi) &= \{\psi, \chi\} \cup \{\oplus(\psi\, \mathcal{U}_{I-\delta}\, \chi) \parallel \delta \geq 0\} \\
Sub(\psi\, \mathcal{S}_I\, \chi) &= \{\psi, \chi\} \cup \{\ominus(\psi\, \mathcal{S}_{I-\delta}\, \chi) \parallel \delta \geq 0\}
\end{aligned} \tag{3.12}$$

For the interval operation we define that $I - \delta = \emptyset$ for all $\delta > c$, where c is the interval endpoint – this constraint is also present for past time intervals in the reversed form. Now consider the numbers $N - 1$ – the number of temporal and boolean operators occurring in ϕ – and $C - 1$ – the biggest interval boundary. In the way we defined the set $Closure(\neg\phi)$, we see that the number of subformulas inside the set has to be bounded by the number of operators times the number intervals are split. Since the greatest arity of some temporal operator defined above is binary and the highest number of splits is given by the biggest interval, we define the upper bound for the magnitude of the closure as $|Closure(\neg\phi)| \leq 2C \cdot N$. This assumes all operators are binary temporal with the same maximum sized intervals.

We restrict the proof to timed state sequences with time differences between states $\tau(i+1) - \tau(i), i \geq 0$ of at most

$$K = C \cdot \prod_{i \geq 0} d_i$$

where d_i denotes all constants occurring in congruence relations $\equiv_d c$. This boundary is important for the finite state character of the state sequence.

The information regarding time given implicitly in the state sequence can now be modeled in

- Time difference propositions $Tdiff_\delta^-$, where $0 \leq \delta \leq K$. This represents the current time δ in the initial state of the timed state sequence and the time difference compared to the last state in all other timed states.
- Time difference propositions $Tdiff_\delta^+$, where $0 \leq \delta \leq K$. This denotes the difference to the successor state rather than the predecessor.
- Time congruence propositions $Tcong_{K.t}$, where $0 \leq t < K$. This shows the remainder t of the current time modulo K.

Now let $Closure^*(\neg\phi)$ be the union of the closure of $\neg\phi$ and all the time difference propositions $Tdiff_\delta^-$, $Tcong_{K.t}$ and $Tdiff_\delta^+$. We call any subset Φ of this set maximally consistent if and only if it satisfies the following conditions:

- $\mathit{Tdiff}^-_\delta \in \Phi$ for precisely one $0 \leq \delta \leq K$, where this $\delta \in T$ is referred to as δ^-_Φ
- $\mathit{Tdiff}^+_\delta \in \Phi$ for precisely one $0 \leq \delta \leq K$, where this $\delta \in T$ is referred to as δ^+_Φ
- $\mathit{Tcong}_{K.t} \in \Phi$ for precisely one $0 \leq \delta < K$, where this $\delta \in T$ is referred to as γ_Φ
- $\bot \notin \Phi$
- $\psi \to \chi \in \Phi$ if either $\psi \notin \Phi$ or $\chi \in \Phi$
- $\psi \, \mathcal{U}_I \, \chi \in \Phi$ for an interval I, iff
  - $I \neq \emptyset$ and
  - either $0 \in I$ and $\chi \in \Phi$, or $\psi \in \Phi$ and $\oplus(\psi \, \mathcal{U}_{I - \delta^+_\Phi} \, \chi) \in \Phi$
- $\psi \, \mathcal{U}_{\equiv_d c} \, \chi \in \Phi$ iff either $\gamma_\Phi \equiv_d c$ and $\chi \in \Phi$ or $\psi \in \Phi$ and $\oplus(\psi \, \mathcal{U}_{\equiv_d c} \, \chi) \in \Phi$
- $\psi \, \mathcal{S}_I \, \chi \in \Phi$ for an interval I, iff
  - $I \neq \emptyset$ and
  - either $0 \in I$ and $\chi \in \Phi$, or $\psi \in \Phi$ and $\ominus(\psi \, \mathcal{S}_{I - \delta^-_\Phi} \, \chi) \in \Phi$
- $\psi \, \mathcal{S}_{\equiv_d c} \, \chi \in \Phi$ iff either $\gamma_\Phi \equiv_d c$ and $\chi \in \Phi$ or $\psi \in \Phi$ and $\ominus(\psi \, \mathcal{S}_{\equiv_d c} \, \chi) \in \Phi$

We consider a tableau $\mathfrak{T}(\neg\phi)$ to be a directed graph with vertices of all maximally consistent subsets of $\mathit{Closure}^*(\neg\phi)$. A direct edge is established from vertex Φ to vertex Ψ if the following conditions hold:

- $\delta^+_\Phi = \delta^-_\Psi$
- $\gamma_\Psi \equiv_K \gamma_\Phi + \delta^+_\Phi$
- For all $\oplus_I \psi \in \mathit{Closure}(\neg\phi)$, $\oplus_I \psi \in \Phi$ iff $\delta^+_\Phi \in I$ and $\psi \in \Psi$
- For all $\oplus_{\equiv_d c} \psi \in \mathit{Closure}(\neg\phi)$, $\oplus_{\equiv_d c} \psi \in \Phi$ iff $\gamma_\Psi \equiv_d c$ and $\psi \in \Psi$
- For all $\ominus_I \psi \in \mathit{Closure}(\neg\phi)$, $\ominus_I \psi \in \Psi$ iff $\delta^-_\Psi \in I$ and $\psi \in \Phi$
- For all $\ominus_{\equiv_d c} \psi \in \mathit{Closure}(\neg\phi)$, $\ominus_{\equiv_d c} \psi \in \Psi$ iff $\gamma_\Phi \equiv_d c$ and $\psi \in \Phi$

All models of the given formula $\neg\phi$ now correspond to some infinite path through the tableau graph $\mathfrak{T}(\neg\phi)$. This means that $\neg\phi$ is satisfiable if and only if the following conditions of a path $\{\Phi_0, \Phi_1, ...\}$ hold:

- $\neg\phi \in \Phi$
- Φ_0 contains no formula $\ominus\psi$
- For all $i \geq 0$ and intervals I, $\psi\ U_I\ \chi \in \Phi_i$ implies $\chi \in \Phi_j$ for some $j \geq i$ with $\sum\limits_{i \leq k < j} \delta^+_{\Phi_k} \in I$
- For all $i \geq 0$, $\psi\ U_{\equiv_d c}\ \chi \in \Phi_i$ implies $\chi \in \Phi_j$ for some $j \geq i$ with $\gamma_{\Phi_j} \equiv_d c$

Because of the upper bound we specified earlier, we know that the tableau $\mathfrak{T}(\neg\phi)$ contains $O(K \cdot 2^{C \cdot N})$ vertices. Each vertex has at most $O(C \cdot N)$ elements, so the graph $\mathfrak{T}(\neg\phi)$ can be constructed and checked for infinite paths in deterministic exponential time. □

3.3.2 Complexity of MTL$_P$

The next step is to define the algorithmic complexity for the decision procedure of validity problems of MTL$_P$ as defined above.

Theorem 2 *The validity problems for MTL and MTL$_P$ are EXPSPACE-complete.*

Proof. From the Theorem 1 and Savitch's Theorem [Sav70] it follows that the non-deterministic tableau is in EXPSPACE. To show the lower bound for it being EXPSPACE-hard, we use EXPSPACE-bounded Turing machines as was done by Alur and Henzinger to prove the complexity of *Timed Temporal Logic* (TPTL) in [AH89] (Theorem 2, second part).

For this we consider a deterministic 2^n-space-bounded Turing machine $\mathfrak{M}$. We construct an MTL$_P$ formula ϕ_X for each input X with length n, that has a length of $O(n \cdot log\ n)$. This formula has to be valid iff $\mathfrak{M}$ accepts X. From the hierarchy theorem of space we know that there has to be a constant $c > 0$ so that every Turing machine that solves the validity problem for formulas ϕ of length l takes $S(l) \geq 2^{c \cdot l / log\ l}$ space infinitely often. This means given the initial tape contents X we only have to construct:

- A sufficiently succinct formula ϕ_X that describes the unique computation of $\mathfrak{M}$ on X as an infinite sequence of propositions and

- A sufficiently succinct formula ϕ_{ACCEPT} that characterizes the computation of $\mathfrak{M}$ on X as accepting.

The implication $\phi_X \rightarrow \phi_{ACCEPT}$ is then valid iff $\mathfrak{M}$ accepts the input X.

Each tape symbol i is represented by a proposition p_i, where p_0 is the special *blank* symbol. For every state j of $\mathfrak{M}$ we use a proposition q_j, where q_0 is the initial state of the Turing machine. Additionally there are three abbreviations following from this:

$$\begin{aligned}
\hat{p} : &\qquad p_i \wedge \bigwedge_{i' \neq i} \neg p_{i'} \wedge \bigwedge \neg q_j \\
r_{i,j} : &\qquad p_i \wedge q_j \wedge \bigwedge_{i' \neq i} \neg p_{i'} \wedge \bigwedge_{j' \neq j} \neg q_{j'} \\
s : &\qquad \bigwedge \neg p_i \wedge \bigwedge \neg q_j
\end{aligned} \tag{3.13}$$

Each configuration of $\mathfrak{M}$ is represented by $\hat{p}$-state sequences of length 2^n that are separated by void s-states. The position of the read-write head is represented by an r-state $r_{i,j}$, where the first index is the position on the tape and the second index references the current state. The transition function of $\mathfrak{M}$ is denoted by $f_{\mathfrak{M}} : P \times Q \times R \rightarrow P \cup Q \cup R$, where each input set P, Q and R ranges over all propositions of $\hat{p}_i, r_{i,j}$ and s, so that it properly models the behaviour of the machine. The following two conditions determine the computation of $\mathfrak{M}$ on X:

- The Turing machine will always start with the initial configuration.
- Every configuration follows from the previous one by a move of $\mathfrak{M}$.

The sequence of symbols on the tape is denoted by X_i with $i \in [0, n]$. We now want to formulate both conditions as formulas in MTL$_P$. First, since between each time state exactly one moment passes, we add the formula $\boxplus_T \oplus_{[1,1]} \top$ to ϕ_X. This formula can be obtained from the state counter in TPTL, which looks as follows: $\Box x.Oy.y = x + 1$. We then model the constraints for the behaviour of the Turing machine:

$$\phi_{INITIAL} = s \wedge \oplus_T r_{X1,0} \wedge \bigwedge_{i \in [2,n]} \boxplus_{[i,i]} \hat{p}_{X_i} \wedge \boxplus_{[n+1,2^n]} \hat{p}_0 \tag{3.14}$$

$$\begin{aligned}
\phi_{MOVE} = &\boxplus_T (s \rightarrow \diamondplus_{[2^n+1,2^n+1]} s) \wedge \\
&\bigwedge_{P,Q,R} \boxplus_T ((P \wedge \oplus_{[1,1]} Q \wedge \oplus_{[2,2]} R) \rightarrow \diamondplus_{[2^n+2,2^n+2]} f_{\mathfrak{M}}(P, Q, R))
\end{aligned} \tag{3.15}$$

The formula $\phi_{INITIAL}$ models the original bounded tape and configuration of the machine, so that the first state is the void state s as defined before. The next time step is the position of the read-write head on the tape symbol X_1, so the first symbol in the given input X, in state 0. The third part of the formula states that the remaining tape contents appear in the order of the input, while the space between tape location $n+1$ (the end of the input definition) to 2^n has to be empty. This creates the exponential boundary of the tape. The second formula ϕ_{MOVE} has again two parts. The first part requires that between each machine configuration of 2^n $\hat{p}$-states, there is one void configuration s. The second part demands that each triplet of configurations, consisting of either r, $\hat{q}$ or s configuration, creates some configuration in the next iteration of the configuration of $\mathfrak{M}$. This choice of predicate depends on the encoding of the transition function $f_{\mathfrak{M}}$. For instance writing in state j on input i the symbol k and then moving to the right and entering state j' can be seen as $f_{\mathfrak{M}}(\hat{p}_i, r_{i',j}, \hat{p}_{i''}) = \hat{p}_k$ and then in the next time state $f_{\mathfrak{M}}(r_{i',j}, \hat{p}_{i''}, \hat{p}_{i'''}) = r_{i'',j'}$.

The computation is now accepting X if and only if the accepting state F occurs:

$$\phi_{ACCEPT} = \bigvee_{i \in X} \diamondplus_T \, r_{i,F} \tag{3.16}$$

The length of $\phi_{INITIAL}, \phi_{MOVE}$ and ϕ_{ACCEPT} is then $O(n \cdot log\ n), O(n)$ and $O(1)$ respectively. This means that the upper bound of $O(n \cdot log\ n)$ was in fact correct for ϕ_X. $\square$

3.3.3 Expressive Completeness of MTL$_P$

Next we will have a look at the expressive completeness of MTL$_P$, which can express the non-elementary first-order language $\mathfrak{L}_T$.

Let $\mathfrak{L}$ denote the first-order language with uninterpreted unary predicate symbols and an order predicate $\leq$ evaluated over the set of natural numbers $\mathbb{N}$. This order predicate entails the existence of the constant 0 as $\forall y.(x \leq y)$ and the successor function $y = x + 1$ as $x < y \wedge \forall z.(x < z \rightarrow y \leq z)$, when considering the set of natural numbers. In $\mathfrak{L}$ we only consider formulas without any free variables, i.e. variables that do not occur in the scope of a quantifier.

We now change the domain of the language $\mathfrak{L}$ to make it express timed state changes by adding in a time function τ to associate each state with a time. To model the properties of time, the function τ has to be monotonic as time is not known to go backwards in the observable universe. The newly defined language, now called $\mathfrak{L}_T$, then has two different sorts, one for the states and one for the time in the time domain $(T, \preceq)$, linked together by the function τ. Since it is a fragment of first-order logic, we only allow quantification over individuals. This language can now express any given timed state sequence (σ, τ).

Theorem 3 *For every formula ϕ of $\mathfrak{L}_T$, there is a formula ψ of MTL$_P$ such that for every initial timed state sequence S, S is a model of ϕ if and only if S is also a model of ψ.*

Proof. Let $M_T(\phi)$ denote the set of models of ϕ with time domain T. We will now construct a PTL formula ϕ' with additional time-difference propositions Tdiff_t and $\mathit{Tdiff}_{\geq t}$ and time congruence propositions $\mathit{Tcong}_{d.t}$, so that $M^*_T(\phi) = M(\phi')$. As an additional constraint, we only allow these time propositions to be out of scope of any temporal operators, or directly linked to a next operator $\oplus$.

Consider only *initial* models of MTL$_P$ formulas, i.e. those where the proposition $\tau(0) = 0$ holds. From ϕ' we derive the desired MTL$_P$ formula by eliminating the time difference and congruence propositions. Each occurrence of Tdiff_t, $\mathit{Tdiff}_{\geq t}$ and $\mathit{Tcong}_{d.t}$ is replaced by $\top$ if $t = 0$, and by $\perp$ if it is any other number. The subformulas $\oplus\mathit{Tdiff}_t$ are replaced with $\oplus_{[t,t]}\top$, $\oplus\mathit{Tdiff}_{\geq t}$ is replaced with $\oplus_{\geq t}\top$ and $\oplus\mathit{Tcong}_{d.t}$ with $\oplus_{\equiv_d t}\top$. Only the next operator has to be subscripted with time constraints, because of the semantics of the operators. As a reminder, these require the next given time state to be in the subscripted interval. □

CHAPTER 4

Reasoning in Metric Temporal Logic

In this section we will explain how we can use Metric Temporal Logic with past time operators to reason over streaming data. Since names for this type of datalog-type language are already using *datalogMTL* or *datalogMITL*, we name the result of this thesis *Metric LARS*.

The translation is created using an incremental approach to defining and then refining translations to the LARS framework, that is based on Linear Temporal Logic. We start with the easiest of such languages, called plain LARS, then building more complex operator constructs and negation, ending with nested window functions. Finally we will define Answer Set Semantics for Metric LARS.

4.1 General Definitions

To start, we will first define the most important properties of a possible language Σ for programs of LARS or MTL_P:

Definition 6 ***The Program Language.*** *We define three sets that make up the language of a program $\mathfrak{P}$, the finite set of constants C, the set P of all predicate names and the set of variables V. A denotes the set of all the atoms that can be created from the combination of predicate names and constants.*

This is the same as for regular answer set programs. Since Metric Temporal Logic only works with propositions in A, we will also reduce the expressiveness of LARS to these atoms. This means that grounding a program is not necessary, as per definition, all rules already are ground.

4.2 Plain Metric LARS

4.2.1 Plain LARS

The first translation is concerned with the least expressive formalism of the LARS framework. The language is called plain LARS and features datalog-style rules with minimal complexity. A rule in plain LARS looks like the following:

$$\begin{aligned} &\alpha \leftarrow \beta_1,\, \beta_2,\, \ldots,\, \beta_n. \\ &\textit{where}\;\; \alpha ::= p \mid @_t p \\ &\qquad\quad \beta_i ::= p \mid @_t p \mid \boxplus^n \Box p \mid \boxplus^n \Diamond p \\ &\qquad\quad p \in A \end{aligned} \tag{4.1}$$

where $\boxplus^n$ denotes the time-based window operator of LARS. Other than in regular LARS, the window operators are only evaluated in the past; for the sake of ease we will focus on time-based windows in the following. A more detailed look at window operators will be given in the later sections of this chapter. Atoms that occur only in the body of such a rule are called external, while those that occur in at least one rule head are called internal. The semantics of Plain LARS on a discrete data stream is given in Chapter 2.

4.2.2 Translation

The biggest difference in syntax between LARS and MTL is the notion of window operators. As these are not present in regular MTL, they present the biggest difficulty when trying to find a suitable translation. Since the binary connectors $\wedge$ and $\rightarrow$ are the same in both LARS and MTL, there only needs to be a translation of $@_t p, \boxplus^w \Box p$ and $\boxplus^w \Diamond p$.

Proposition 1 ***Plain LARS to MTL$_P$ Translation.*** *We propose the following translation for temporal operators.*

$$\begin{array}{lcl} \textit{LARS} & & \textit{MTL}_P \\ \boxplus^n \Box p & \hat{=} & \boxminus_{[0,n]}\, p \\ \boxplus^n \Diamond p & \hat{=} & \Diamondminus_{[0,n]}\, p \\ @_t p & \hat{=} & \boxminus_{[t',t']}\, p,\; t' = \hat{t} - t \\ & \hat{=} & \Diamondminus_{[t',t']}\, p,\; t' = \hat{t} - t \end{array} \tag{4.2}$$

where $\hat{t}$ is the current time. This variable is necessary, as MTL only defines time intervals relative to the current timed state, so in order to model the absolute time t, as required *by the @ operator of LARS, we have to subtract the absolute time from the current time, to receive the relative time interval.*

Proof. $\boxplus^n \Box p \rightarrow \boxminus_{[0,n]} p$: The window operator cuts the current stream into a substream, reaching from the current point in time, back n time points. Assume in this substream p always holds, fulfilling $\Box p$. This means that the same property holds on the interval $[-n, 0]$, with 0 being the current time. This fulfills $\boxminus_{[0,n]} p$.

$\boxplus^n \Box p \leftarrow \boxminus_{[0,n]} p$: Assume now $\boxminus_{[0,n]} p$ is given on a data stream. If we cut this same interval and make it a substream, every point in time satisfies p. This fulfills $\boxplus^n \Box p$.

In a similar fashion $\boxplus^n \Diamond p \mathrel{\hat{=}} \diamondminus_{[0,n]}\, p$ can be shown. Of more interest is the translation $@_t p \mathrel{\hat{=}} \boxminus_{[t',t']}\, p \mathrel{\hat{=}} \diamondminus_{[t',t']}\, p$. As it was already said, the @ operator uses absolute time, rather than relative time like the window operators in LARS or interval operators in MTL. This means we cannot simply adopt t in the translation. Considering we know the current time $\hat{t}$, we obtain the relative time by subtraction. Once this is done, we revisit the definitions of semantics for @:

$$
\begin{aligned}
& S, \hat{t} \models @_t p \qquad \textit{iff} \quad S, t \models p \wedge S, t \models \top \\
& t = \hat{t} - t' \\
& \diamondminus_{[t',t']}\, p \;=\; \top\, \mathcal{S}_{[t',t']}\, p \\
& S, \hat{t} \models \top\, \mathcal{S}_{[t',t']}\, p \quad \textit{iff} \quad S, t \models p \wedge S, t \models \top \\
& \top\, \mathcal{S}_{[t',t']}\, p \mathrel{\hat{=}} @_t p \\
& \boxminus_{[t',t']}\, p \;= \neg \diamondminus_{[t',t']}\, \neg p \mathrel{\hat{=}} \neg @_t \neg p \;=\; @_t p
\end{aligned}
$$

As we can see all three expressions are identical. $\Box$

Example 7 *Take the rule $p \leftarrow \boxplus^5 \diamond q, r$. This rule states intuitively, that if there is the atom r given in the current moment and in the last five moments there was at least one occurrence of atom q, we can infer that in this moment atom p has to be true. Translating the rule to Plain Metric LARS it yields $p \leftarrow \boxminus_{[0,5]} q, r$.*

4.3 Extending Metric LARS to general LARS Formulas

As Plain LARS is just a fragment of the original logic, we now have a look at the full version. Extended LARS has three major differences compared to Plain LARS: default negation, nested window operators and window operators are allowed to define windows in the future. Since the syntactic properties of default negation and future windows are negligible, they will be discussed in the next section on the Answer Set Semantics of Metric LARS.

4.3.1 LARS Formulas

LARS uses not only standard components as elements in the rules, but rather complex LARS formulas. These can consist of the following subformulas:

$$\phi ::= p \mid \neg\, \phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid \boxplus^n \phi \mid @_t\, \phi \mid \,\triangleright \phi \mid \Box\, \phi \mid \Diamond\, \phi \tag{4.3}$$

As before the binary connectors $\wedge, \vee$ and $\rightarrow$ behave in the exact same way as in MTL, so we will not feature a proof on their translation. The issues arise with unbounded occurrences of window operators, always and finally. This translation therefore needs more steps than before. We start by converting an arbitrary LARS formula into *Nested Window Normal Form*. This is needed to reduce the number of subformulas in the further steps of the translation and helps us to reduce the exponential blowup in size.

4.3.2 Nested Window Normal Form

Nested Window Normal Form tries to bind each temporal operator with a window operator, which is necessary since MTL does not feature a distinction between a temporal operator and its interval on the stream. First, the formula has to be shaped into Negation Normal Form with the additional rules for temporal operators:

$$\begin{aligned}
\neg\Box\phi &\;\hat{=}\; \Diamond\neg\phi \\
\neg\Diamond\phi &\;\hat{=}\; \Box\neg\phi \\
\neg \boxplus^n \phi &\;\hat{=}\; \boxplus^n \neg\phi \\
\neg \triangleright \phi &\;\hat{=}\; \triangleright \neg\phi \\
\neg @_t\phi &\;\hat{=}\; \begin{cases} @_t\neg\phi & \textit{if } t \in T \\ \top & \textit{otherwise} \end{cases}
\end{aligned} \tag{4.4}$$

The window function and stream reset are not changed by a negation since the stream is not altered in a different way. The @ operator requires t to be in the current streams time domain T. If so, the operation will not be affected, however since it evaluates to $\bot$ otherwise, it returns $\top$ as the negated form. For the always and finally operator the rules are derived from their definition. Since the type of temporal operator does not matter for the translation, we propose a new notation:

Notation 1 ***Anonymous Temporal Operators.*** *The temporal operators of LARS $\Box, \Diamond, @$ are generalized by the temporal operator symbol $\odot$. Similarly, this will also be done with the MTL temporal operators $\diamondplus, \diamondminus, \boxplus, \boxminus$, with the difference that they may have a subscript for their interval like $\odot_I$.*

In the next step, each Window operator is pulled into the formula until it is *guarded* by a temporal operator.

Proposition 2 ***Preliminary Nested Window Normal Form.*** *We propose the following transition rules to achieve preliminary Nested Window Normal Form:*

$$
\begin{aligned}
\boxplus^n\ (\phi_1 \wedge \phi_2) &\quad\hat{=}\quad \boxplus^n\ \phi_1 \wedge \boxplus^n \phi_2 \\
\boxplus^n\ (\phi_1 \vee \phi_2) &\quad\hat{=}\quad \boxplus^n\ \phi_1 \vee \boxplus^n \phi_2 \\
\boxplus^n\ (\phi_1 \rightarrow \phi_2) &\quad\hat{=}\quad \boxplus^n\ \phi_1 \rightarrow \boxplus^n \phi_2 \\
\boxplus^n\ \boxplus^m \phi &\quad\hat{=}\quad \boxplus^{min(n,m)}\ \phi \\
\boxplus^n\ \odot_1 \ldots \odot_k\ \phi &\quad\hat{=}\quad \begin{cases} \bot & \textit{iff } \odot_i = @_t \wedge t \notin T \\ \boxplus^n\ \odot_k\ \phi & \textit{otw.} \end{cases} \\
\boxplus^n\ \triangleright \phi &\quad\hat{=}\quad \triangleright\ \phi \\
\triangleright \ldots \triangleright \phi &\quad\hat{=}\quad \triangleright\ \phi \\
\boxplus^n\ \alpha &\quad\hat{=}\quad \alpha
\end{aligned}
\tag{4.5}
$$

Here α is any ordinary formula, so that no temporal operator occurs in it. Each window operator that is paired with a proposition or with the reset operator can be removed from the formula. In the first case this is because the proposition is implicitly evaluated at the current time t, making the window obsolete and in the second case is simply reset to the last window, or the original stream should there exist no other window operator. Nested window operators cut the stream twice, which means that the smaller bound is taken and the windows can be subsumed. For the case of window operators that are paired with a stack of any length of temporal operators, it is equivalent to the last temporal operator used.

Example 8 *Take the LARS formula $\neg(\boxplus^n(p \rightarrow \Box q))$. In NNF this formula will look like this: $\boxplus^n(p \wedge \Diamond\neg q)$. In the next step we try to get the window as far as possible down to the temporal operators, creating the NWNF $p \wedge \boxplus^n \Diamond \neg q$.*

Definition 7 ***Window Introduction Rule.*** *In a special case we have a formula ϕ of the shape $\boxplus^n \odot \phi$, that starts with a window operator, followed by some temporal operator $\odot$. In the scope of this operator is a formula ϕ, in which a subformula $\odot\psi$ starting with a temporal operator occurs, that is not yet guarded by a window function. For this case there needs to be another rule, allowing us to pair this single operator, called the* Window Introduction Rule*:*

$$\odot\,\psi \mathrel{\hat{=}} \triangleright \boxplus^{min(n,\ldots)} \odot \psi \tag{4.6}$$

The new boundary of the window operator is derived from the minimum value of all windows that it is in the scope of to properly cut the stream according to the semantics of window functions.

Example 9 *To make Equation 4.6 more tangible, consider the formula $\boxplus^w \Diamond \boxplus^v \Box(p \rightarrow \Diamond q)$. With the formulas in Equation 4.5 this would already be finished as we cannot move the window further in. However we still want to guard each temporal operator with a window function. We employ the last rule to obtain $\boxplus^w \Diamond \boxplus^v \Box(p \rightarrow \triangleright \boxplus^{min(w,v)} \Diamond q)$, which satisfies this constraint.*

Theorem 4 ***Semantic Equivalence of Nested Window Normal Form.*** *Let ϕ be an arbitrary LARS formula. The Nested Window Normal Form $NWNF(\phi)$ is semantically equivalent to ϕ and can be obtained by applying the rules of the Preliminary NWNF (Proposition 2) and the Window Introduction Rule (Definition 7).*

Proof. The proof on semantic equivalence will be done inductively on the re-writing rules.

- We start with the first three rules as they are quite easy to show:

$$\begin{array}{lll} S,t \models \boxplus^n(\phi_1 \wedge \phi_2) & \textit{iff} & S',t \models \phi_1 \wedge \phi_2 \\ S,t \models \boxplus^n(\phi_1 \wedge \phi_2) & \textit{iff} & S',t \models \phi_1 \textit{ and } S',t \models \phi_2 \\ S,t \models \boxplus^n\phi_1 \wedge \boxplus^n\phi_2 & \textit{iff} & S',t \models \phi_1 \textit{ and } S',t \models \phi_2 \end{array}$$

  In the same fashion also $\vee$ and $\rightarrow$ can be proved:

$$\begin{array}{lll} S,t \models \boxplus^n(\phi_1 \vee \phi_2) & \textit{iff} & S',t \models \phi_1 \vee \phi_2 \\ S,t \models \boxplus^n(\phi_1 \vee \phi_2) & \textit{iff} & S',t \models \phi_1 \textit{ or } S',t \models \phi_2 \\ S,t \models \boxplus^n\phi_1 \vee \boxplus^n\phi_2 & \textit{iff} & S',t \models \phi_1 \textit{ or } S',t \models \phi_2 \end{array}$$

$$\begin{array}{lll} S,t \models \boxplus^n(\phi_1 \rightarrow \phi_2) & \textit{iff} & S',t \models \phi_1 \rightarrow \phi_2 \\ S,t \models \boxplus^n(\phi_1 \rightarrow \phi_2) & \textit{iff} & S',t \not\models \phi_1 \textit{ or } S',t \models \phi_2 \\ S,t \models \boxplus^n\phi_1 \rightarrow \boxplus^n\phi_2 & \textit{iff} & S',t \not\models \phi_1 \textit{ or } S',t \models \phi_2 \end{array}$$

- Next we have a look at the rule for merging window functions. Assume the window function 1 cuts the stream at some point in the past. the next window operator then has two possibilities

  - Either it cuts it further down than the previous function,
  - Or it tries to cut it beyond the limit which results in the same substream as before.

  Thus the overall new window will only employ the minimal interval imposed on the stream. Inductively, if there are more window functions following the two first ones, the observation still holds, resulting in the function $min(n_1, ...n_m)$, with m being the number of consecutive window functions.

- Next we have the case of multiple consecutive temporal operators. First we have a look at the different combinations for two such operators and then we inductively add any number of additional ones.

  - $S, t \models \Box\Box\phi$ iff every time point t' satisfies $S, t' \models \Box\phi$, which is then true if every time point t'' satisfies $S, t'' \models \phi$. This means to satisfy the entire formula the following has to hold: $\forall t' \forall t'' : S, t'' \models \phi$. As we can see, the first temporal operator did not matter in the evaluation, which means that it can savely be removed, resulting in the equal formula $S, t \models \Box\phi$.
  - $S, t \models \Box\Diamond\phi$ iff every time point t' satisfies $S, t' \models \Diamond\phi$. For this to be the case, some t'' in S has to fulfill $S, t'' \models \phi$. This can now be rewritten in the formula $\forall t' \exists t'' : S, t'' \models \phi$, which is equal to $\exists t'' : S, t'' \models \phi$ and can be written as $S, t \models \Diamond\phi$.
  - $S, t \models \Box @_{\bar{t}}\phi$ iff every time point t' satisfies $S, t' \models @_{\bar{t}}\phi$ for some time point $\bar{t}$. This is then true iff $S, \bar{t} \models \phi$ and results in the satisfaction requirement $\forall t' : S, \bar{t} \models \phi$. This is then equal to $S, \bar{t} \models \phi$, which is checked in the current time as $S, t \models @_{\bar{t}}\phi$.

  Since the other possibilities work in the exact same way, we will shorten the proof here a bit and only have a look at the semantics of the finally and at operator:

  - $S, t \models \Diamond \odot \phi$ iff some t' in S satisfies $S, t' \models \odot\phi$. No matter what the temporal operator is, if $\odot\phi$ is fulfilled at some, all or a specific time t'', it will also be fulfilled at t' as we can just set $t' = t''$.
  - $S, t \models @_{\bar{t}} \odot \phi$ iff $S, \bar{t} \models \odot\phi$. Note that $\bar{t}$ can be outside the current substream, which means that in this case the formula evaluates to $\bot$. If the time point is in the current stream, the subformula $\odot\phi$ will be evaluated on the current substream, as the temporal operator is not bounded by the time point $\bar{t}$ from the @ operator.

Now that the base case is established, we can go forth inductively by increasing the number of consecutive operators to m. We take the first two and apply the base case, which means that the first operator gets cancelled, reducing the number of operators left to $m-1$. By doing this one step at a time we end up with only a single operator left, which is the innermost operator from the original stack.

- The next rule uses the combination of window function and reset operator $\boxplus^n \triangleright \phi$. This proof is again quite simple. The reset operator changes the current substream to the original data stream. This means that it simply removes all stream constraints imposed by the previous windows. Therefore any window function directly in front of a reset operator serves no purpose, as the substream immediately gets reset to the original stream.

- Additionally, if there are multiple adjacent reset operators, as the stream gets reset by the first for the subformula, all can be reduced to one single reset operator. Of course, any leading reset operator can be omitted as there is no point in resetting the data stream to itself.

- Next we have the rule concerning $\boxplus^n \alpha$, where α is an ordinary formula, i.e. there is no temporal operator or window function occurring in α. Since α can only be evaluated in the current time, as there is no temporal meaning imposed on it by a temporal operator, the window serves no meaning and can be omitted.

- The last rule is the window introduction rule, which is used to find a window for the temporal operators that are in the scope of one, but not directly adjacent. We establish the base case by stating

$$\boxplus^n \odot \phi \mathrel{\hat{=}} \boxplus^n \triangleright \boxplus^n \odot \phi$$

Now assume there is a formula $\boxplus^n \odot \phi$ where no rule of the above can be applied. We further assume that there is a subformula ψ in ϕ, that is of the form $\odot\chi$, where the temporal operator is not adjacent to a window function. Since we know that it is still in the scope of one, we have to evaluate it in this substream. To achieve this we employ the transformation above, which allows us to create a new window function without changing the semantics of the formula. The resulting subformula ψ then looks like $\triangleright \boxplus^n \odot\chi$. For the induction step we nest the subformula ψ further into a structure with many window functions and temporal operators. Assume a nesting depth of δ. We now can not apply the above transformation as the stream would be reset to the original stream, but in reality we only need to emulate the latest window function. To combat this, we limit the window to the interval this temporal operator would occur in. Therefore we have to use all previous windows and stack them together in consecutive order like so

$$\boxplus^{n_1} \odot \ldots \boxplus^{n_\delta} \odot \ldots \odot \chi \mathrel{\hat{=}} \boxplus^{n_1} \odot \ldots \boxplus^{n_m} \odot \ldots \triangleright \boxplus^{n_1} \ldots \boxplus^{n_\delta} \odot \chi$$

By the rule for consecutive window occurrences we can then simplify this by the following rule:

$$\boxplus^{n_1} \odot \ldots \boxplus^{n_\delta} \odot \ldots \odot \chi \mathrel{\hat{=}} \boxplus^{n_1} \odot \ldots \boxplus^{n_m} \odot \ldots \triangleright \boxplus^{min(n_1,\ldots n_\delta)} \odot \chi$$

This results in the definition of the rule as given above.

These new rules make the NWNF complete in that every temporal can be guarded by a window operator, making the resulting translation far easier. □

4.3.3 Translating MTL$_P$ to LARS

Now we can begin with the actual translation. The biggest difference between window operators in LARS and intervals in MTL is that windows truncate the stream and therefore all nested windows, while in MTL temporal operators are not limited by the previous layer of operators. This makes the translation from MTL operators to LARS very easy:

Proposition 3 ***MTL$_P$ Operator to LARS Window Translation.*** *MTL$_P$ operators can be translated to LARS formulas with the following rules:*

$$\begin{array}{lcl} MTL_P & & LARS \\ \diamondminus_{[0,n]}\ \phi & \hat{=} & \triangleright \boxplus^n \Diamond \phi \\ \boxminus_{[0,n]}\ \phi & \hat{=} & \triangleright \boxplus^n \Box \phi \end{array} \tag{4.7}$$

This can be employed in any size of nested temporal operators, however the leading reset operator may be omitted.

Proof. We inductively prove the translation on both operators:

- First we have a look at the translation for the diamond or finally operator.

$$\begin{array}{lll} S,t \models \diamondminus_{[0,n]}\ \phi & iff & \exists t' : S,t' \models \phi \text{ and } t' \in [t-n,t] \\ S,t \models \boxplus^n \Diamond \phi & iff & \exists t' : S',t' \models \phi \text{ and } S' = \{\nu(t'') \,|\, t-n \leq t'' \leq t\} \end{array}$$

  As we can see, the effective interval for the operator is the same, the two notions just operate with different tools, either cutting the stream and evaluating the formula on all of it, or simply limiting the stream to a certain interval in which the formula is evaluated. Now assume that there are some temporal operators occurring before such a MTL formula. Other than in LARS, the stream is not cut, but rather just evaluated by the interval in each time point specified. This means that in LARS we have to reset the substream to the original stream in every instance of the temporal operator to preserve the semantics at any nesting depth δ.

$$\odot_{[0,n_1]} \dots \diamondminus_{[0,n_\delta]}\ \phi \mathrel{\hat{=}} \boxplus^{n_1} \odot \dots \triangleright \boxplus^{n_\delta} \Diamond \phi$$

As the temporal operators must evaluate on the original stream, we can generalize this to

$$\diamondminus_{[0,n]}\ \phi \mathrel{\hat{=}} \triangleright \boxplus^{n} \Diamond \phi$$

making the translation valid.

- By the same arguments we can also prove that

$$\boxminus_{[0,n]} \phi \mathrel{\hat{=}} \triangleright \boxplus^{n} \Box \phi$$

holds. □

Example 10 *The MTL formula $\boxminus_{[n,0]} \boxminus_{[m,0]} (p \wedge \diamondminus_{[i,0]}\ q)$ can be translated to the LARS formula $\boxplus^n \Box \triangleright \boxplus^m \Box (p \wedge \triangleright \boxplus^i \diamond q)$.*

We start with the outermost operator $\boxminus_{[0,n]} \phi$, which is then turned to $\triangleright \boxplus^n \Box \phi$. In the same fashion, we translate the next subformula $\boxminus_{[0,m]} \phi$ to $\triangleright \boxplus^m \Box \phi$. The proposition p as well as the logical and '$\wedge$' do not need to be changed and can be taken as is. The last operator that needs translation is $\diamondminus_{[0,i]}\ q$, which is rewritten to $\triangleright \boxplus^i \diamond q$. Put together in the original sequence we obtain $\triangleright \boxplus^n \Box \triangleright \boxplus^m \Box (p \wedge \triangleright \boxplus^i \diamond q)$. The leading reset operator is omitted and we obtain the above formula.

4.3.4 Translating NWNF to MTL$_P$

On the other hand, the translation of window operators to MTL is more difficult. The intuition behind it is, that every nested window creates a truncated version of the temporal operator at each time step. To model this in MTL, it is necessary to split each time point apart from the computation and create its own version of the interval. For a single nested window operator this looks as follows:

$$\begin{array}{ccc} LARS & & MTL_P \\ \boxplus^n \Box \boxplus^m \odot \phi & \mathrel{\hat{=}} & \bigwedge\limits_{i \in [0,n]} \boxminus_{[i,i]} \odot_{[0,min(m,n-i)]}\ \phi \\ \boxplus^n \Diamond \boxplus^m \odot \phi & \mathrel{\hat{=}} & \bigvee\limits_{i \in [0,n]} \boxminus_{[i,i]} \odot_{[0,min(m,n-i)]}\ \phi \end{array} \tag{4.8}$$

For any nesting depth greater than one we first propose a new notation to deal with nested window operations:

Notation 2 ***Subformula Window Intervals.*** *A LARS formula $\phi = \boxplus^n \psi$ with any subformula ψ will further be denoted as ϕ^n, where the superscript defines the type and boundary of the outermost window function. If there are multiple such window function like $\phi = \boxplus^n \psi \vee \boxplus^m \chi$, they can be superscripted as a list of such intervals $\phi^{n,m}$.*
An MTL formula $\phi = \odot_{[i,j]} \psi$ similarly receives a subscript with the outermost temporal interval like $\phi_{[i,j]}$. This can again be done in the form of a list if there are multiple operators in the same nesting depth.

Proposition 4 ***Translation of Formulas with arbitrary Nesting Depth.*** *We propose the following more general rules, utilizing the above notation:*

$$\begin{array}{lcl} LARS & & MTL_P \\ \boxplus^n \, \Box \phi^m & \hat{=} & \bigwedge_{i \in [0,n]} \boxminus_{[i,i]} \phi_{[0, min(m, n-i)]} \\ \boxplus^n \, \Diamond \phi^m & \hat{=} & \bigvee_{i \in [0,n]} \boxminus_{[i,i]} \phi_{[0, min(m, n-i)]} \end{array} \tag{4.9}$$

This translation can be used recursively starting from the outermost window operator and continuing until only one window operator is left that can be translated as a regular MTL operator. The second window operator may also be nested inside a subformula, in which case the entire subformula replaces $\odot \phi$ and the next window gets the new interval assigned.

Proof. We start with the base case for the always operator as defined above:

$$\boxplus^n \Box \, \boxplus^m \odot \phi \quad \hat{=} \quad \bigwedge_{i \in [0,n]} \boxminus_{[i,i]} \odot_{[0, min(m, n-i)]} \phi$$

The left hand side is true iff every time point in the substream created by the window function satisfies the subformula $\odot \phi$ in the subsubstream created by the second window function. This means that each time point in the substream has to check whether this subformula is true in the subsubstream. The first window function therefore has to be eliminated by pulling it apart into the time points of the substream. The semantic equivalence can be seen here:

$$\begin{array}{lll} S, t \models \boxplus^n \Box \phi & \textit{iff} & \forall t' : S', t' \models \phi \\ S, t \models \boxplus^n \Box \phi & \textit{iff} & \bigwedge_{t' \in [t-n, t]} S, t' \models \phi \\ S, t \models \boxplus^n \Box \phi & \textit{iff} & S, t \models \bigwedge_{t' \in [0,n]} @_{t-t'} \, \phi \end{array}$$

The above formula can then be directly translated to MTL as

$$S,t \models \bigwedge_{t' \in [0,n]} @_{t-t'}\, \phi \quad \hat{=} \quad S,t \models \bigwedge_{i \in [0,n]} \boxminus_{[i,i]} \phi$$

Because of the truncating nature of window functions, the latest time point in the substream can create only a single time point in the subsubstream. This is modelled by assigning each time point in the substream its own subformula checking only the truncated time interval. The truncation happens in the MTL operator interval starting with 0, since we can only observe past time windows currently. The function $min(m, n-i)$ then truncates the interval at either the point where the inner window would end the stream, or where the outer window would truncate the inner stream. This means that with an $i = n$ only the latest time point will be seen, which fulfills the semantics of nested windows. Similarly, the finally operator base case

$$\boxplus^{n} \Diamond \boxplus^{m} \odot \phi \quad \hat{=} \quad \bigvee_{i \in [0,n]} \boxminus_{[i,i]} \odot_{[0,min(m,n-i)]}\, \phi$$

can be explained, except that there only needs to be one point in time fulfilling the subformula, which makes it a disjunction instead.

From this base case we add additional window functions to ϕ of any nesting depth δ. We go inductively starting with the outermost window function to the innermost, until we reach the base case. Assume a formula

$$\boxplus^{n_0} \Box \phi^{n_1}$$

As we have defined above, the subformula ϕ includes a window function with the superscript m, defining the subsubstream. Using the formula from above we can pull the window apart to emulate the substream generation:

$$\boxplus^{n_0} \Box \phi^{n_1} \quad \tilde{\hat{=}} \quad \bigwedge_{i \in [0,n_0]} \boxminus_{[i,i]} \phi^{n_1}$$

Next the subformula ϕ^{n_1} has to be adapted to fit the truncating nature of substreams. We adjust the next outermost window of ϕ^{n_1} to define the window $min(n_1, n_0 - i)$ so that the interval models the substream.

$$\boxplus^{n_0} \Box \phi^{n_1} \quad \hat{=} \quad \bigwedge_{i \in [0,n_0]} \boxminus_{[i,i]} \phi^{min(n_1,n_0-i)}$$

In the next nesting level of $\phi^{min(n_1,n_0-i)} = \boxplus^{min(n_1,n_0-i)} \psi^{n_2}$, we again employ the strategy of pulling apart the window function to create a conjunction or disjunction of time points. This can be continued until $n_{\delta-1}$, where the base case can be used to preserve the last temporal operator without pulling it apart. $\Box$

Example 11 *We look at the NWNF formula* $\boxplus^5 \Diamond \boxplus^2 \Diamond(p \vee q)$.

- *We start with the translation beginning with* $\boxplus^5 \Diamond \phi^2$*, which is in MTL then*

$$\bigvee_{i \in [0,5]} \boxminus_{[i,i]} \phi_{[0,min(2,5-i)]}.$$

- *This is written in full form*

$$\boxminus_{[0,0]} \phi_{[0,2]} \vee \boxminus_{[1,1]} \phi_{[0,2]} \vee \boxminus_{[2,2]} \phi_{[0,2]} \vee \boxminus_{[3,3]} \phi_{[0,2]} \vee \boxminus_{[4,4]} \phi_{[0,1]} \vee \boxminus_{[5,5]} \phi_{[0,0]}$$

- *The subformula*

$$\phi_{[0,min(2,5-i)]} \equiv \boxplus^{min(2,5-i)} \Diamond(p \vee q)$$

  is then translated with regular MTL equivalence to

$$\diamondminus_{[0,min(2,5-i)]} (p \vee q)$$

  and inserted back into the formula as the MTL logic formula

$$\boxminus_{[0,0]} \diamondminus_{[0,2]} (p \vee q) \vee \boxminus_{[1,1]} \diamondminus_{[0,2]} (p \vee q) \vee \boxminus_{[2,2]} \diamondminus_{[0,2]} (p \vee q) \vee$$

$$\boxminus_{[3,3]} \diamondminus_{[0,2]} (p \vee q) \vee \boxminus_{[4,4]} \diamondminus_{[0,1]} (p \vee q) \vee \boxminus_{[5,5]} \diamondminus_{[0,0]} (p \vee q).$$

As it can be seen, the formula size is increased by the product of interval sizes of all nested windows. This makes it an exponential blowup roughly of size $O(\iota^\delta)$, where ι is the size of the biggest window interval and δ is the maximum window nesting depth.

4.4 Custom Window Functions

In the previous sections, we were only able to utilize past time time-based window functions. However this may be too limiting for the rich expressivity of LARS. For this we now add the notion of general window functions and intervals in LARS and Metric LARS. Further we have a look at what is currently not possible to express in regular MTL$_P$.

4.4.1 Future Time Window Functions

First we want to extend the time-based window functions to include future time intervals. We therefore change the formulas to consider from $\boxplus^n \odot \phi$ to $\boxplus^{[i,j]} \odot \phi$, where the resulting substream clips the original stream to the interval $[t-i, t+j]$, relative to some time t where the window is evaluated.

We start by adding the notion of future time window functions. These window functions define an interval starting at 0, i.e. the current time $\hat{t}$, and end at some point $\hat{t}+j$. Therefore the currently available windows look as follows: $\boxplus^{[0,j]}$ for future time windows and $\boxplus^{[i,0]}$ for past time intervals. The translation for these strictly future and past window functions is then easy:

$$
\begin{array}{lcl}
LARS & & MTL_P \\
\triangleright \boxplus^{[i,0]} \Box \phi & \hat{=} & \boxminus_{[0,i]}\ \phi \\
\triangleright \boxplus^{[0,j]} \Box \phi & \hat{=} & \boxplus_{[0,j]}\ \phi \\
\triangleright \boxplus^{[i,0]} \Diamond \phi & \hat{=} & \diamondminus_{[0,i]}\ \phi \\
\triangleright \boxplus^{[0,j]} \Diamond \phi & \hat{=} & \diamondplus_{[0,j]}\ \phi
\end{array}
\tag{4.10}
$$

The leading reset operator signifies that there may not be any leading windows for this translation to work. For nested window functions we have to rewrite the translation rules from before here to make sure all the different combinations of past and future intervals are dealt with:

$$
\begin{array}{lcl}
\mathit{LARS} & & \mathit{MTL}_P \\
\boxplus^{[0,j]}\,\Box\phi^{[0,n]} & \hat{=} & \bigwedge_{k\in[0,j]} \boxplus_{[k,k]}\phi^{[0,min(n,i-k)]} \\
\boxplus^{[0,j]}\,\Box\phi^{[n,0]} & \hat{=} & \bigwedge_{k\in[0,j]} \boxplus_{[k,k]}\phi^{[min(n,k),0]} \\
\boxplus^{[i,0]}\,\Box\phi^{[0,n]} & \hat{=} & \bigwedge_{k\in[0,i]} \boxminus_{[k,k]}\phi^{[0,min(n,k)]} \\
\boxplus^{[i,0]}\,\Box\phi^{[n,0]} & \hat{=} & \bigwedge_{k\in[0,i]} \boxminus_{[k,k]}\phi^{[min(n,i-k),0]} \\
\boxplus^{[0,j]}\,\Diamond\phi^{[0,n]} & \hat{=} & \bigvee_{k\in[0,j]} \boxplus_{[k,k]}\phi^{[0,min(n,i-k)]} \\
\boxplus^{[0,j]}\,\Diamond\phi^{[n,0]} & \hat{=} & \bigvee_{k\in[0,j]} \boxplus_{[k,k]}\phi^{[min(n,k),0]} \\
\boxplus^{[i,0]}\,\Diamond\phi^{[0,n]} & \hat{=} & \bigvee_{k\in[0,i]} \boxminus_{[k,k]}\phi^{[0,min(n,k)]} \\
\boxplus^{[i,0]}\,\Diamond\phi^{[n,0]} & \hat{=} & \bigvee_{k\in[0,i]} \boxminus_{[k,k]}\phi^{[min(n,i-k),0]}
\end{array}
\tag{4.11}
$$

Next we will have a look at the mixed intervals in the base case, so without any nested window functions:

$$
\begin{array}{lcl}
\mathit{LARS} & & \mathit{MTL}_P \\
\boxplus^{[i,j]}\,\Box\phi & \hat{=} & \boxminus_{[i,0]}\,\phi \wedge \boxplus_{[0,j]}\phi \\
\boxplus^{[i,j]}\,\Diamond\phi & \hat{=} & \diamondminus_{[i,0]}\,\phi \wedge \diamondplus_{[0,j]}\,\phi
\end{array}
\tag{4.12}
$$

These formulations make it now easy to see a common scheme far a general translation of any form of interval window function.

Proposition 5 ***Mixed Time Window Translation.*** *We propose the following rules to translate any LARS formula with window functions of future- and past-time intervals* *to MTL_P:*

$$
\begin{array}{lcl}
LARS & & MTL_P \\
\boxplus^{[i,j]}\,\Box\phi^{[n,m]} & \hat{=} & \bigwedge\limits_{k\in[0,i]} \boxminus_{[k,k]}\phi^{[min(n,i-k),min(m,j+k)]}\wedge \\
 & & \bigwedge\limits_{l\in[0,j]} \boxplus_{[l,l]}\phi^{[min(n,i+l),min(m,j-l)]} \\
\boxplus^{[i,j]}\,\Diamond\phi^{[n,m]} & \hat{=} & \bigvee\limits_{k\in[0,i]} \diamondminus_{[k,k]}\,\phi^{[min(n,i-k),min(m,j+k)]}\vee \\
 & & \bigvee\limits_{l\in[0,j]} \diamondplus_{[l,l]}\,\phi^{[min(n,i+l),min(m,j-l)]}
\end{array}
\tag{4.13}
$$

The translations mentioned in Equations 4.11 and 4.12 are then special cases of this general translation and do not need their own proof.

Proof. We start by showing that the general window interval can be split into past and future without loosing semantic accuracy in the nesting depth 0 case.

$$
\begin{array}{lcl}
S,t \models \boxplus^{[i,j]}\Box\phi & \textit{iff} & \forall t' \in [t-i,t+j] : S,t' \models \phi \\
S,t \models \boxplus^{[i,j]}\Box\phi & \textit{iff} & S,t \models \bigwedge\limits_{t'\in[t-i,t+j]} @_{t'}\phi
\end{array}
$$

This can then be split into past and future parts:

$$
S,t \models \bigwedge\limits_{t'\in[t-i,t+j]} @_{t'}\phi \quad \textit{iff} \quad S,t \models \bigwedge\limits_{t_P\in[t-i,t]} @_{t_P}\phi \wedge \bigwedge\limits_{t_F\in[t,t+j]} @_{t_F}\phi
$$

In the same way we obtain the translation for the finally operator:

$$
S,t \models \boxplus^{[i,j]}\Diamond\phi \quad \textit{iff} \quad S,t \models \bigvee\limits_{t_P\in[t-i,t]} @_{t_P}\phi \vee \bigvee\limits_{t_F\in[t,t+j]} @_{t_F}\phi
$$

These resulting formulas can be directly translated to MTL in the way we have seen in the base case that is Equation 4.12. Next we assume there are multiple window functions nested in ϕ up to a nesting depth δ.

$$
\begin{array}{lcl}
S,t \models \boxplus^{[i,j]}\Box\phi^{[n,m]} & \textit{iff} & S,t \models \bigwedge\limits_{t'\in[-i,j]} @_{t+t'}\phi^{I} \\
S,t \models \bigwedge\limits_{t'\in[-i,j]} @_{t+t'}\phi^{I} & \textit{iff} & S,t \models \bigwedge\limits_{t_P\in[0,i]} @_{t-t_P}\phi^{I_P} \wedge \bigwedge\limits_{t_F\in[0,j]} @_{t+t_F}\phi^{I_F}
\end{array}
$$

where I, I_P and I_F have to be determined. We start with I, which determines the interval for the inner window function in the non-split case. This will be only truncated at the border of the outer window, so we have to check for the minimum of either its own border or the sliding effect of closing in. This results in the new interval $I = [min(n, i - t'), min(m, j - t')]$.

$$S, t \models \boxplus^{[i,j]} \Box \phi^{[n,m]} \qquad \textit{iff} \qquad S, t \models \bigwedge_{t' \in [-i,j]} @_{t+t'} \phi^{[min(n,i-t'),min(m,j-t')]}$$

The last two intervals are trickier because the truncating outer border is now split in two parts. This means that we have to allow the substream to go beyond the scope of the counting variable instead of stopping at 0, like in the previous cases. This propagates the new window intervals down a nesting layer that has again a future and past part in the most cases. For the past, we have to use the previous trick of truncating at the border, as well as truncating in the future if the inner window interval is bigger than the outer one. This is done with the past time interval $I_P = [min(n, i - t_P), min(m, j + t_P)]$ and the future time interval $I_F = [min(n, i + t_F), min(m, j - t_P)]$. The reason for their shape is that in the case of I_P that the counter decreases the current time by its current value. The interval for the past is then either as defined or is truncated by the border. The border is closer in each time step further down. This means we need to compare the defined bound with the approached bound. For the future part, it is the other way around, so the bigger the counter gets, the further we leave the future bound, so it has to be gradually increased until we can use the defined bound. For I_F this is the same, but the other way around. With this we result in the final equation:

$$S, t \models \bigwedge_{t_P \in [0,i]} @_{t-t_P} \phi^{[min(n,i-t_P),min(m,j+t_P)]} \wedge \bigwedge_{t_F \in [0,j]} @_{t+t_F} \phi^{[min(n,i+t_F),min(m,j-t_P)]}$$

Translated into MTL by the semantics for the @ operator we receive the exact equation as proposed in Equation 4.13. $\Box$

Example 12 *Consider the formula* $\boxplus^{[2,2]} \Diamond \boxplus^{[1,2]} \Box p$*.*

We start with the first window function $\boxplus^{[2,2]} \Diamond \phi^{[1,2]}$*. In this case we have a future and past time window interval to deal with, so we use the translation formula from Equation 4.13. This yields the preliminary formula:*

$$\bigvee_{t_P \in [0,2]} \diamondminus_{[t_P,t_P]} \phi^{[min(1,2-t_P),min(2,2+t_P)]} \vee \bigvee_{t_F \in [0,2]} \diamondplus_{[t_F,t_F]} \phi^{[min(1,2+t_F),min(2,2-t_F)]}.$$

Spelled out, this results in the following formula:

$$\diamondminus_{[0,0]} \phi^{[1,2]} \vee \diamondminus_{[1,1]} \phi^{[1,1]} \vee \diamondminus_{[2,2]} \phi^{[0,2]} \vee \diamondplus_{[0,0]} \phi^{[1,2]} \vee \diamondplus_{[1,1]} \phi^{[1,1]} \vee \diamondplus_{[2,2]} \phi^{[1,0]}.$$

For each subformula ϕ we now have to adjust the outermost window function. Since there are no more nested windows, we can use the base case and directly translate to MTL as seen in Equation 4.12:

$$\diamondminus_{[0,0]} (\boxminus^{[0,1]} p \wedge \boxplus^{[0,2]} p) \vee \diamondminus_{[1,1]} (\boxminus^{[0,1]} p \wedge \boxplus^{[0,1]} p) \vee \diamondminus_{[2,2]} (\boxminus^{[0,0]} p \wedge \boxplus^{[0,2]} p) \vee$$

$$\diamondplus_{[0,0]} (\boxminus^{[0,1]} p \wedge \boxplus^{[0,2]} p) \vee \diamondplus_{[1,1]} (\boxminus^{[0,1]} p \wedge \boxplus^{[0,1]} p) \vee \diamondplus_{[2,2]} (\boxminus^{[0,1]} p \wedge \boxplus^{[0,0]} p).$$

The final result can be optimized a bit by removing redundant subformulas:

$$\begin{aligned}
\boxplus^{[2,2]} \Diamond \boxplus^{[1,2]} \Box p \quad \hat{=} \quad & \diamondminus_{[0,0]} (\boxminus^{[0,1]} p \wedge \boxplus^{[0,2]} p) \vee \\
& \diamondminus_{[1,1]} (\boxminus^{[0,1]} p \wedge \boxplus^{[0,1]} p) \vee \\
& \diamondminus_{[2,2]} \boxplus^{[0,2]} p \vee \\
& \diamondplus_{[0,0]} (\boxminus^{[0,1]} p \wedge \boxplus^{[0,2]} p) \vee \\
& \diamondplus_{[1,1]} (\boxminus^{[0,1]} p \wedge \boxplus^{[0,1]} p) \vee \\
& \diamondplus_{[2,2]} \boxminus^{[0,1]} p
\end{aligned}$$

This example shows how much small LARS formulas may blow up when translated to MTL. We will have a closer look at this blow up in Lemma 2 on page 59.

4.4.2 General MTL$_P$ Interval Operators

So far we only looked at general intervals for LARS window functions, however also MTL$_P$ has a more extended notion of intervals subscripted to its operators. Other than the window functions, the temporal operators of MTL need not contain the current point in time. This means that the interval $[i, j]$ subscripted under a past time operator defines the interval $[\hat{t} - j, \hat{t} - i]$, where $\hat{t}$ is the current time.

Proposition 6 ***General MTL$_P$ Interval Operator Translation.*** *Since MTL does not truncate the stream, we do not need to worry about nested window operations in these formulas and we can simply propose the following translation:*

$$\begin{aligned}
\diamondminus_{[i,j]} \phi \quad & \hat{=} \quad \triangleright @_{\hat{t}-i} \boxplus^{[j-i,0]} \Diamond \phi \\
\diamondplus_{[i,j]} \phi \quad & \hat{=} \quad \triangleright @_{\hat{t}+i} \boxplus^{[0,j-i]} \Diamond \phi \\
\boxminus_{[i,j]} \phi \quad & \hat{=} \quad \triangleright @_{\hat{t}-i} \boxplus^{[j-i,0]} \Box \phi \\
\boxplus_{[i,j]} \phi \quad & \hat{=} \quad \triangleright @_{\hat{t}+i} \boxplus^{[0,j-i]} \Box \phi
\end{aligned}$$

This is a direct generalization of the translation given in the previous section of this thesis.

Proof. We start by demonstrating how a general temporal operator in MTL can be rewritten as two strict ones:

$$\diamondminus_{[i,j]}\ \phi \quad \hat{=} \quad \boxminus_{[i,i]}\ \diamondminus_{[0,j-i]}\ \phi$$

The formula $\diamondminus_{[i,j]}\ \phi$ is true, iff there exists a time point in the interval $[\hat{t}-j, \hat{t}-i]$ where ϕ is fulfilled. By first freezing the time to $\hat{t}-i$ and then applying the interval operator $\diamondminus$, we can emulate the exact behavior of the original formula. This can now be translated to LARS directly:

$$\boxminus_{[i,i]}\ \diamondminus_{[0,j-i]}\ \phi \quad \hat{=} \quad \triangleright\, @_{\hat{t}-i}\, \boxplus^{[0,j-i]}\, \Diamond\phi$$

The reset operator has to be added since the boundaries of the previous window functions do not concern us in MTL. Again, a leading reset may be omitted, as the original stream will always be reset to itself. The proof for the always operator and future time interval works in the same way. □

4.4.3 General Monotonous Window Functions

Metric temporal logic only considers sliding intervals for its operators, but LARS is quite an open system when it comes to the definition of window functions. This means that only a certain number of window functions can be emulated by Metric LARS. In particular we will have a look at a more general approach to make monotonous interval-based window functions like tumbling windows and others possible. Other windows like Filter or Tuple-based window functions will not be covered as they can not be emulated by regular MTL$_P$, since they go beyond the notion of intervals constraining the data stream.

Definition 8 ***Oracle Intervals.*** *An* Oracle Interval *is a predicate of the form*

$$\Omega(f, A_0, ..., A_n, S, E). \tag{4.14}$$

where

- f *is the identifier of the window function*
- $A_0, ..., A_n$ *are arguments of the window function* f
- S *is the starting point of an interval for a temporal operator*
- E *is end point of an interval for a temporal operator*

This predicate accepts any numbers $A_i, S, E \in \mathbb{N}$ and evaluates only then to true, if the start and end point match the computation of the interval of the specified temporal operator with the given arguments. This means it emulates the evaluation of the function $f : \mathbb{N}^n \to \mathbb{N}^2$.

Example 13 *To demonstrate how such a window can be translated, consider the following formula $\boxplus^{t[0,5]}\Diamond\phi$, where the window symbol refers to a tumbling window. This defines a window that splits the stream into parts of 5 time points and evaluates the formula within the current time, i.e. 0, and the next boundary of 5. We can simulate this type of window with the MTL formula*

$$\diamondplus_{[S,E]}\ \phi \wedge \Omega(tumble, \hat{t}, S, E).$$

The function argument $\hat{t}$ refers to the current point in time and the word tumble identifies the function we have defined as tumble : $T \to 0 \times \mathbb{N}$. In detail we calculate the tuple $(0,\ 5 - \hat{t}\ mod\ 5)$.

This oracle is introduced to emulate more complex intervals in the Metric LARS formulas. As is the case in regular LARS we will not allow too complex window functions and restrict them to polynomial time complexity so that $f \in P$. This makes it now possible to allow tumbling time-based windows, partition-based windows etc. to be translated. In a Metric LARS formula, the oracle just has to be conjuncted and can use the variables S and E to define the interval borders of any temporal operator.

4.5 Answer State Semantics

With the translation of general LARS formulas established, we will now have a look at the answer set semantics of Metric LARS. When thinking back about the definition of streams, LARS and MTL in their intention serve different goals and therefore use a different notion of data streams. LARS assumes that there is a state at each time point, while MTL treats a timeline just as a series of states with unknown amounts of time in between where there are no states. To compare the semantics of both systems, we hereby propose two different ways to translate state sequences into LARS streams. After that we have a look at LARS answer streams and then derive Metric LARS answer streams from them.

4.5.1 Continuous Stream Assumption

MTL tries to model systems that rarely receive updates of their current status. For this reason, the temporal operators use intervals to relax the need of having to remember exact time points. The next operator is the best example, since it is defined with an interval to make sure the next state occurs in a given number of steps. MTL can easily adopt the notion of LARS data streams by adding the formula $\boxplus_T \oplus_{[1,1]} \top$. This formula ensures that there is always a state in the next moment of time.

Assume now that we only receive updates when the system state changes. It may still be important to compare the new state to the last in case the difference creates the meaning of the data, which is not uncommon for systems. To model this consider the following formula to transform sparse state sequences to continuous LARS streams using the *Continuous Stream Assumption (CSA(S))*:

Definition 9 ***Continuous Stream Assumption****.* *Given the state sequence* $S = \{\langle \sigma_i, i\rangle \ \mid 0 \leq i < n, n \in \mathbb{N}\}$ *and function* $\tau : \mathbb{N} \mapsto T$*, mapping each state to the system time, then the Continuous State Assumption of* S *is the resulting LARS stream:*

$$CSA(S) = \{\, (\sigma_i, j) \mid \tau(i) \leq j < \tau(i+1) \wedge \langle \sigma_i, i\rangle \in S \,\}. \tag{4.15}$$

4.5.2 Sparse Stream Assumption

On the other hand we also need a method to create sparse LARS streams from state sequences, that does not allow consecutive states to be the same. For this matter we use the *Sparse Stream Assumption (SSA(S))*, that behaves like the differential to the given sequence S.

Definition 10 ***Sparse Stream Assumption.*** *Given a state sequence $S = \{\langle \sigma_i, i\rangle \ \mid 0 \leq i < n, n \in \mathbb{N}\}$ and function $\tau : \mathbb{N} \mapsto T$, mapping each state in the sequence to the system time, the Sparse Stream Assumption of S is the resulting LARS stream:*

$$SSA(S) = \begin{cases} (\sigma_i, \tau(i)) & \forall i :\ 0 \leq i < n \\ (\emptyset, j) & \forall j :\ j \in T \backslash \{\tau(i) \,|\, i \in \mathbb{N}\}. \end{cases} \tag{4.16}$$

4.5.3 LARS Stream Sequence

Of course we not only want to transform state sequences into LARS streams but also the other way around. For this we propose the *LARS Stream Sequence* transformation:

Definition 11 ***LARS Stream Sequence.*** *Given a LARS stream $S = (T, \nu)$, where T is a time domain and ν a function, mapping atoms to time points in T, then the LARS Stream Sequence of S is defined as the following timed state sequence:*

$$LSS(S) = \{\langle \sigma_i, i\rangle \mid \sigma_i = \nu(t_i) \wedge \sigma_i \neq \emptyset\}. \tag{4.17}$$

4.5.4 Report Only Changes State Sequence

Typically MTL state sequences only report the changes in a system. This means that it should not occur in two directly following states, that they bear the same information. For this type of LARS stream translation, we propose the *Report Only Changes State Sequence*, or short ROCS:

Definition 12 ***Report Only Changes State Sequence.*** *Given a LARS stream $S = (T, \nu)$, where T is a time domain and ν a function, mapping atoms to time points in T, then the Report Only Changes State Sequence of S is defined as:*

$$ROCS(S) = \{\langle \sigma_i, i\rangle \mid \sigma_i = \nu(t_i) \backslash \nu(t_{i-1}) \wedge \sigma_i \neq \emptyset\}. \tag{4.18}$$

4.5.5 Comparing Timed State Sequences

It is important for the definition of answer sets to compare two different timed state sequences. For this reason we introduce the subset operator $\subseteq$ for the semantic structures.

Definition 13 ***Timed State Sequence Subsets.*** *Consider two timed state sequences $I = \{\langle \bar{p}_i, i \rangle, \tau_I \mid i \leq n\}$ and $J = \{\langle \bar{q}_j, j \rangle, \tau_J \mid j \leq m\}$. The sequence I is considered to be a subset of J, written $I \subseteq J$, if the following conditions hold:*

1. $\forall\, i, j : \tau_I(i) = \tau_J(j) \rightarrow \bar{p}_i \subseteq \bar{q}_j$.
2. $\forall\, i \leq n\, \exists\, j \leq m : \tau_I(i) = \tau_J(j)$.

This means that every specified time point in I needs to have a corresponding time point in I, of which the set of propositions of J is also a subset of the propositions of J.

Example 14 *Consider the timed state sequences I:*

$$\begin{aligned} t = 0 : &\quad \sigma_1 = \{b, c\} \\ t = 2 : &\quad \sigma_2 = \{a\} \\ t = 3 : &\quad \sigma_3 = \{a, d\} \\ t = 5 : &\quad \sigma_4 = \{a, c\} \end{aligned}$$

and J:

$$\begin{aligned} t = 0 : &\quad \sigma_1 = \{a, b, c\} \\ t = 1 : &\quad \sigma_2 = \{a\} \\ t = 2 : &\quad \sigma_3 = \{a, b, c, d\} \\ t = 3 : &\quad \sigma_4 = \{a, d\} \\ t = 4 : &\quad \sigma_5 = \{a, b, c\} \\ t = 5 : &\quad \sigma_6 = \{a, c, d\} \end{aligned}$$

over the same time domain T and set of propositions $P = \{a, b, c, d\}$. We now want to see if the relation $I \subseteq J$ holds. We start with the first constraint. Each state of I with the same time mapping as a state within J has to be a propositional subset of J. In this case we need to look at the states at time points 0, 2, 3 and 5. As we can see, all the states of I with a corresponding state in J fulfill this. In the next constraint, we have to check if each state in I has a corresponding state in J. As we have already seen in the previous condition, this is true for the state sequences.

This then leads to a very important property of timed state sequences. Together with the subset operator, a set of timed state sequences is a partial order:

Proposition 7 ***Partial Order of Timed State Sequences.*** *Let $\mathfrak{S} = \{s_0, s_1, ...\}$ denote the set of timed state sequences over a set time domain T and a set of propositions P. Further, let $\subseteq$ be the subset operator. This operator is then a partial ordering on $\mathfrak{S}$ and $(\mathfrak{S}, \subseteq)$ is a partially ordered set.*

Proof. Recall that for a set S, for a binary relation $R \subseteq S^2$ to be a partial ordering of S, R has to fulfill the three properties of Reflexivity (aRa for each $a \in S$), Transitivity (if aRb and bRc then also aRc), and Antisymmetry (if aRb and bRa then $a = b$).

- Reflexivity: Let a be a timed state sequence. A reflexive operator satisfies $a \subseteq a$ for every a. The first condition as in Definition 13 states that for each time point that is in the same real time point for both sequences, the propositions have to be equal or a subset. This is always the case as the propositions are equal in all equal timed states. The second condition states that all timed states of the left sequence have to be included in the right sequence. As both are equal, this is always the case.
- Transitivity: Let a, b and c be timed state sequences. To satisfy transitivity, we have to show that if $a \subseteq b$ and $b \subseteq c$ then $a \subseteq c$ is always the case. Assume the left hand side is given, so that $a \subseteq b$ and $b \subseteq c$. This means that b includes all the information of a and c includes every information provided in b. As there is no loss of information between these relations, c has to include all the information of a, which means that
  - In each timed state of a the propositions of c are at least equal, and
  - each timed state in a is included in c.
- Antisymmetry: Let a and b denote timed state sequences. For the relation to be called antisymmetric, it has satisfy that for all a, b if $a \subseteq b$ and $b \subseteq a$ then $a = b$. Assume $a \subseteq b$ and $b \subseteq a$ holds. This means that each timed state in a is included in b and each timed state in b is included in a. This means that both have the same amount and timing of their states. Further, in each state the propositions of a and b have to be equal as both include each others propositions. Therefore the sequences have to be equal to each other.

Satisfying all these properties makes the subset operator and the set of timed state sequences a partially ordered set. □

4.5.6 Metric LARS Answer State Sequences

As we have seen, there is semantic equivalence for LARS and Metric LARS formulas and we can transform LARS streams into timed state sequences and back. This makes it now easy to define the answer set semantics of Metric LARS programs.

Definition 14 ***Metric LARS Formulas.*** *A Metric LARS Formula is any formula in MTL_P that can be translated to LARS. This leads to the following syntax definition:*

$$\phi ::= p \mid \neg \phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid \boxplus_I \phi \mid \boxminus_I \phi \mid \diamondplus_I \phi \mid \diamondminus_I \phi \tag{4.19}$$

Where p is a proposition and I an interval.

Definition 15 ***Metric LARS Rules.*** *A Metric LARS rule is of the form*

$$\phi_0 \leftarrow \phi_1 \wedge ... \wedge \phi_n. \tag{4.20}$$

where $\phi_0, ..., \phi_n$ are arbitrary Metric LARS formulas. The left hand side is called the rule head – $head(r)$ – and the right hand side is the rule body – $body(r)$. A Metric LARS program consists of a set of Metric LARS rules.

Definition 16 ***Answer State Sequences.*** *Metric LARS is evaluated on a timed state sequence D called the* data state sequence *in a model structure $\mathfrak{M} = \langle I, O, B \rangle$ composed of a timed state sequence I called the* interpretation sequence *with $D \subseteq I$, a possibly empty set of oracle intervals O and B, the background data. We use the FLP-reduct $\mathfrak{P}^{\mathfrak{M},t} = \{r \in \mathfrak{P} \mid \mathfrak{M}, t \models body(r)\}$ at time point t to determine the answer state sequences of a Metric LARS program $\mathfrak{P}$. For I to be an answer state sequence of $\mathfrak{P}$ at time point t, I has to be minimal compared to all other models of the reduct $\mathfrak{P}^{\mathfrak{M},t}$. The set of all answer state sequences of the program $\mathfrak{P}$ on the data sream D at the time point t is denoted as $AS(\mathfrak{P}, D, t)$.*

Example 15 *Assume the data state sequence $D = \{\sigma_1 = \{b, c\}, \sigma_2 = \{c\}\}$ with its corresponding mapping function $\tau_D = \{1 \mapsto 0, 2 \mapsto 3\}$ and the program*

$$\begin{aligned} \mathfrak{P} = \{ \quad & a \leftarrow \diamondminus_{[0,2]}\, b. \\ & d \leftarrow \boxplus_{[1,1]} c. \quad \} \end{aligned}$$

Assume the model structure for time point $t = 2$ is the following:

$$\mathfrak{M} = \langle I = \{\sigma_1 = \{b, c\}, \sigma_2 = \{a, d\}, \sigma_3 = \{c\}\}, \emptyset, \emptyset\} \rangle$$

where I has the corresponding mapping function $\tau_I = \{1 \mapsto 0, 2 \mapsto 2, 3 \mapsto 3\}$. As no oracle intervals are used in this program, the set of oracles is empty.

The reduct $\mathfrak{P}^{\mathfrak{M},2}$ has the form

$$\begin{aligned} \mathfrak{P}^{\mathfrak{M},2} = \{ \quad & a \leftarrow \diamondminus_{[0,2]}\, b. \\ & d \leftarrow \boxplus_{[1,1]} c. \quad \} \end{aligned}$$

Here nothing has changed but compare to the reduct of time point $t = 0$:

$$\mathfrak{P}^{\mathfrak{M},0} = \{ \quad a \leftarrow \diamondminus_{[0,2]} b. \quad \}$$

The interpretation state sequence I is therefore the answer state sequence of $\mathfrak{P}$ for D at $t = 2$, so that $I \in AS(\mathfrak{P}, D, 2)$. The answer states for this program on the given data sequence at the other time points t will look as follows:

$$\begin{aligned} t = 0: &\quad \sigma_{t=0} = \{a, b, c\} \\ t = 1: &\quad \sigma_{t=1} = \{a\} \\ t = 2: &\quad \sigma_{t=2} = \{a, d\} \\ t = 3: &\quad \sigma_{t=3} = \{c\} \end{aligned}$$

Note that renaming states in a sequence is not important for comparing them via the subset operator as defined in Definition 13. It only matters that states refer to the same real time when they are compared. The answer state sequence I is therefore correctly a proper superset of D.

Proposition 8 ***Answer Stream to Answer State Equivalence.*** *An answer stream I in an interpretation $M = \langle I, W, B \rangle$ of a LARS program P at some time t is identical to the answer state sequence translation $LSS(I)$ in the interpretation $M' = \langle LSS(I), O, B \rangle$ of the Metric LARS translation P' of P on at the same point in time t, if the oracle intervals O can emulate the intervals imposed by the LARS window functions W.*

Proof. Assume the answer stream and answer state were different. This means that either or both of them contain different propositions. For this to be true, either the stream I and timed state sequence $LSS(I)$ had a different mapping of real time to sets of propositions or the program consists of different rules. Since the background data is the same in both structures and the streams contain the same information, as shown in the section on stream translation, only the rules may be different. As we have shown, the translation of rules is semantically identical to their LARS/Metric LARS counterpart, which means that the FLP-reduct has to be the same as well. Therefore either one of the two answer streams/states is not a minimal model of the reduct or not a model at all, contradicting the assumption. □

4.5.7 Semantic Properties

Now we have a look at the properties of the answer states. We start with the minimality of answer state sequences, incomparability of answer state sequences, then the supportedness of answer state sequences and lastly consistency of positive Metric LARS programs.

As we have seen that there is an equivalence of LARS answer streams and Metric LARS answer states, the proofs will be rather simple as these properties have already been established for LARS by Beck et al. [BDTE18]. Similarly, we let the sets of Background Data B and Oracle intervals O be fixed in the following theorems.

Theorem 5 ***Minimality.*** *Let $\mathfrak{P}$ be a Metric LARS program, D be a data sequence, t be a time point and $AS(\mathfrak{P}, D, t)$ denote its answer states. Then by Definition 16 each interpretation sequence $I \in AS(\mathfrak{P}, D, t)$ builds a minimal model $\mathfrak{M} = \langle I, O, B\rangle$ of $\mathfrak{P}$ for D at time t.*

Proof. Here we have to show that each minimal model of $\mathfrak{P}^{\mathfrak{M},t}$ is also a minimal model of $\mathfrak{P}$. Consider $\mathfrak{M} = \langle I, O, B\rangle$ and $I \in AS(\mathfrak{P}, D, t)$. To be a model of $\mathfrak{P}$, each rule $r \in \mathfrak{P}$ has to be satisfied by the model $\mathfrak{M}, t \models r$. Now r can either be satisfied in $\mathfrak{P}^{\mathfrak{M},t}$ as defined at t, or the rule body is not fulfilled, satisfying the implication, so that $\mathfrak{M}, t \not\models body(r)$ and $\mathfrak{M}, t \models body(r) \rightarrow head(r)$.

Next assume there exists some $\mathfrak{M}' = \langle I, O, B\rangle$, where $\mathfrak{M}' \subset \mathfrak{M}$ is a model of $\mathfrak{P}$ at time t. Suppose a rule $r \in \mathfrak{P}^{\mathfrak{M},t}$. As $\mathfrak{P}^{\mathfrak{M},t} \subseteq \mathfrak{P}$ and $\mathfrak{M}', t \models \mathfrak{P}^{\mathfrak{M},t}$ holds, it follows that $\mathfrak{M}', t \models r$. Thus $\mathfrak{M}', t \models \mathfrak{P}^{\mathfrak{M},t}$ must also hold, which contradicts that $\mathfrak{M}$ is an answer set of $\mathfrak{P}$ at t. $\square$

Corollary 1 ***Incomparability.*** *As was seen in Theorem 5, answer states are minimal. Therefore the answer states cannot be compared by the subset operator $\subseteq$. This means for each $I, J \in AS(\mathfrak{P}, D, t)$ either $I = J$ or $I \not\subseteq J$ and $J \not\subseteq I$.*

Theorem 6 ***Supportedness.*** *Let $I \in AS(\mathfrak{P}, D, t)$ be an interpretation sequence. Each intensional proposition $p \in I \backslash D$ in any state $\sigma_i \in I$ has some justifying rule $r \in \mathfrak{P}$, so that both*

- $\mathfrak{M}, t \models body(r)$
- $\mathfrak{M}', t \not\models r$*, where* $\mathfrak{M}' = \langle I \backslash \{p \in \sigma_i\}, O, B\rangle$ *and* $I \backslash \{p \in \sigma_i\}$ *denotes the interpretation sequence* I *without the proposition* p *in the state* $\sigma_i \in I$.

For ordinary logic programs, where all rule heads consist of either a proposition or a disjunction of propositions, the semantics of ordinary answer set programs is used, that states that per answer set one of the head propositions has to be in the set.

Proof. Let $I \in AS(\mathfrak{P}, \mathfrak{D}, \mathfrak{t})$, so that $\mathfrak{M} = \langle I, O, B \rangle$ and $p \in I \backslash D$, so that p is an intentional proposition in a state σ_i, denoted $p \in \sigma_i$. To prove the supportedness, we construct a contradiction. Assume that for all $r \in \mathfrak{P}$ either $\mathfrak{M}, t \not\models body(r)$ or $\mathfrak{M}', t \models r$, where $\mathfrak{M}' = \langle I \backslash \{p \in \sigma_i\}, O, B \rangle$.

The first case cannot hold for all rules, as the reduct would be empty in this case and the data sequence D being the interpretation sequence $I = D$ is a contradiction with $p \in I \backslash D$ and $p \in \sigma_i$.

In the second case some rules are part of the reduct, i.e. $\mathfrak{P}^{\mathfrak{M},t} \neq \emptyset$. Since $\mathfrak{M}', t \models r$ for all rules $r \in \mathfrak{P}^{\mathfrak{M},t}$ and $\mathfrak{M}' \subset \mathfrak{M}$, the model $\mathfrak{M}$ is not a minimal model of $\mathfrak{P}^{\mathfrak{M},t}$. This contradicts the assumption that $\mathfrak{M}$ is an answer state of $\mathfrak{P}$ for D at t. □

Theorem 7 ***Consistency.*** *Let $\mathfrak{P}$ be a positive Metric LARS program such that all rule heads are satisfiable. Then for any data sequence D and time point t:*

- *The set of answer states is not empty $AS(\mathfrak{P}, D, t) \neq \emptyset$.*
- *An interpretation $\mathfrak{M} = \langle I, O, B \rangle$ is a minimal model of the reduct $\mathfrak{P}^{\mathfrak{M},t}$ at t iff I is an answer state $I \in AS(\mathfrak{P}, D, t)$.*

Proof. For the first statement assume we let all intentional propositions be true in some model $\mathfrak{M}$. Now any model $\mathfrak{M}' \subseteq \mathfrak{M}$ of $\mathfrak{P}^{\mathfrak{M},t}$ for D at time t satisfies $\mathfrak{M}, t \models \mathfrak{P}$ by monotonicity $\mathfrak{M}', t \not\models body(r)$ for all $r \in \mathfrak{P} \backslash \mathfrak{P}^{\mathfrak{M},t}$. We repeat this for $\mathfrak{M}'$, creating a sequence $\mathfrak{S} = \mathfrak{M}, \mathfrak{M}', \mathfrak{M}'', \ldots$ of strictly decreasing models of $\mathfrak{P}$. Let the intersection of all models in $\mathfrak{S}$ be denoted by $\mathfrak{N}$, which is again a model of $\mathfrak{P}$ for D at time point t. This also means that $\mathfrak{N}$ is a minimal model and therefore the answer state of the program.

The second statement follows from Theorem 5. According to the minimality of answer states, each $I \in AS(\mathfrak{P}, D, t)$ with $\mathfrak{M} = \langle I, O, B \rangle$ is a minimal model of $\mathfrak{P}^{\mathfrak{M},t}$. With the previous sequence construction, starting with any minimal model $\mathfrak{M}$ yields $\mathfrak{M} = \mathfrak{N}$, so therefore $I \in AS(\mathfrak{P}, D, t)$ holds. □

4.5.8 A Note on Translation Complexity

As we have seen in this chapter, there can be a direct translation between LARS and MTL formulas. Since we only deal with a fragment of MTL and LARS, the corresponding language was called Metric LARS. However the translation needs to break the LARS window function up into the covered time points to properly work. This means that a translated LARS formula with a nesting depth greater than 1 will grow exponentially in size.

Lemma 2 ***Metric LARS Translation Size.*** *Given a LARS formula ϕ with nesting depth $\delta = \delta(\phi)$, the corresponding MTL$_P$ equivalent will be of size $O(2\,\iota^{\delta})$, where $\iota = \iota(\phi)$ is the size of the greatest interval covered by a window function in ϕ.*

The factor of 2 is related to the base case ($\delta = 0$) where we have to split the intervals into past and future parts. Just like in LARS, this means that the performance is far better if the nesting depth of window functions is bounded by a constant. Otherwise we suffer the exponential explosion similar to translations in other normal forms like CNF.

Example 16 *Consider a LARS formula with a nesting depth of 0, so no nested window functions occur. This may look like*

$$\boxplus^{[n,m]}\Diamond\phi.$$

To translate this formula to MTL, we only need to split it into its past and future part:

$$\diamondminus_{[0,n]}\ \phi \vee \diamondplus_{[0,m]}\ \phi$$

The size expansion is by a factor of 2, just like predicted in the formula above: $2(n+m)^0 = 2$.

Now consider the formula $\boxplus^{[2,2]}\Diamond\boxplus^{[1,2]}\Box p$ from Example 12. The nesting depth here is 1 and the size of the biggest interval is 5 because of the window $\boxplus^{[2,2]}$. After the translation the corresponding MTL formula is as follows:

$$\begin{aligned}
\boxplus^{[2,2]}\Diamond\boxplus^{[1,2]}\Box p \quad \hat{=} \quad & \diamondminus_{[0,0]}\ (\boxminus^{[0,1]}p \wedge \boxplus^{[0,2]}p)\vee \\
& \diamondminus_{[1,1]}\ (\boxminus^{[0,1]}p \wedge \boxplus^{[0,1]}p)\vee \\
& \diamondminus_{[2,2]}\ \boxplus^{[0,2]}p\vee \\
& \diamondplus_{[0,0]}\ (\boxminus^{[0,1]}p \wedge \boxplus^{[0,2]}p)\vee \\
& \diamondplus_{[1,1]}\ (\boxminus^{[0,1]}p \wedge \boxplus^{[0,1]}p)\vee \\
& \diamondplus_{[2,2]}\ \boxminus^{[0,1]}p
\end{aligned}$$

The proposition p now occurs in 10 different subformulas, which is equal to the estimation of $2 \times 5^1 = 10$.

CHAPTER 5

Relationship with other Real Time Logics

In this chapter we consider some other logics that extend Metric LARS or serve as other formalizations of the given expressions. For this reason, we have a look at Signal Temporal Logic, an extension of Metric Interval Temporal Logic, and Timed Propositional Temporal Logic, the original real time logic on which MTL is based.

5.1 Metric Interval Temporal Logic

The first real time logic we introduce is *Metric Interval Temporal Logic*, or short MITL. It was first described by Alur et. al. in 1996 [AFH96]. This is a fragment of MTL that was made decidable by inhibiting the use of point intervals in temporal operators.

We start by defining the timed state sequences in MITL, which contain different notions than the ones we have seen so far.

Definition 17 ***Timed State Sequence.*** *Given a time domain $T = \mathbb{R}_{\geq 0}$ starting at time point 0 without an upper bound, we define an interval sequence $\bar{I} = I_0\, I_1 \ldots$ as an infinite sequence of intervals so that*

- *I_0 is left-closed, i.e. it includes the starting point, and starts at 0*
- *Each interval is adjacent to its neighbors, so that each interval I_i is either right-open and I_{i+1} is left-closed, or the other way around*
- *Every time point $t \in T$ belongs to some interval I_i*

Let P be a finite set of propositions. A state sequence $\bar{s} = s_0\, s_1 ...$ is an infinite sequence of states $s_i \subseteq P$.
A Timed State Sequence *$\tau = (\bar{s}, \bar{I})$ consists of a state sequence and an interval sequence. For each time point $t \in I_i$, we retain the corresponding state by using the function $\tau^* : T \rightarrow \bar{s}$, so that $\tau^*(t) = s_i$.*

Intuitively this definition wants to model the same notion as regular MTL, but instead of time points, the states are identified by intervals and count as long as their interval defines. The syntax and semantics of MITL is the same as in MTL with the sole exception that subscript intervals may never be punctual. This means that Intervals of the form $I = [i, i]$ are prohibited.

5.1.1 Defining the MITL Fragment for Metric LARS

There have already been many attempts to create MITL fragments to create MITL logic programs, as for instance in [BKK+17]. Since MITL is a fragment of MTL we will continue by proposing a formalization of MITL within Metric LARS.

Definition 18 ***Continuous Interval Sequence.*** *Given a Metric LARS timed state sequence $S = \{\langle \sigma_i, i\rangle \;\mid 0 \leq i < n, n \in \mathbb{N}\}$ and function $\tau : \mathbb{N} \mapsto T$, mapping each state to the system time, then the Continuous Interval Sequence of S yields the following MITL state sequence:*

$$CIS(S) = \{\, (\sigma_i, I_i) \mid I_i = [\tau(i), \tau(i+1) - 1] \wedge \langle \sigma_i, i\rangle \in S \,\} \tag{5.1}$$

We further define LARS MITL as a fragment of Metric LARS:

Definition 19 ***MITL fragment of Metric LARS.*** *A Metric LARS formula that does not contain an interval of the form $I = [i, i]$ is said to be in MITL. A rule is in MITL, if every formula contained in said rule is in MITL. If each rule in a Metric LARS program is in MITL, the entire program is in MITL.*

To evaluate MITL in our formalization the way its unique semantics was intended, we adopt above notion of timed state sequences for evaluation of the MITL fragment of Metric LARS:

Proposition 9 ***Evaluation of MITL Programs.*** *Metric LARS programs of the MITL fragment can be evaluated on any timed state sequence using the Metric LARS* *semantics. This is done by converting a timed state sequence using the CIS to the proper MITL sequence.*

Note that LARS formulas that shall be translated to MITL may not include nested window functions. Plain LARS formulas without the @ operator can therefore be directly translated. This greatly reduces the complexity of LARS formulas in this fragment.

5.2 Signal Temporal Logic

To increase the possibilities of MITL, *Signal Temporal Logic* was designed in 2004 by Maler and Nickovic [MN04]. It was made to model continuous signals for interval bounded MITL. These signals require a non-trivial definition:

Definition 20 ***Signal.*** *Let T be the time domain so that $T = \mathbb{R}_{\geq 0}$. A signal is a function s that maps the time domain to some domain D. In the case of STL, we restrict D to the set $\mathbb{B} = \{\top, \bot\}$. Then $s : T \to \mathbb{B}$ is defined over an interval $I = [0, r)$, where r is the length of the signal. Each point in time $t > r$ that is not defined by the signal is defined as $s[t] = \bot$.*

The semantics of signals is defined as *Interval Coverings*:

Definition 21 ***Interval Covering.*** *Let $I = [0, r)$ be an interval. An interval covering of I is an interval sequence $\bar{I} = I_0 I_1 ...$ as in MITL. We define the following properties:*

- *Consistency: An interval covering is consistent with a signal s if $s[t] = s[t']$ for every t, where t' belongs to the same interval I_i. In this case we write $s(I_i)$.*
- *Finite Variability: A signal s with a finite interval covering is said to be of finite variability.*
- *Refinement: An interval covering $\bar{I}$ refines $\bar{I}'$ iff $\forall I \in \bar{I} \; \exists \, I' \in \bar{I}' : I \subseteq I'$.*

The minimal consistent interval covering of a signal s with finite variability is denoted by $\bar{I}_s$. Such an interval covering can be split into positive and negative sets, where the positive intervals of s are $\bar{I}_s^+ = \{I \in \bar{I} : s(I) = \top\}$. The negative intervals can be defined simply as $\bar{I}_s^- = \bar{I}_s \backslash \bar{I}_s^+$.

STL features the same syntax as MITL, however satisfaction of a formula works a bit differently as there are no timed state sequences. Instead the semantics for the various operators of STL uses the projection function π on signals s. This function filters the specified frequency, or in this case proposition, p from a signal. This means signals $s_p : T \to \mathbb{B}$, $s_q : T \to \mathbb{B}$ and $s_{pq} : T \to \mathbb{B} \times \mathbb{B}$ can be combined respectively separated by the pairing and projection functions as such:

$$\begin{aligned} s_p \parallel s_q &= s_{pq} \quad \textit{if } \forall t : s_{pq}[t] = (s_p[t], s_q[t]) \\ s_p &= \pi_p(s_{pq}) \\ s_q &= \pi_q(s_{pq}) \end{aligned} \tag{5.2}$$

where $\|$ is the pairing function and π_p is the projection function.

With these concepts now fully explained, we can define the semantics of the STL syntax elements:

Definition 22 ***STL Semantics.***

$$\begin{aligned}
(s,t) &\models p && \textit{iff} \quad \pi_p(s)[t] = \top \\
(s,t) &\models \neg\phi && \textit{iff} \quad (s,t) \not\models \phi \\
(s,t) &\models \phi_1 \rightarrow \phi_2 && \textit{iff} \quad (s,t) \not\models \phi_1 \textit{ or } (s,t) \models \phi_2 \\
(s,t) &\models \diamond_{[a,b]}\phi && \textit{iff} \quad \exists\, t' \in t + [a,b] : (s,t') \models \phi \\
(s,t) &\models \square_{[a,b]}\phi && \textit{iff} \quad \forall\, t' \in t + [a,b] : (s,t') \models \phi
\end{aligned} \tag{5.3}$$

Note that the implication and negation build the full Boolean algebra as seen in Equations 3.2 to 3.5.

The most distinctive feature of STL compared to MITL is the usage of predicates. For this we need to relax the previous constraint that the target domain of signals may only be used for the set $\mathbb{B} = \{\top, \bot\}$ and define real-valued signals.

Definition 23 ***Real-Valued Signals.*** *Let T be the time domain and $\mathbb{R}^m$ be any m dimensional space of real values. A real-valued signal defines a function $s : T \rightarrow \mathbb{R}^m$.*

As real-valued signals may be chaotic, we introduce the notion of well-behaved signals, to which we will constrain the usage of STL predicates. If a signal has an unbounded frequency, the function μ may create a Boolean signal of infinite variability.

Definition 24 ***Well-behaved Signals.*** *Assume a function $\mu : \mathbb{R} \rightarrow \mathbb{B}$ and a signal s. We restrict STL to signals where each $\mu(s)$ is of finite variability and every change of $\mu(s)$ is detected, so that each t, where*

$$\mu(s[t]) \neq lim_{t' \rightarrow t}\mu(s[t']) \tag{5.4}$$

holds is included in the sampling. This is similar to smoothing the signal s.

Of course the previous semantics has to be redefined as m-dimensional vectors can hardly be used with regular logic connectors. For this reason we define a function $\mu : \mathbb{R}^m \rightarrow \mathbb{B}$ to "binarize" the real-valued signals.

Definition 25 ***STL Predicates.*** *Let $s : T \to \mathbb{R}^m$ be a well-behaved signal and $U = \{\mu_1, ..., \mu_n\}$ be a set of m-ary predicates. Predicates are effective functions of the form $\mu_i : \mathbb{R}^m \to \mathbb{B}$. An STL(U) formula is then an interval-bounded MITL formula over the atomic propositions $\mu_1(x), ..., \mu_n(x)$.*

As Maler and Nickovic [MN04] have established, STL predicates can be transformed to propositions by creating a new Boolean signal.

Proposition 10 ***STL Predicate Transformation.*** *Any signal s with respect to some set of predicates U can be transformed into a Boolean signal $s' : T \to \mathbb{B}^n$ so that $s' = \mu_1(s) \parallel ... \parallel \mu_n(s)$. Every STL formula ϕ on signal s can be made propositional, by replacing each occurrence of $\mu_i(x)$ by a proposition p_i and evaluating it on s'.*

Example 17 *Consider a simple real-valued signal s that corresponds to the function $s(x) = \sin(x)$. Further we assume two STL predicates: $gtz(s)$ and $ltz(s)$. These correspond to the greater than zero and less than zero functions respectively. By applying the predicate transformation we can create a new Boolean signal $s' = gtz(s) \parallel ltz(s)$. This new signal will alternate the two Boolean output parameters in intervals of π length.*

5.2.1 Defining an STL Fragment for Metric LARS

Similar to Metric Interval Logic, the syntax of STL is very similar to MTL, therefore we only need to find a way to translate the structure on which STL is evaluated to timed state sequences as used by MTL and Metric LARS. We start by defining a signal representation of MTL timed state sequences. In the translation from a timed state sequence to a signal, we have to make each proposition its own partial signal. This means we do not have to deal with real-valued signals, as it can be modelled using only Boolean signals.

Definition 26 ***Timed State Signal.*** *Let S be a Metric LARS timed state sequence $S = \{\langle \sigma_i, i \rangle \mid 0 \leq i < n, n \in \mathbb{N}\}$ with a function $\tau : \mathbb{N} \mapsto T$, mapping each state to the system time, we define a signal for each proposition $p_1, ..., p_n$ in S as follows:*

$$s_i = \{ [\tau(j), \tau(j+1)] \mid \forall j : \langle \sigma_j, j \rangle \in S \land p_i \in \sigma_j \} \tag{5.5}$$

The Timed State Signal of S is defined as the following STL signal:

$$\begin{aligned} TSS_1(S) &= s_1 \\ TSS_{i+1}(S) &= TSS_i(S) \parallel s_{i+1} \\ TSS(S) &= TSS_n(S) = s_1 \parallel s_2 \parallel ... \parallel s_n \end{aligned} \tag{5.6}$$

where s_i is a partial Boolean signal that corresponds to one proposition p_i in S.

Intuitively, we try to incrementally build the signal by individually looking at each proposition in the timed state sequence. Each proposition is stored in a subsignal $s_i : T \to \mathbb{B}$, which is represented by a set of intervals, that denote its positive up-time. This signal is then paired with the resulting signal using the pairing function. The result is an n dimensional signal modelling the up- and down-time of each proposition.

The other way around we have to model a timed state sequence starting from a signal. Because real-valued signals may occur, the sequence first has to transform the signal into a multidimensional Boolean signal as it was proposed in Definition 10.

Definition 27 ***Signal State Sequence.*** *Let $s : T \to \mathbb{R}^m$ be an m-dimensional real-valued signal and a set $U = \{\mu_1, ..., \mu_n\}$ of m-ary predicates. Let $s' : T \to \mathbb{B}^n$ be a transformation of signal s, where each dimension in the target domain corresponds to a predicate in U, so that $s' = \mu_1(s) \parallel ... \parallel \mu_n(s)$. Next construct a set $P = \{p_1, ..p_n\}$ of propositions corresponding to each signal $\mu_i(s)$ in s'. We define the SSS(s) as the following MTL timed state sequence:*

$$SSS(s) = \{(\sigma_i, i) \mid \sigma_i = \{p_j \mid \mu_j(s)[t] = \top\} \wedge \sigma_i \neq \emptyset,\, \tau(t) = i,\, t \in T\} \tag{5.7}$$

where i is the index of the timed state at time point t. This means that each time point that is not empty – so that there is at least one STL predicate true in this moment – receives a new timed state.

With this we now propose the Signal Temporal Logic Fragment of Metric LARS:

Definition 28 ***STL Fragment of Metric LARS.*** *A Metric LARS formula that does not contain a punctual interval of the form $I = [i, i]$, where every occurring interval $I = [s, e]$ is bounded by a real number $s, e \in \mathbb{R}$, is in the MITL$_{[a,b]}$ fragment of Metric LARS. If this formula is to be evaluated on a real-valued or Boolean signal, the formula is said to be in STL. A Metric LARS rule that only consists of STL formulas is said to be in STL as well. A Metric LARS program is in STL, if each rule in it is in STL.*

With the STL fragment of Metric LARS we can now model real-valued time domains, as well as predicates which makes a great additions to Metric LARS. It must be noted, that as with MITL the price of the fragment is, that it can not translate any nested window functions in LARS. However, since atoms are allowed in Plain LARS, this fragment shows almost the full capability of Plain LARS with the sole exception being the restriction of punctual intervals. Another important consideration is that STL predicates all have the same arity n, corresponding to the number of channels in a signal. For predicates with an assumed arity smaller than n certain channels may be omitted.

5.3 Timed Propositional Temporal Logic

The last logic we have a look at is *Timed Propositional Temporal Logic* (TPTL). This logic was proposed by Alur and Henzinger in 1989 and is the first real time logic they worked on [AH89]. This logic introduced the notion of timed state sequences, which was then adopted by the authors for Metric Temporal Logic. This means the semantic structure on which formulas have to be evaluated does not change from the original definition of timed state sequences as shown in this thesis. However the syntax of this logic is quite different as we will see.

5.3.1 Syntax

TPTL was designed as an extension of Linear Temporal Logic by adding time variables. These variables are bound by a *freeze quantifier* $x.\phi$, where x is the frozen variable. This makes TPTL more expressive than MTL since absolute time points can be defined in a formula instead of only sliding time. It features the following syntax:

Definition 29 ***TPTL Syntax.*** *A TPTL formula is of the form:*

$$\begin{aligned} \pi &::= x + c \mid c \\ \phi &::= p \mid \bot \mid \pi_1 \leq \pi_2 \mid \pi_1 \equiv_d \pi_2 \mid \phi_1 \to \phi_2 \mid \oplus\, \phi \mid \phi_1 \mathcal{U}\, \phi_2 \mid x.\phi \end{aligned} \tag{5.8}$$

where x is a time variable, c is a constant and ϕ is a formula. Arithmetic expressions concerning time variables are represented by π-terms. TPTL also features the *Next* operator $\oplus$.

5.3.2 Semantics

As said, the semantics of TPTL is defined over timed state sequences, the same as the ones used in MTL. This makes the semantics easily definable within the context of this thesis.

Definition 30 ***TPTL Semantics.*** *Given a timed state sequence S and a function τ mapping state identifiers to the system time. Further let V be the set of variables and $\mathfrak{I} : V \to T$ be the environment for variables. The semantics of TPTL formulas for a time* *point t is defined as follows:*

$$\begin{array}{lll}
S,t \models_{\mathfrak{I}} \bot & \textit{iff} & S,t \not\models_{\mathfrak{I}} \bot \\
S,t \models_{\mathfrak{I}} p & \textit{iff} & p \in \sigma_i \wedge \tau(i) = t \\
S,t \models_{\mathfrak{I}} \pi_1 \leq \pi_2 & \textit{iff} & \mathfrak{I}(\pi_1) \leq \mathfrak{I}(\pi_2) \\
S,t \models_{\mathfrak{I}} \pi_1 \equiv_d \pi_2 & \textit{iff} & \mathfrak{I}(\pi_1) \equiv_d \mathfrak{I}(\pi_2) \\
S,t \models_{\mathfrak{I}} \phi_1 \rightarrow \phi_2 & \textit{iff} & S,t \not\models_{\mathfrak{I}} \phi_1 \textit{ or } S,t \models_{\mathfrak{I}} \phi_2 \\
S,t \models_{\mathfrak{I}} \oplus\phi & \textit{iff} & S,t' \models_{\mathfrak{I}} \phi,\ t' = \tau(i+1),\ t = \tau(i) \\
S,t \models_{\mathfrak{I}} \phi_1 \mathcal{U} \phi_2 & \textit{iff} & \exists t' \geq t : S,t' \models_{\mathfrak{I}} \phi_2 \wedge \\
 & & \forall t'' \in [t,t') : S,t'' \models_{\mathfrak{I}} \phi_1 \\
S,t \models_{\mathfrak{I}} x.\phi & \textit{iff} & S,t \models_{\mathfrak{I}[x:=t]} \phi
\end{array} \tag{5.9}$$

Compared to the semantics of MTL this definition should look rather familiar. The sole exception is the new freeze quantifier and the lack of interval boundaries on the Until connective. Alur and Henzinger proposed different extensions of their logic which we will not cover in this thesis. Noticeable is the past time extension TPTL$_P$, which introduces the Before $\ominus$ and Since $\mathcal{S}$ operators.

5.3.3 TPTL Extensions for Metric LARS

Since MTL was defined as a fragment of TPTL, the translation of MTL to TPTL was already given in [AH93]. However TPTL is more expressive, so there are formulas that cannot be expressed properly by MTL. The TPTL translation of the MTL operators is as follows:

Definition 31 ***MTL$_P$ to TPTL$_P$ Translation.***

$$\begin{array}{lcl}
MTL_P & & TPTL_P \\
p & \hat{=} & p \\
\bot & \hat{=} & \bot \\
\phi_1 \rightarrow \phi_2 & \hat{=} & \phi_1 \rightarrow \phi_2 \\
\oplus_I\ \phi & \hat{=} & x.\oplus y.(y \in x + I \wedge \phi) \\
\ominus_I\ \phi & \hat{=} & x.\ominus y.(y \in x - I \wedge \phi) \\
\phi_1 \mathcal{U}_I \phi_2 & \hat{=} & x.(\phi_1 \mathcal{U} y.(y \in x + I \wedge \phi_2)) \\
\phi_1 \mathcal{S}_I \phi_2 & \hat{=} & x.(\phi_1 \mathcal{S} y.(y \in x - I \wedge \phi_2)) \\
\diamondplus_I\ \phi & \hat{=} & x.(\top \mathcal{U} y.(y \in x + I \wedge \phi)) \\
\diamondminus_I\ \phi & \hat{=} & x.(\top \mathcal{S} y.(y \in x - I \wedge \phi))
\end{array} \tag{5.10}$$

As before, the $\boxplus$ and $\boxminus$ operators can be derived from the $\diamondplus$ and $\diamondminus$ operators.

We can see that only certain formulas can be translated from TPTL to MTL, as we would expect. A full extension of Metric LARS to TPTL will however not be given as this is beyond the scope of this thesis. In this sense the translation given in this chapter is rather meant as a Metric LARS fragment for TPTL.

CHAPTER 6

Conclusion and Outlook

In this chapter we summarize our findings and reflect on work that still has to be done. We start with the conclusion and close the thesis with an outlook on future work.

6.1 Conclusion

The thesis started with a broad introduction in logic programs and showed the capability of answer set programming, which is form of logic programming. We then had a look at modal logic and in particular the temporal logic family, with the example of Linear Temporal Logic. At last, we presented the syntax and semantics of the LARS framework, a logic based framework for stream reasoning tasks using answer set programming.

From there we explored Metric Temporal Logic as it was originally designed in 1993 by Alur and Henzinger. This inspection featured the full proofs for the expressivity and complexity of the logic.

In the main part of this thesis, we explored different fragments of the LARS language and how to translate them to MTL. We introduced Plain Metric LARS, a translation of Plain LARS to MTL and back. To make the translation to the full syntax of LARS formulas possible, we first introduced a new normal form, called Nested Window Normal Form (NWNF). This normal form serves the purpose to guard LARS temporal operators with window functions. This was done due to the combined nature of MTL temporal operators serving as window functions and operators at the same time. Based on the NWNF of LARS formulas, we could then transform them to MTL syntax and proved the semantic correctness.

In the last step of the translation, we revisited the LARS window functions and lifted some constraints on them, like a mixture of past- and future-time intervals. We introduced the notion of oracle intervals to simulate windows that do not use sliding interval boundaries. These oracles may further be able to help us to create non-monotonic temporal operators in MTL. Overall this final translation suffers an exponential blow-up in size.

Some features of LARS had to be simulated as they had no direct correspondence within MTL. This was the direct access to system time, as in the LARS @ operator, or window functions that were non-monotonic.

We defined the language of LARS-translatable MTL formulas as Metric LARS and provided answer set semantics for its logic programming fragment. Since the semantic structures on which the different logics are evaluated differ, we also proposed various ways to translate the LARS streams to MTL timed state sequences and back. Depending on the situation, streams may feature different semantics so MTL may express a stream in another fashion. The different ways we proposed serve to show the options instead of one unified translation.

In the last chapter we had a look at different related logics and created fragments of Metric LARS to utilize their unique features. We first looked at Metric Interval Temporal Logic, which restricts the usage of punctual intervals in operators. This logic features its own type of timed state sequences using intervals rather than timed states, where propositions hold. We proposed a translation of timed state sequences to their interval counterpart and introduced a fragment of MITL for Metric LARS. Based on MITL, we looked at Signal Temporal Logic, which extends the idea of using intervals rather than time points and applied it to signals on real-valued time domains. This logic first defines MITL on Boolean signals, but introduces the notion of predicates on real-valued signals. Exploring this idea, we provided a translation to and from STL signals and an STL fragment for Metric LARS. The extension of MITL to signals and predicates may make expressions more succinct while still being quite expressive. The last real-time logic observed was Timed Temporal Logic, of which MTL is a fragment. This logic is evaluated on the same timed state sequences as MTL, but uses a freeze operator that makes it more expressive. We presented the translation given by Alur and Henzinger [AH93] to transform TPTL to MTL and back. This plethora of new formalisms and translations shows how much there is to learn from other logics and how they treat data streams. Having a great variety of inter-translatable suggestions for data semantics makes real-time logics even better at modelling real-world scenarios.

6.2 Outlook

This thesis with all its findings served only as a theoretical work and still misses an actual application on which it can be tested. Answer set programming features great properties for reasoning tasks and stream reasoning is a very highly sought after field of research. Applying ASP to tasks of stream reasoning is therefore a valuable cause as it can directly model these problems using the various formalisms. Especially with the many ways data streams can be handled by signals or state sequences, we assume that interest in ASP for stream reasoning will only grow in the future.

Although there are many potential fields of application, the first step has to be to implement the translator to make Metric LARS programs executable within the LARS framework. On the other hand, a Metric LARS reasoner might lead to further insight and inspire progress in other reasoners due to the nature of timed state sequences. For this reason, further research on oracle intervals will be needed to implement non-monotonic LARS window functions like tuple-based windows and see the boundaries of Metric LARS oracles in actual reasoning tasks.

This thesis does not feature a full study of the computational complexity for Metric LARS and its fragments. MTL has been thoroughly studied in this regard but this task is still open for the new logics we proposed in this thesis. Future theoretical work may start with this analysis.

Another topic for future work may be the extension of LARS to Timed Temporal Logic. As we have seen in this thesis, Metric LARS would require a rather big extension to be lifted to the same expressivity as TPTL. In this sense, a direct translation from LARS to TPTL and back may be more interesting, as we had to omit certain features of LARS for MTL anyways. Most likely such a translation would have similar problems as MTL when it comes to the correct representation of non-trivial window functions. The translation may however be more succinct than Metric LARS.

Bibliography

[AFH96] Rajeev Alur, Tomás Feder, and Thomas A Henzinger. The benefits of relaxing punctuality. *Journal of the ACM (JACM)*, 43(1):116–146, 1996.

[AH89] Rajeev Alur and Thomas Henzinger. A really temporal logic. In *Proc. 30th IEEE FOCS*, pages 181–204, 1989.

[AH93] Rajeev Alur and Thomas A Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.

[BDTE16] Harald Beck, Minh Dao-Tran, and Thomas Eiter. Equivalent stream reasoning programs. In *International Joint Conference on Artificial Intelligence*, pages 929–935, 2016.

[BDTE18] Harald Beck, Minh Dao-Tran, and Thomas Eiter. Lars: A logic-based framework for analytic reasoning over streams. *Artificial Intelligence*, 261:16–70, 2018.

[BET11] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[BKK$^+$17] Sebastian Brandt, Elem Güzel Kalaycı, Roman Kontchakov, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyaschev. Ontology-based data access with a horn fragment of metric temporal logic. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31(1), pages 1070–1076, 2017.

[BvBW06] Patrick Blackburn, Johan FAK van Benthem, and Frank Wolter. *Handbook of modal logic*. Elsevier, 2006.

[CDSS20] Pedro Cabalar, Martin Dieguez, Torsten Schaub, and Anna Schuhmann. Towards metric temporal answer set programming. *Theory and Practice of Logic Programming*, 20(5):783–798, 2020.

[DDVvHB17] Daniele Dell'Aglio, Emanuele Della Valle, Frank van Harmelen, and Abraham Bernstein. Stream reasoning: A survey and outlook. *Data Science*, 1(1-2):59–83, 2017.

[DEHLP18] Daniele Dell'Aglio, Thomas Eiter, Fredrik Heintz, and Danh Le-Phuo. Editorial: Special issue on stream reasoning. *Semantic Web Journal*, 2018.

[FT15] Jie Fu and Ufuk Topcu. Computational methods for stochastic control with metric interval temporal logic specifications. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 7440–7447. IEEE, 2015.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Fifth International Conference and Symposium on Logic Programming*, volume 88, pages 1070–1080, 1988.

[Lif19] Vladimir Lifschitz. *Answer set programming*. Springer Berlin, 2019.

[Mar03] Nicolas Markey. Temporal logic with past is exponentially more succinct. *Bulletin of the EATCS*, 79:122–128, 01 2003.

[MN04] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[RKG$^+$18] Alessandro Ronca, Mark Kaminski, Bernardo Cuenca Grau, Boris Motik, and Ian Horrocks. Stream reasoning in temporal datalog. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32(1), pages 1941–1948, 2018.

[RN02] Stuart Russell and Peter Norvig. *Artificial intelligence: A Modern Approach*. Pearson, third edition, 2002.

[Sav70] Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[TCWCGK21] David J Tena Cucala, Przemysław A Wałęga, Bernardo Cuenca Grau, and Egor Kostylev. Stratified negation in datalog with metric temporal operators. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(7):6488–6495, May 2021.

[WKG19] Przemysław Andrzej Wałęga, Mark Kaminski, and Bernardo Cuenca Grau. Reasoning over streaming data in metric temporal datalog. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33(1), pages 3092–3099, 2019.