Practical Reasoning and Rule Mining with DatalogMTL

Dingmin Wang

Linacre College

University of Oxford

A thesis presented for the degree of

Doctor of Philosophy

Hilary 2024

I dedicate this thesis to my parents, Huaying and Yuhua.

Two roads diverged in a wood, and I—

I took the one less traveled by.

And that has made all the difference.

— Robert Frost

New York is 3 hours ahead of California,

but it does not make California slow.

...

Life is about waiting for the right moment to act.

So, RELAX.

You are not LATE. You are not EARLY.

You are very much ON TIME, and in your, TIME

ZONE Destiny set up for you.

— Anonymous

Acknowledgements

I would like to first extend my deepest gratitude to my supervisors, Prof. Bernardo Cuenca Grau and Dr. Przemysław Andrzej Wałęga. I am deeply grateful to Bernardo for his patient mentorship throughout my DPhil journey. He not only taught me how to write quality research papers and the proper conduct of a good researcher, but he also gave me ample freedom to explore various directions that I am interested in and passionate about. Despite his often packed schedule, he always responded promptly whenever I needed his help and made time to discuss and talk with me. In particular, though our conversations rarely strayed from research, his advice against unwarranted comparisons with peers, along with his insistence on the value of high-quality work , has always helped alleviate my anxiety and restore my confidence in this research journey. His wise words, 'Be the Hammer, Not the Nail,' will forever resonate with me. I am grateful to Przemek for dedicating substantial time to mentor me in the art of academic writing and for his involvement in every stage of our paper development, from initial ideation and drafting to the final polished version. His guidance has been particularly crucial in areas where I needed the most help, such as proof development. Przemek patiently walked me through each step, providing insightful comments and steering me in the right direction. Every time I approached him with basic or seemingly naive questions about DatalogMTL and other logical concepts, he always responded with clarity and patience, often pointing me towards helpful reading materials. His willingness to make time for discussions whenever I faced challenges was invaluable. Beyond being an exceptional supervisor, Przemek has also been a good friend throughout my DPhil journey, for which I am profoundly thankful.

I would like to extend my heartfelt thanks to Prof. Ian Horrocks, who has played a pivotal role as an assessor at each milestone of my DPhil studies. During cru-

cial phases such as the Transfer of Status and Confirmation of Status, he provided detailed reports, highlighting valuable suggestions and areas for improvement, all of which have greatly benefited my research. Ian's approachability and friendly demeanor have been a constant source of support. Furthermore, I thank my assessors Prof. Michael Benedikt (Transfer), Dr. Matthias Lanzinger (Confirmation) and Prof. Víctor Gutiérrez Basulto (Viva) their constructive feedbacks.

Besides, I would like to thank friends and collaborators I met at Oxford, Pan Hu, Qi Liu, Michał Zawidzki, David Tena Cucala, Matthew Jackson, Shuwen Liu, Jiaoyan Chen, Yuan He, and Hang Dong, who provide invaluable contributions through insightful research discussions or constructive feedbacks during the writing of various papers were instrumental in my DPhil journey.

Furthermore, I wish to express my profound gratitude to the Engineering and Physical Sciences Research Council (EPSRC) Studentship for providing the financial support necessary for my DPhil studies. Additionally, I am immensely thankful to the Clarendon Fund for honoring me with the prestigious Clarendon Fund Scholarship, and for welcoming me into its diverse, international, and multidisciplinary community.

Finally, this thesis is dedicated to my family and Jun for their continuous love, encouragement in the past years.

List of My Publications

This thesis is based on my following publications:

Dingmin Wang, Pan Hu, Przemysław Andrzej Wałęga, and Bernardo Cuenca Grau. "MeTeoR: Practical reasoning in datalog with metric temporal operators." In Proceedings of the 36th AAAI Conference on Artificial Intelligence, pp. 5906-5913. 2022.

Dingmin Wang, Przemysław Andrzej Wałęga, and Bernardo Cuenca Grau. "Seminaïve Materialisation in DatalogMTL." In International Joint Conference on Rules and Reasoning, pp. 183-197. 2022.

Przemysław Andrzej Wałęga, Michał Zawidzki, Dingmin Wang, and Bernardo Cuenca Grau. "Materialisation-based reasoning in DatalogMTL with bounded intervals." In Proceedings of the 37th AAAI Conference on Artificial Intelligence, pp. 6566-6574. 2023.

Przemysław Andrzej Wałęga, Mark Kaminski, Dingmin Wang, and Bernardo Cuenca Grau. "Stream reasoning with DatalogMTL." Journal of Web Semantics: 100776. 2023.

Dingmin Wang, Przemysław Andrzej Wałęga, Pan Hu, and Bernardo Cuenca Grau. "Practical reasoning in DatalogMTL." Theory and Practice of Logic Programming (2024).

Dingmin Wang, Przemysław Andrzej Wałęga, and Bernardo Cuenca Grau. "MTLearn: Extracting temporal rules using Datalog rule learners". In Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning. 2024

I have published as first-author other papers that are excluded from the thesis:

Dingmin Wang and Yeyuan Chen. "Calibrate and boost logical expressiveness of GNN over multi-relational and temporal graphs." Thirty-seventh Conference on Neural Information Processing Systems. 2023.

Dingmin Wang, Yeyuan Chen, and Bernardo Cuenca Grau. "Efficient embeddings of logical variables for query answering over incomplete knowledge graphs." In Proceedings of the 37th AAAI Conference on Artificial Intelligence, pp. 4652-4659. 2023.

Dingmin Wang, Shengchao Liu, Hanchen Wang, Bernardo Cuenca Grau, Linfeng Song, Jian Tang, Song Le, and Qi Li. “An empirical study of retrieval-enhanced graph neural networks.” European Conference on Artificial Intelligence, pp. 2443–2450. 2023.

Abstract

DatalogMTL is a powerful temporal knowledge representation language, which has recently gained prominence in contexts involving temporal data. Its applications extend to areas like ontology-based query answering, network flow anomaly detection, equipment malfunction monitoring, and streaming reasoning, showcasing its versatility and effectiveness in handling time-sensitive information. Reasoning in DatalogMTL is, however, of high complexity, namely ExpSpace-complete and PSpace-complete with respect to data size, which makes reasoning in data-intensive applications challenging. Thus, theoretical research has focused on establishing a suitable trade-off between expressive power and complexity of reasoning, by identifying lower complexity fragments of DatalogMTL and studying alternative semantics with favourable computational behaviour. The design and implementation of practical reasoning algorithms for DatalogMTL remains, however, a largely unexplored area—something that has so far prevented its widespread adoption in applications. In this thesis, our primary focus lies in the exploration of practical temporal reasoning algorithms in DatalogMTL. Our proposed algorithms were implemented within a system named MeTeoR. The results of our evaluation underscore the practical viability and effectiveness of our approach. Besides, we have developed a universal framework for learning rules written in DatalogMTL which capitalises on the availability of increasingly mature non-temporal rule learners. We have implemented the approach in a system MTLearn which is compatible with any Datalog rule learner, as well as with a wide range of strategies for scoring the output temporal rules. Our evaluation demonstrates its competitive performance, on par with state-of-the-art deep learning models, while also providing easily interpretable results.

Table of Contents

1 | Introduction

1.1 Background and Motivations

Temporal data refers to information that is collected and documented across a period of time, where each data entry is usually associated with a precise timestamp or an time interval. The significance of temporal data lies in its ability to accurately capture the dynamic nature of changes, evolving trends, and complex patterns that unfold over time. This kind of data is ubiquitous in many application scenarios, such as stock trading (Nuti, Mirghaemi, Treleaven, and Yingsaeree, 2011; Wei, 2007), meteorological sciences (Anderson, 2005; Salman, Kanigoro, and Heryadi, 2015), social media (Ferraz Costa et al., 2015), and temporal knowledge graphs – an extensions of knowledge graphs where relations between entities are associated with validity intervals (Leblay and Chekol, 2018).

Reasoning over temporal data involves the formal capture of information that changes or evolves over time. In the past few decades, various temporal languages have been introduced (McDermott, 1982; Abadi and Manna, 1989; Augusto, 2001; Vardi, 2009; Demri, Goranko, and Lange, 2016), including the seminal linear temporal logic (LTL) (Pnueli, 1977) with its fragments and modifcations (Alur and La Torre, 2004; Demri and Schnoebelen, 2002), metric temporal logic (MTL) (Koymans, 1990), Datalog$_{1S}$ (Chomicki and Imieliński, 1988; Baudinet, Chomicki, and Wolper, 1993), Temporal Datalog (Ronca, Kaminski, Cuenca Grau, Motik, and Horrocks, 2018), temporal extensions of description logics (DLs) (Gutiérrez-Basulto, Jung, and Ozaki, 2016; Baader, Borgwardt, Koopmann, Ozaki, and Thost, 2017; Artale and Franconi, 1998), and DatalogMTL which was first proposed by Brandt, Kalaycı, Ryzhikov, Xiao, and Zakharyaschev

(2018) and then further explored by Wałęga, Cuenca Grau, Kaminski, and Kostylev (2019). DatalogMTL is a powerful rule-based language for representing and reasoning about temporal data that has recently received significant attention. Compared with LTL, Datalog$_{1S}$ and Temporal Datalog, which are over a discrete timeline such as $(\mathbb{N}, \leq)$ or $(\mathbb{Z}, \leq)$, DatalogMTL is interpreted over the rational timeline which provides a better way of modelling temporal data and events. DatalogMTL uses the (more succinct) metric temporal logic (MTL) with operators like $\boxminus_\varrho$, $\diamondminus_\varrho$, and $\mathcal{S}_\varrho$ which refer to the past, and $\boxplus_\varrho$, $\diamondplus_\varrho$, and $\mathcal{U}_\varrho$ which refer to the future (details about their semantics could be found in Table 2.1). For example, formulas $\boxminus_{[1,2]}P$ and $\diamondminus_{[1,2]}P$ mean that these two formulas hold at at time t if P holds at each (respectively, some) moment in $[t-2, t-1]$. Besides, for the binary operator $\mathcal{S}$, the $P\mathcal{S}_{[1,2]}Q$ formula holds at time t if Q occurs somewhere between $[t-2, t-1]$ and P holds continuously since Q up to the current time point t. Similarly, the other three future counterparts of such operators model temporal data associated with future time points. The following two examples illustrate the practical use of DatalogMTL rules in different applications.

Example 1 *During the COVID-19 pandemic, many countries implemented strict travel policies. These policies required travelers to meet specific health and immunity criteria before being allowed entry. To facilitate this, consider a system designed to automatically verify travelers' eligibility based on their personal health data, which includes:*

1. *Any COVID-19 symptoms experienced recently, such as fever and sore throat.*
2. *Vaccination status, including the date of vaccination.*
3. *Results of a COVID-19 rapid lateral flow test, specifically if it was negative.*
4. *Any previous COVID-19 infections and their dates.*

According to Feikin, Higdon, Abu-Raddad, Andrews, Araos, Goldberg, Groome, Huppert, O'Brien, Smith, Wilder-Smith, Zeger, Deloria Knoll, and Patel (2022), emerging evidence suggests that individuals can develop immunity to COVID-19 for a minimum of three months under specific conditions. This includes having been vaccinated and either remaining asymptomatic or displaying a negative test result within three to four weeks post-vaccination. Additionally, immunity is considered likely if an individual had a COVID-19 infection within the past six months, provided the last ten days were symptom-free. Leveraging these prior medical knowledge, a system could be designed to function with a set of DatalogMTL rules formulated to assess this information and determine travel eligibility. These rules are detailed as follows:

$$\boxplus_{[0,90]} Immunity(x) \leftarrow NoSymptom(x) \mathcal{S}_{[21,28]} Vaccinated(x), \tag{1.1}$$

$$\boxplus_{[0,90]} Immunity(x) \leftarrow NegativeTest(x) \wedge \diamondminus_{[21,28]} Vaccinated(x), \tag{1.2}$$

$$Immunity(x) \leftarrow \diamondminus_{(10,183]} Infected(x) \wedge \boxminus_{[0,10]} NoSymptom(x), \tag{1.3}$$

$$NegativeTest(x) \leftarrow \boxminus_{[0,5]} Immunity(x), \tag{1.4}$$

$$AuthorisedEntry(x) \leftarrow \diamondminus_{[0,7]} Immunity(x). \tag{1.5}$$

Rules (1.1)–(1.3) *represent the three kinds of conditions for acquiring 3-month-long immunity. Rule* (1.1) *checks whether an individual remained without symptoms between* 21 *and* 28 *days* since *receiving vaccination (using the metric version of the 'since' operator* $\mathcal{S}_{[21,28]}$*); Rule* (1.2) *checks whether they received a negative test and got vaccinated at some point in the last* 21 *to* 28 *days (using the 'diamond past' operator* $\diamondminus_{[21,28]}$*); in turn, Rule* (1.3) *checks whether an individual infected at some point in the last six months excluding the last 10 days (operator* $\diamondminus_{(10,183]}$*) remained continuously without symptoms in the last 10 days (using the 'box past' operator* $\boxminus_{[0,10]}$*). Rule* (1.4) *says that individuals who have been immune for the*

last 5 days will display a negative test result. Ultimately, Rule (1.5) *denotes that an authorised entry is granted to individuals who have maintained a status of immunity at least at some time point in the past seven days.*

Example 2 *Network attacks are an issue in our digital world. Let's explore a scenario where a computer network is rigorously monitored for external threats using an Intrusion Detection Policy (IDP). This network employs specialised monitoring devices to detect bursts of data—indicative of unusually high data transfer—between any two nodes. These occurrences are captured and streamed as timestamped records to the network's management center. One critical task at the center is to identify potentially compromised nodes based on the IDP and subsequently add them to a blacklist. Understanding the contents of this blacklist at any given time is essential for making informed decisions about subsequent actions. The IDP could be defined by a set of DatalogMTL rules shown below,*

$$Attack(x, y) \leftarrow Burst(z, y) \wedge \boxminus_{[1,1]} Burst(x, y), \tag{1.6}$$

$$\boxplus_{[0,\infty)} Blacklist(x) \leftarrow Attack(x, y) \wedge \boxminus_{[1,1]} Attack(x, y). \tag{1.7}$$

Rule (1.6) *defines an attack on a node y within the network as occurring when a data burst from node x to y is followed by another data burst from node z to y after one second. Following this, Rule* (1.7) *establishes a criterion within the Intrusion Detection Policy (IDP) where two successive attacks from node x on node y lead to the node x to be indefinitely (stipulated by $\boxplus_{[0,\infty)}$) added the network's blacklist.*

The most common technique of choice in scalable Datalog reasoners is materialisation (a.k.a. forward chaining), where facts entailed by a program and dataset are derived in successive rounds of rule applications until a fixpoint is reached

(Bry, Eisinger, Eiter, Furche, Gottlob, Ley, Linse, Pichler, and Wei, 2007; Motik, Nenov, Piro, Horrocks, and Olteanu, 2014; Carral, Dragoste, González, Jacobs, Krötzsch, and Urbani, 2019; Bellomarini, Sallinger, and Gottlob, 2018). Once the materialisation has been computed, queries can be answered directly and rules are not further considered. In contrast to Datalog where materialisation naturally terminates, in DatalogMTL a fixpoint may only be reachable after infinitely many rounds of rule applications. Consider, for example, a program consisting of rule $\boxplus_{[0,1]} Alive(x) \leftarrow Alive(x)$, which states that if x is alive at some day, then they will be alive until the next day. Now, consider a dataset with a single fact $Alive(Adam)@0$ saying that Adam is known to be alive in day 0. Then it means that he will be alive always in future, namely, the fact $Alive(Adam)@[0, \infty)$ can be derived. However, each rule application extends the fact $Alive(Adam)$ by only one day. To derive $Alive(Adam)@[0, \infty)$, an infinite number of materialisation steps, in particular ω (the first uncountable ordinal) applications, are required. Although the complexity of standard reasoning tasks in both the full DatalogMTL language and its fragments has been investigated in depth (Brandt et al., 2018; Wałęga, Cuenca Grau, Kaminski, and Kostylev, 2019; Wałęga, Cuenca Grau, Kaminski, and Kostylev, 2021; Bellomarini, Nissl, and Sallinger, 2021), the design and implementation of practical reasoning algorithms for DatalogMTL still remains a largely unexplored area—something that has so far prevented its adoption in applications. In particular, the following constitutes prominent challenges:

1. In DatalogMTL a direct implementation of materialisation-based reasoning systems is problematic since forward chaining may require infinitely many rounds of rule applications. Exploring the development of an algorithm that simultaneously guarantees both soundness and completeness is challenging.

2. Materialisation in Datalog has undergone extensive exploration and numerous optimised techniques have been proposed to enhance its efficiency.

In contrast, materialisation-based reasoning in DatalogMTL faces several challenges, primarily due to the incorporation of metric temporal operators in its rules. The study of efficient materialisation techniques within DatalogMTL remains largely underdeveloped.

3. Crafting DatalogMTL rules is a labor-intensive task that demands considerable expertise. To the best of our knowledge, there is currently no existing research dedicated to the automated learning of DatalogMTL rules. This stands in stark contrast to the advancements in other non-temporal rule-based languages (Meilicke, Chekol, Ruffinelli, and Stuckenschmidt, 2019; Galárraga, Teflioudi, Hose, and Suchanek, 2015; Cropper and Morel, 2021).

1.2 Contributions of the Thesis

Our contributions are twofold. On one hand, we address the previously mentioned Challenges 1 and 2 by developing and implementing practical reasoning algorithms for DatalogMTL. On the other hand, we present a framework specifically designed for learning DatalogMTL rules, targeting Challenge 3. These efforts provide solutions and methodologies for reasoning and learning with DatalogMTL.

Our sound and complete reasoning algorithm for the full DatalogMTL combines materialisation and automata-based reasoning. The automata-based reasoning procedure was primarily proposed to achieve tight complexity bounds and had no efficient implementation. This is primarily because the constructed automata are exponentially large in size, which makes direct implementations impractical. Our approach addresses these difficulties by providing optimised materialisation-based algorithms together with an effective way of combining the scalability of materialisation-based reasoning and the completeness guaranteed by automata-based procedures, thus providing ‘the best of both worlds’. To ensure favourable

scalability of the materialisation component, we propose a novel seminaïve materialisation procedure for DatalogMTL enjoying the non-repetition property, which ensures that each specific rule application will be considered at most once throughout the entire execution of the algorithm. Moreover, our materialisation procedure is enhanced with additional optimisations which further reduce the number of redundant computations performed during materialisation by disregarding rules as soon as it is certain that they cannot derive new facts in subsequent materialisation steps. We have implemented our practical algorithms in a system called MeTeoR[1].

Although the automata construction could be employed to ensure completeness where materialisation does not terminate, our experimental results show that the automata-base procedure will cause a significant performance reduction due to its high computational complexity. Hence, an alternative approach is to focus on DatalogMTL fragments for which materialisation is guaranteed to terminate. Motivated by this and based on the DatalogMTL reasoner MeTeoR, we explore the design of reasoning algorithms for DatalogMTL that rely solely on materialisation, dispense with automata construction altogether, and do not limit recursion. We focus on bounded DatalogMTL programs and datasets, where ∞ is not mentioned as an end-point of any interval. Although materialisation might not naturally terminate in this setting, the structure of canonical interpretations needs to be ultimately periodic. By calculating a partial materialisation satisfying specific saturation conditions, we are able to recover the full canonical interpretation via *unfolding*. We have implemented our algorithm and evaluated its performance on various temporal benchmarks. Our results show that the overhead introduced by saturation checks during materialisation is negligible, saturation can be reached after a reasonable number of rounds of rule applications, and entailment checks performed after reaching saturation can be conducted very efficiently.

[1]The code repository is available at https://github.com/wdimmy/MeTeoR.

Furthermore, we exploit the usage of DatalogMTL in *Stream Reasoning*, the problem of query answering over data streams in the presence of background knowledge. We consider a stream reasoning algorithm that is applicable to forward propagating DatalogMTL programs, in which propagation of derived information towards past time points is precluded. We have implemented this approach as an extension of MeTeoR and tested it experimentally. The obtained experimental results support the feasibility and efficiency of the approach in practice.

In addition to our primary focus on developing efficient reasoning algorithms for scenarios where both the DatalogMTL program and the temporal dataset are available, this thesis also explores the automated extraction of DatalogMTL rules from temporal data. In real-world situations it is often the case that temporal data lacks established temporal rules. This is primarily attributed to the time-consuming and expertise-dependent nature of crafting such rules. Existing research encompasses areas such as Inductive Logic Programming (Muggleton and De Raedt, 1994; Cropper and Dumančić, 2022), Neural Logic Programming (Yang, Yang, and Cohen, 2017), and Rule Learning (Fürnkranz, Gamberger, and Lavrač, 2012; Fürnkranz and Kliegr, 2015). Additionally, various ready-to-use systems for rule extraction, such as AnyBURL (Meilicke, Chekol, Ruffinelli, and Stuckenschmidt, 2019), AMIE+ (Galárraga, Teflioudi, Hose, and Suchanek, 2013), RLvLR (Omran, Wang, and Wang, 2018), RARL (Pirrò, 2020), and Popper (Cropper and Morel, 2021), have been developed, offering practical solutions to this challenge. These systems, however, were not specifically designed to handle temporal data and, as a result, the rules they generate do not incorporate appropriate temporal constructs. We propose a framework for temporal rule learning which capitalises on the availability of increasingly mature non-temporal rule learners. Our approach is based on the idea of splitting a temporal dataset into windows, extracting rules from each window with an off-the-shelf Datalog rule learner, and then combining the

obtained non-temporal rules into suitable temporal rules. Rules generated by our approach are expressed in DatalogMTL and are given a time-sensitive confidence score. We have implemented our approach in a system MTLearn compatible with any Datalog rule learner, as well as with a wide range of strategies for scoring the output temporal rules. The results of the evaluation on temporal link prediction tasks shows that our approach is highly competitive, achieving performance comparable to that of state-of-the-art Machine Learning models for the task, while at the same time providing interpretable results.

1.3 Outline

We introduce three kinds of practical reasoning algorithms for DatalogMTL as well as a framework dedicated to learning temporal rules written in DatalogMTL. The remainder of this thesis is organised as follows:

1. In Chapter 2, we begin by revisiting some fundamental background knowledge about Datalog, considering that DatalogMTL is an extension of Datalog. Following this, we shift our focus to DatalogMTL, introducing its syntax and semantics, along with two major reasoning problems associated with it. Then, we recapitulate different techniques available for reasoning in DatalogMTL, which we will take as a starting point for the development of our approach. On the one hand, we present a variant of the (non-terminating) naïve materialisation procedure, and prove its soundness and completeness; on the other hand, we describe existing algorithms based on exponential reductions to reasoning in linear temporal logic (LTL) and to emptiness checking of Büchi automata.

2. Major contributions of this thesis start from Chapter 3. In this chapter, we first present a *seminaïve* materialisation procedure for DatalogMTL. Then,

we present an optimised variant of our seminaïve procedure which further reduces the number of redundant computations performed during materialisation by disregarding rules during the execution of the procedure as soon as we can be certain that their application will never derive new facts in subsequent materialisation steps. Finally, in Section 3.3 we propose a sound and complete reasoning algorithm combining optimised seminaïve materialisation and our realisation of the automata-based reasoning approach of Wałęga, Cuenca Grau, Kaminski, and Kostylev (2019). Our algorithm is designed to delegate the bulk of the computation to the scalable materialisation component and resort to automata-based techniques only as needed to ensure termination and completeness.

3. In Chapter 4, we investigate a pure materialisation-based reasoning algorithm, where partial representations of the canonical interpretation are acquired through iterative rounds of rule application. We have implemented the algorithm as an extension of the MeTeoR system, and our experiments show that our approach is both scalable and robust. In particular, we can efficiently materialise recursive DatalogMTL programs consisting of complex rules and datasets including tens of millions of temporal facts.

4. In Chapter 5, we present a stream reasoning algorithm introduced by Wałęga, Kaminski, and Cuenca Grau (2019) that is applicable to forward-propagating DatalogMTL programs, in which propagation of derived information towards past time points is precluded. We have implemented this stream reasoning algorithms as an extension of MeTeoR and tested it experimentally. The obtained results support the feasibility of the approach in practice.

5. In Chapter 6, we propose a framework for temporal rule mining which capitalises on the availability of mature non-temporal rule learners. We have

implemented our approach in a system MTLearn compatible with any Datalog rule learner, as well as with a wide range of strategies for scoring the output temporal rules. The results of the evaluation on temporal link prediction tasks shows that our approach is competitive, achieving performance comparable to that of state-of-the-art Machine Learning models for the task, while at the same time providing interpretable results.

6. In Chapter 7, we delve into the related work, providing a brief overview and discussion; Chapter 8 is dedicated to concluding our findings and presenting potential avenues for interesting future research.

Apart from the content in Chapter 6, which is currently under review, most of the techniques and results in this thesis have previously been published. Theoretical proofs in Chapters 4 and 5 are available in Wałęga, Zawidzki, Wang, and Cuenca Grau (2023); Wałęga, Kaminski, Wang, and Grau (2023).

2 | Preliminaries

2.1 Datalog

Datalog is a standard rule language commonly used in Knowledge Representation (Van Harmelen, Lifschitz, and Porter, 2008), Data Management (Garcia-Molina, Ullman, and Widom, 2009; Abiteboul, Hull, and Vianu, 1995), and Logic Programming (Dantsin, Eiter, Gottlob, and Voronkov, 2001). We now recapitulate the definitions of the syntax and the semantics of Datalog.

Syntax of Datalog A *term* is a constant or a variable. An *atom* has the form $P(t_1, \ldots, t_k)$, where P is a k-ary predicate with $k \geq 0$ and each t_i, $1 \leq i \leq k$, is a term. A fact is a variable-free atom and a dataset is a finite set of facts. Datalog rules are interpreted in a standard way as universally quantified, function-free Horn formulas in First Order Logic (Abiteboul, Hull, and Vianu, 1995). Formally, a Datalog *rule* has the form

$$A' \leftarrow A_1 \wedge \ldots A_m \wedge not\ A_{m+1} \wedge \cdots \wedge\ not\ A_n \tag{2.1}$$

where $0 \leq m \leq n$ and A' and all A_i are atoms. For r a rule, $\mathbf{h}(r) = A'$ is the *head* and $\mathbf{b}(r) = \{A_1 \wedge \ldots A_m \wedge\ not\ A_{m+1} \wedge \cdots \wedge\ not\ A_n\}$ is the rule *body*. $\mathbf{b}^+(r) = \{A_1 \wedge \ldots A_m\}$ is the set of *positive body atoms* and $\mathbf{b}^-(r) = \{not\ A_{m+1} \wedge \cdots \wedge\ not\ A_n\}$ is the set of *negative body atoms*. Each rule r must be *safe* – that is, each variable occurring in r must also occur in r in at least one positive atom. A *program* is a finite set of rules. A rule is *positive* if it has no *negative body atoms*; moreover, a program is *positive* if all of its rules are *positive*.

Substitution A *substitution* σ is a mapping of finitely many variables to constants. For α a term, an atom, a rule, or a set thereof, $\sigma\alpha$ is the result of replacing each occurrence of a variable x in α with $\sigma(x)$, if the latter is defined.

Semantics of Datalog For a program Π, a rule r and a dataset $\mathcal{D}$, we define the set $\mathsf{inst}_r[G]$ of instances of r obtained by applying a rule r to G , and the set of $\Pi[\mathcal{D}]$ of facts obtained by applying a program Π to $\mathcal{D}$ as follows. Each rule instance considered in computing $\Pi[\mathcal{D}]$ is said to be *applicable* to $\mathcal{D}$.

$$\mathsf{inst}_r[\mathcal{D}] = \{r\sigma \mid \sigma \text{ is a substitution s.t. } b^+(r)\sigma \subseteq \mathcal{D} \text{ and } b^-(r)\sigma \cap \mathcal{D} = \emptyset\}$$

$$\Pi[\mathcal{D}] = \bigcup_{r \in \Pi} \{\mathbf{h}(r') \mid r' \in \mathsf{inst}_r[\mathcal{D}]\}$$

Let $\Pi^0[\mathcal{D}] = \mathcal{D}$; and let $\Pi^{i+1}[\mathcal{D}] = \Pi\big[\Pi^i[\mathcal{D}]\big]$ for $i \geq 0$. Then, the closure of Π on $\mathcal{D}$ is defined as $\Pi^\infty[\mathcal{D}] = \bigcup_{i \geq 0} \Pi^i[\mathcal{D}]$. When the closure is computed in advance and persisted, we refer to both $\Pi^\infty[\mathcal{D}]$ and the process of computing it as *materialisation* (Motik, Nenov, Piro, and Horrocks, 2019a).

Due to its ability to declaratively and succinctly specify complex problems, Datalog plays an important role in advanced applications that mix AI and Data Management techniques. For example, some commercial Datalog systems have been developed and applied in real-world applications, such as Oracle's database[1], RDFox[2], LogicBlox[3], and GraphDB[4]. In these Datalog systems, the main computational problem is answering queries over the facts that logically follow from a set of given facts and Datalog rules. A common Datalog reasoning method is often realised by precomputing and storing all consequences of a Datalog program

[1] http://docs.oracle.com/database/122/RDFRM/.
[2] https://www.oxfordsemantic.tech/rdfox
[3] http://www.logicblox.com.
[4] http://ontotext.com.

and a set of facts in a preprocessing step; this process and its output are both called materialisation. After such preprocessing, queries can be answered directly over the materialisation, disperse with the rules together. Consequently, materialisation has often become the technique of choice in Datalog systems, with a myriad of optimised techniques for calculating it proposed over the past few decades. These range from the naïve and semi-naïve algorithms (Abiteboul, Hull, and Vianu, 1995) to the more recent advancements like the modular materialisation algorithm (Hu, Motik, and Horrocks, 2022).

While the use of Datalog rules is notably effective, the conventional method of acquiring them often demands expert knowledge that is not readily available and involves substantial human effort. Consequently, an alternative research area focuses on the design and implementation of *Datalog rule learners*, which are used to extract Datalog rules from a dataset, together with a confidence value for each extracted rule. Existing rule learners exhibit significant heterogeneity due to variations in the techniques and algorithms they are based on, their underlying assumptions, and the syntactic form of the rules they generate.

For example, AnyBURL (Meilicke, Chekol, Ruffinelli, and Stuckenschmidt, 2019) and AMIE+ (Galárraga, Teflioudi, Hose, and Suchanek, 2015) are specifically developed to efficiently generate rules from large-scale Knowledge Graphs (KGs) represented in RDF format. Both systems accept a set of facts involving binary predicates as input and compute Datalog rules satisfying certain syntactic restrictions. AnyBURL uses a bottom-up approach to generate rules. Initially, the system identifies a collection of paths from the input KG; subsequently, these identified paths are generalised into rules by substituting constants with variables.

Rules obtained by AnyBURL conform to the following syntactic forms:

$$\begin{aligned} A'(c', x) &\leftarrow A_1(x, z_2) \wedge \cdots \wedge A_n(z_n, c), \\ A'(c', x) &\leftarrow A_1(x, z_2) \wedge \cdots \wedge A_n(z_n, z_{n+1}), \\ A'(y, x) &\leftarrow A_1(x, z_2) \wedge \cdots \wedge A_n(z_n, y), \end{aligned}$$

where c and c' are constants, x and y are variables occurring in both the body and the head of the corresponding rule, and each z_i is a variable occurring only in the body. Rules generated by AMIE+ are connected (i.e., the graph obtained from the rule body by interpreting each term as a node and each atom as an undirected edge is connected), thus precluding rules with completely unrelated atoms; furthermore, each variable in a rule is required to appear at least twice in different atoms. Other Datalog rule learners following similar approaches include RLvLR (Omran, Wang, and Wang, 2018) and RARL (Pirrò, 2020).

Popper (Cropper and Morel, 2021) is an inductive logic programming (ILP) system. Instead of enforcing a predetermined syntactic form for the output rules, Popper offers a declarative mechanism that allows users to introduce syntactic restrictions on the predicates and associated arities permitted in rules. This provides flexibility in specifying the syntactic structure of the generated rules according to the user's requirements and preferences.

2.2 DatalogMTL

2.2.1 Syntax of DatalogMTL

In DatalogMTL, we consider a *timeline* consisting of ordered rational numbers, denoted by $\mathbb{Q}$, which we also call *time points*. A *(rational) interval* ϱ is a set of

rational numbers that is either empty or such that

- for all $t_1, t_2, t_3 \in \mathbb{Q}$ satisfying $t_1 < t_2 < t_3$ and $t_1, t_3 \in \varrho$, it must be the case that $t_2 \in \varrho$, and
- the greatest lower bound ϱ^- and the least upper bound ϱ^+ of ϱ belong to $\mathbb{Q} \cup \{-\infty, \infty\}$.[5]

The bounds ϱ^- and ϱ^+ are called the *left* and the *right endpoints* of ϱ, respectively. An interval is *punctual* if it contains exactly one time point. Intervals ϱ_1 and ϱ_2 are *union-compatible* if $\varrho = \varrho_1 \cup \varrho_2$ is also an interval. We represent a non-empty interval ϱ using the standard notation $\langle \varrho^-, \varrho^+ \rangle$, where the *left bracket* '$\langle$' is either '[' or '(', and the *right bracket* $\rangle$ is either ']' or ')'. We assume that each rational endpoint is given as a (not necessary reduced) fraction with an integer numerator and a positive integer denominator, both encoded in binary[6]. As usual, the brackets '[' and ']' indicate that the corresponding endpoints are included in the interval, whereas '(' and ')' indicate that they are not included. Observe that every interval has multiple representations due to possibly non-reduced fractions. Each representation, however, uniquely determines an interval and so, if it is clear from the context, we will abuse notation and identify an interval representation with the unique interval it represents.

We consider a *signature* consisting of pairwise disjoint countable sets of constants, variables, and predicates with non-negative integer arities. As usual, a term is either a constant or a variable. A *relational atom* is an expression of the form $P(\mathbf{s})$, with P a predicate and $\mathbf{s}$ a tuple of terms whose length matches the arity of P. A *metric atom* is an expression given by the following grammar, where $P(\mathbf{s})$

[5] By this restriction, for example, the set of all rational numbers between 0 and π is not an interval, as it has no least upper bound in $\mathbb{Q}$. Also note that in contrast to Brandt et al. (2018), we do not require finite endpoints of intervals to be dyadic rational numbers.

[6] This is for the algorithmic complexity reason because the complexity is expressed as a function of the length of the input representation.

is a relational atom, $\top$ and $\bot$ are logical truth and falsehood respectively, and $\diamondminus$ ('sometime in the past'), $\diamondplus$ ('sometime in the future'), $\boxminus$ ('always in the past'), $\boxplus$ ('always in the future'), $\mathcal{S}$ ('since'), and $\mathcal{U}$ ('until') are MTL operators that can be indexed with any intervals ϱ containing only non-negative rationals:

$$M ::= \top \mid \bot \mid P(\mathbf{s}) \mid \diamondminus_{\varrho} M \mid \diamondplus_{\varrho} M \mid \boxminus_{\varrho} M \mid \boxplus_{\varrho} M \mid M \mathcal{S}_{\varrho} M \mid M \mathcal{U}_{\varrho} M.$$

We refer to $\diamondminus$, $\boxminus$, and $\mathcal{S}$ as *past operators* and we refer to $\diamondplus$, $\boxplus$, and $\mathcal{U}$ as *future operators*. A *rule* is an expression of the form

$$M' \leftarrow M_1 \wedge \cdots \wedge M_n, \qquad \text{for } n \geq 1, \tag{2.2}$$

with each M_i a metric atom, and M' generated by the following grammar:

$$M' ::= \bot \mid P(\mathbf{s}) \mid \boxminus_{\varrho} M' \mid \boxplus_{\varrho} M'.$$

The conjunction $M_1 \wedge \cdots \wedge M_n$ in Expression (2.2) is the rule's *body* and M' is the rule's *head*. A rule is *safe* if each variable in its head also occurs in the body, and this occurrence is not in a left operand of $\mathcal{S}$ or $\mathcal{U}$; for instance, a rule such as $P(x) \leftarrow Q(x) \mathcal{S}_{[0,0]} \top$ is not safe; in fact, it can be equivalently rewritten as $P(x) \leftarrow \top$ (see the DatalogMTL semantics in Table 2.1).

An expression is *ground* if it mentions no variables. A *fact* is an expression of the form $M@\varrho$ with M a ground relational atom and ϱ an interval; a *dataset* is a finite set of facts. The *coalescing* of facts $M@\varrho_1$ and $M@\varrho_2$ with union-compatible intervals ϱ_1 and ϱ_2 is the fact $M@(\varrho_1 \cup \varrho_2)$. The *coalescing* of a dataset $\mathcal{D}$, denoted by $\mathsf{coal}(\mathcal{D})$, is the unique dataset obtained by iteratively coalescing facts in $\mathcal{D}$ until no more facts can be coalesced. The *grounding* $\mathsf{ground}(\Pi, \mathcal{D})$ of a program Π with respect to a dataset $\mathcal{D}$ is the set of all ground rules that can be obtained by

assigning constants in Π or $\mathcal{D}$ to variables in Π.

The *dependency graph* of a program Π is the directed graph G_Π. Each predicate P in Π is considered as a node v_P in the graph, and an edge (v_Q, v_R) is created whenever there is a rule in Π mentioning Q in the body and R in the head. Program Π is *recursive* if G_Π has a cycle. A predicate P is *recursive* in Π if G_Π has a path ending in v_P and including a cycle (the path can be a self-loop); otherwise, it is *non-recursive*. A metric atom is *non-recursive* in Π if so are all its predicates; otherwise it is *recursive*. The (non-)recursive fragment of a program Π is the (unique) subset of rules in Π with (non-)recursive atoms in heads.

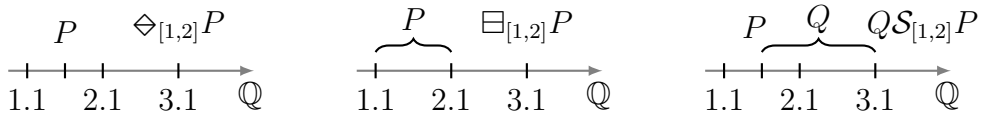
For a predicate P, a rule r is *P-relevant* in Π if there exists a rule r' in Π mentioning P or $\bot$ in the head and a path in G_Π starting from a vertex representing the predicate in the head of r and ending in a vertex representing some predicate from the body of r'.

2.2.2 Semantics of DatalogMTL

An *interpretation* $\mathfrak{I}$ is a function which assigns to each time point t a set of ground relational atoms; if an atom $P(\mathbf{c})$ belongs to this set, we say that $P(\mathbf{c})$ is *satisfied* at t in $\mathfrak{I}$ and we write $\mathfrak{I}, t \models P(\mathbf{c})$. This extends to arbitrary metric atoms as shown in Table 2.1. The meaning of past operators $\diamondminus$, $\boxminus$, and $\mathcal{S}$ is additionally visualised in figures below. The first figure shows that $\diamondminus_{[1,2]}P$ holds at 3.1 if P holds at some time point located within the interval $[1.1, 2.1]$. The second figure indicates that $\boxminus_{[1,2]}P$ holds at 3.1 if P holds continuously in the interval $[1.1, 2.1]$. The third figure, in turn, shows that $Q\mathcal{S}_{[1,2]}P$ holds at 3.1 if P holds at some time point within $[1.1, 2.1]$ and Q holds continuously from this time point up to the time point 3.1. The meaning of future operators $\diamondplus$, $\boxplus$, and $\mathcal{U}$ is symmetric to the meaning of the corresponding past operators $\diamondminus$, $\boxminus$, and $\mathcal{S}$, respectively.

Table 2.1: Semantics of ground metric atoms.

$\mathfrak{I}, t \models \top$		for each $t \in \mathbb{Q}$
$\mathfrak{I}, t \not\models \bot$		for each $t \in \mathbb{Q}$
$\mathfrak{I}, t \models \diamondminus_\varrho M$	iff	$\mathfrak{I}, t' \models M$ for some t' such that $t - t' \in \varrho$
$\mathfrak{I}, t \models \diamondplus_\varrho M$	iff	$\mathfrak{I}, t' \models M$ for some t' such that $t' - t \in \varrho$
$\mathfrak{I}, t \models \boxminus_\varrho M$	iff	$\mathfrak{I}, t' \models M$ for all t' such that $t - t' \in \varrho$
$\mathfrak{I}, t \models \boxplus_\varrho M$	iff	$\mathfrak{I}, t' \models M$ for all t' such that $t' - t \in \varrho$
$\mathfrak{I}, t \models M_1 \mathcal{S}_\varrho M_2$	iff	$\mathfrak{I}, t' \models M_2$ for some t' such that $t - t' \in \varrho$ and $\mathfrak{I}, t'' \models M_1$ for all $t'' \in (t', t)$
$\mathfrak{I}, t \models M_1 \mathcal{U}_\varrho M_2$	iff	$\mathfrak{I}, t' \models M_2$ for some t' such that $t' - t \in \varrho$ and $\mathfrak{I}, t'' \models M_1$ for all $t'' \in (t, t')$



An interpretation $\mathfrak{I}$ satisfies a fact $M@\varrho$ if $\mathfrak{I}, t \models M$, for all $t \in \varrho$. Interpretation $\mathfrak{I}$ satisfies a ground rule r if, whenever $\mathfrak{I}$ satisfies each body atom of r at a time point t, then $\mathfrak{I}$ also satisfies the head of r at t. Interpretation $\mathfrak{I}$ satisfies a (non-ground) rule r if it satisfies each ground instance of r. Interpretation $\mathfrak{I}$ is a *model* of a program Π if it satisfies each rule in Π, and it is a *model* of a dataset $\mathcal{D}$ if it satisfies each fact in $\mathcal{D}$. Each dataset $\mathcal{D}$ has a unique least model $\mathfrak{I}_\mathcal{D}$, and we say that dataset $\mathcal{D}$ *represents* interpretation $\mathfrak{I}_\mathcal{D}$. Program Π and dataset $\mathcal{D}$ are *consistent* if they have a model, and they *entail* a fact $M@\varrho$ if each model of both Π and $\mathcal{D}$ is a model of $M@\varrho$.

Consistency checking is the problem of checking whether a given program and a dataset admit a common model. *Fact entailment* is the problem of checking whether a program and a dataset entail a given relational fact. Consistency check-

ing and fact entailment in DatalogMTL reduce to the complements of each other. Specifically, to check whether a program Π and a dataset $\mathcal{D}$ are inconsistent, it suffices to check whether they entail fact $P@0$, for P a fresh proposition occurring neither in Π nor in $\mathcal{D}$. In turn, to check whether Π and $\mathcal{D}$ entail a fact $P(\mathbf{c})@\varrho$, it suffices to check whether the following program and dataset are inconsistent, with P' a fresh predicate of the same arity as P, $\mathbf{x}$ a tuple of distinct variables, and t an arbitrary time point belonging to the interval $\varrho = [t_1, t_2)$:

$$\Pi'' = \Pi \cup \{\bot \leftarrow P'(\mathbf{x}) \wedge \boxminus_{\varrho_1} P(\mathbf{x}) \wedge \boxplus_{\varrho_2} P(\mathbf{x})\}$$
$$\mathcal{D}'' = \mathcal{D} \cup \{P'(\mathbf{c})@t\}$$

where $\varrho_1 = [0, t - t_1]$ and $\varrho_2 = [0, t_2 - t)$[7].

2.2.3 Fragments of DatalogMTL

Reasoning in the full DatalogMTL is of high complexity, namely ExpSpace-complete (Brandt et al., 2018) and PSpace-complete with respect to data size (Wałęga, Cuenca Grau, Kaminski, and Kostylev, 2019), which makes reasoning in practice challenging. Therefore, recent works focus on establishing a suitable trade-off between expressive power and complexity of reasoning, by identifying lower complexity fragments of DatalogMTL (Wałęga, Cuenca Grau, Kaminski, and Kostylev, 2021; Wałęga, Zawidzki, and Cuenca Grau, 2021) as well as studying alternative semantics with more favourable computational behaviour (Wałęga, Cuenca Grau, Kaminski, and Kostylev, 2021).

We describe two fragments of DatalogMTL pertinent to this thesis. For a detailed examination of fragments of DatalogMTL under continuous semantics, we refer readers to the work by Wałęga, Cuenca Grau, Kaminski, and Kostylev (2021).

[7] If $t_2 = \infty$, then $t_2 - t$ stands for ∞.

Forward-propagating DatalogMTL programs In these programs, rules are syntactically restricted so that their application to a set of facts can derive information relevant to present and future time points, but not to past time points.

Definition 3 *A rule is* forward-propagating *if it satisfies the following restrictions:*

- *it does not mention* $\top$ *or* $\bot$*,*
- *it does not mention in the body any metric operators except* $\boxminus$ *and* $\diamondminus$*, and*
- *it does not mention in the head any metric operators except* $\boxplus$*.*

A program is forward-propagating if so is each of the rules it contains. Forward-propagating programs are always consistent with a dataset.

Definition 4 *A forward-propagating program is in* normal form *if it contains only rules of the following forms*[8]*:*

$$M' \leftarrow \diamondminus_{\varrho} M, \tag{2.3}$$

$$M' \leftarrow \boxminus_{\varrho} M, \tag{2.4}$$

$$M' \leftarrow M_1 \wedge \cdots \wedge M_n, \qquad \text{for } n \geq 1, \tag{2.5}$$

where $M, M', M_1, \ldots, M_n$ *are relational atoms and* ϱ *is a positive non-empty interval.*

Interval-bounded DatalogMTL programs Bounded DatalogMTL programs exclude the use of infinity (∞) as an endpoint within any interval. It is a natural and expressive fragment of DatalogMTL, where the complexity of reasoning remains as hard as in the unrestricted full DatalogMTL language (Wałęga, Zawidzki, and Cuenca Grau, 2021). he exclusion of ∞ provides the possibility to

[8] According to Wałęga, Kaminski, Wang, and Grau (2023), each forward-propagating program Π can be transformed in polynomial time into a program Π' in normal form.

identify the definite depth of a rule which determines the time points that can be 'affected' by an application of this rule. As we will show in Chapter 4, within the context of DatalogMTL with bounded intervals, we are able to obtain the periodic structure of a canonical interpretation based on a partial materialisation, instead of executing a potentially infinite number of materialization steps.

2.2.4 Canonical Interpretation

Each pair of a consistent program Π and a dataset $\mathcal{D}$ admits a unique least model, which we refer to as their *canonical interpretation* $\mathfrak{C}_{\Pi,\mathcal{D}}$ (Brandt et al., 2018). As in plain Datalog, we can construct this interpretation by applying rules of Π to $\mathcal{D}$ in a forward-chaining manner until a fixpoint is reached. Unlike plain Datalog, however, the fixpoint in DatalogMTL may only be reachable after infinitely many materialisation steps; for example, given a fact $P@0$, the rule $\boxplus_1 P \leftarrow P$ propagates P to all positive integers, requiring ω materialisation steps.

Materialisation is performed using the *immediate consequence operator* T_Π; intuitively, $T_\Pi(\mathfrak{I})$ can be seen as the interpretation obtained from $\mathfrak{I}$ by adding all relational facts that can be derived by applying once all rules in Π which are not $\bot$-rules[9] in all possible ways to $\mathfrak{I}$. Formally, we define T_Π, for a program Π, as the operator mapping each interpretation $\mathfrak{I}$ to the least interpretation containing $\mathfrak{I}$ and satisfying the following property for each ground instance r of a rule in Π that is not a $\bot$-rule: whenever $\mathfrak{I}$ satisfies each body atom of r at a time point t, then $T_\Pi(\mathfrak{I})$ satisfies the head of r at t. Subsequent applications of T_Π to the (least)

[9] $\bot$-rules refers to rules whose head is bottom.

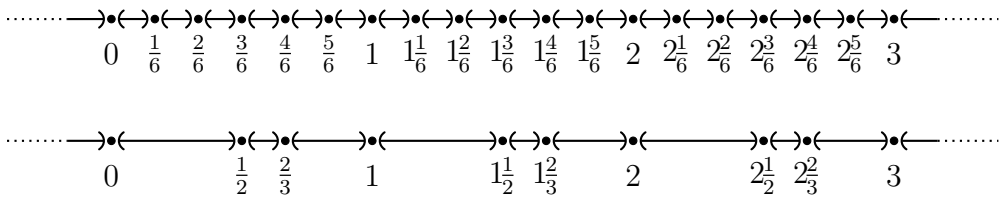
model $\mathfrak{I}_\mathcal{D}$ of $\mathcal{D}$ define the following sequence of interpretations, for all ordinals α:

$$
\begin{aligned}
T^0_\Pi(\mathfrak{I}_\mathcal{D}) &= \mathfrak{I}_\mathcal{D}, \\
T^\alpha_\Pi(\mathfrak{I}_\mathcal{D}) &= T_\Pi\left(T^{\alpha-1}_\Pi(\mathfrak{I}_\mathcal{D})\right), && \text{for } \alpha \text{ a successor ordinal,} \\
T^\alpha_\Pi(\mathfrak{I}_\mathcal{D}) &= \bigcup_{\beta<\alpha} T^\beta_\Pi(\mathfrak{I}_\mathcal{D}), && \text{for } \alpha \text{ a limit ordinal.}
\end{aligned}
$$

The canonical interpretation is obtained after at most ω_1 (i.e., the smallest ordinal number that, considered as a set, is uncountable) applications of T_Π; that is, $\mathfrak{C}_{\Pi,\mathcal{D}} = T^{\omega_1}_\Pi(\mathfrak{I}_\mathcal{D})$ (Brandt, Kontchakov, Ryzhikov, Xiao, and Zakharyaschev, 2017); in turn, if Π and $\mathcal{D}$ are consistent then $\mathfrak{C}_{\Pi,\mathcal{D}}$ is the least common model of Π and $\mathcal{D}$. Furthermore, $\bot$-rules can be treated as constraints, and so Π and $\mathcal{D}$ are consistent if and only if $\mathfrak{C}_{\Pi,\mathcal{D}}$ is a model of all $\bot$-rules in Π.

Although the timeline in DatalogMTL is dense, it can be divided into regularly distributed intervals which are uniform in the sense that, in the canonical interpretation, the same relational atoms hold in all time points belonging to the same interval. This observation was first exploited to partition the rational timeline, for a program Π and dataset $\mathcal{D}$, into punctual intervals $[i \cdot d, i \cdot d]$ and open intervals $((i-1)\cdot d, i\cdot d)$, for each $i \in \mathbb{Z}$, where d is the greatest common divisor (gcd) of the numbers occurring as interval endpoints in Π and $\mathcal{D}$ (Brandt et al., 2018). Later, an alternative partitioning of the timeline was proposed (Wałęga, Cuenca Grau, Kaminski, and Kostylev, 2019), where punctual intervals of the form $[i \cdot d, i \cdot d]$ are replaced with punctual intervals $[t + i \cdot d', t + i \cdot d']$, for all rational numbers t in $\mathcal{D}$, $i \in \mathbb{Z}$, and d' the gcd of numbers occurring in Π; in turn, open intervals of the form $((i-1) \cdot d, i \cdot d)$ were replaced with open intervals located between the new punctual intervals. Below we present example partitionings of the timeline into intervals stemming from both discretisation methods, for the case where the only rationals occurring in $\mathcal{D}$ are $\frac{1}{2}$ and $\frac{2}{3}$ and the gcd of Π is 1 (therefore, $d = \frac{1}{6}$

and $d' = 1$). The second partitioning has the advantage that the gcd is computed independently from $\mathcal{D}$; this was exploited to devise reasoning techniques with a better computational behaviour in data size.



2.3 Reasoning Techniques for DatalogMTL

We recapitulate different techniques for reasoning in DatalogMTL, which we take as a starting point for the development of our approach. In Section 2.3.1, we present a variant of the naïve materialisation procedure (Wałęga, Zawidzki, and Cuenca Grau, 2023; Wałęga, Zawidzki, and Cuenca Grau, 2021) where the k-th iteration of its main loop, for any $k \in \mathbb{N}$, yields a dataset representing the the k-th application of the immediate consequence operator of the input program to the input dataset. In Section 2.3.2, we discuss the reduction of DatalogMTL reasoning to LTL satisfiability proposed by Brandt et al. (2018). In Section 2.3.3 we describe the automata-based procedure exploited by Wałęga, Cuenca Grau, Kaminski, and Kostylev (2019) to establish optimal complexity bounds for reasoning.

2.3.1 Naïve Materialisation

The fixpoint characterisation from Section 2.2.4 suggests the naïve materialisation-based approach specified in Procedure 1. For a program Π, a dataset $\mathcal{D}$, and a fact $P(\mathbf{c})@\varrho$, the procedure applies the immediate consequence operator T_Π to $\mathfrak{I}_\mathcal{D}$ (Lines 3, 4, and 8) until one of the following holds:

Procedure 1: Naïve($\Pi, \mathcal{D}, P(\mathbf{c})@\varrho$)

Input: A program Π, a dataset $\mathcal{D}$, and a fact $P(\mathbf{c})@\varrho$
Output: Either inconst, or a pair of a truth value and a dataset

1 Initialise $\mathcal{N}$ to $\emptyset$ and $\mathcal{D}'$ to $\mathcal{D}$;
2 **loop**
3 $\quad \mathcal{N} \coloneqq \Pi[\mathcal{D}']$; // derive new facts
4 $\quad \mathcal{C} \coloneqq \mathsf{coal}(\mathcal{D}' \cup \mathcal{N})$; // add new facts to partial materialisation
5 $\quad$ **if** $\mathcal{D}'$ satisfies the body of some $\bot$-rule in Π **then return** inconst; // inconsistency is derived
6 $\quad$ **if** $\mathcal{D}' \models P(\mathbf{c})@\varrho$ **then return** $(\mathsf{true}, \mathcal{D}')$; // input fact is derived
7 $\quad$ **if** $\mathcal{C} = \mathcal{D}'$ **then return** $(\mathsf{false}, \mathcal{D}')$; // fixpoint is reached
8 $\quad \mathcal{D}' \coloneqq \mathcal{C}$;

- the partial materialisation of Π and $\mathcal{D}$ yields inconsistency due to satisfaction of the body of some $\bot$-rule, in which case inconst is returned (Line 5),
- the partial materialisation entails $P(\mathbf{c})@\varrho$, in which case the truth value true and the obtained partial materialisation are returned (Line 6), or
- the application of T_Π reaches a fixpoint and the obtained materialisation does not satisfy $P(\mathbf{c})@\varrho$ nor yields an inconsistency, in which case the value false and the full materialisation are returned (Line 8).

Example 5 *As an example, consider the input fact* $R_1(c_1, c_2)@[4, 4]$*, the dataset*

$$\mathcal{D}_{\mathsf{ex}} = \{R_1(c_1, c_2)@[0, 1],\ \ R_2(c_1, c_2)@[1, 2],\ \ R_3(c_2, c_3)@[2, 3],\ \ R_5(c_2)@[0, 1]\},$$

and the program Π_{ex} *consisting of the following rules:*

$$R_1(x,y) \leftarrow \diamondminus_{[1,1]} R_1(x,y), \tag{r_1}$$

$$\boxplus_{[1,1]} R_5(y) \leftarrow R_2(x,y) \wedge \boxplus_{[1,2]} R_3(y,z), \tag{r_2}$$

$$R_4(x) \leftarrow \diamondminus_{[0,1]} R_5(x), \tag{r_3}$$

$$R_6(y) \leftarrow R_1(x,y) \wedge \boxminus_{[0,2]} R_4(y) \wedge R_5(y). \tag{r_4}$$

In naïve materialisation, rules are applied by first identifying the facts that can ground the rule body, and then determining the maximal intervals for which all the ground body atoms hold simultaneously. For instance, Rule r_2 in Example 5 is applied to $\mathcal{D}_{\mathsf{ex}}$ by matching $R_2(c_1, c_2)$ and $R_3(c_2, c_3)$ to the rule's body atoms, and determining that $[1, 1]$ is the maximal interval in which the body atoms $R_2(c_1, c_2)$ and $\boxplus_{[1,2]} R_3(c_2, c_3)$ hold together. As a result, the head $\boxplus_{[1,1]} R_5(c_2)@[1, 1]$ holds, and so the fact $R_5(c_2)@[2, 2]$ is derived. We formalise it below.

Definition 6 *Let* r *be a rule of the form* $M' \leftarrow M_1 \wedge \cdots \wedge M_n$ *for some* $n \geq 1$*, which is not a* $\bot$*-rule, and let* $\mathcal{D}$ *be a dataset. The set of* instances *for* r *and* $\mathcal{D}$ *is defined as follows:*

$$\mathsf{inst}_r[\mathcal{D}] = \big\{(M_1\sigma@\varrho_1, \ldots, M_n\sigma@\varrho_n) \mid \sigma \text{ is a substitution and, for each } i \in \{1, \ldots, n\} \ \varrho_i \text{ is a subset-maximal interval such that } \mathcal{D} \models M_i\sigma@\varrho_i\big\}.$$

The set $r[\mathcal{D}]$ *of facts* derived *by* r *from* $\mathcal{D}$ *is defined as follows:*

$$r[\mathcal{D}] = \{M\sigma@\varrho \mid \sigma \text{ is a substitution, } M \text{ is the relational atom in } M'\sigma, \text{ and there is } (M_1\sigma@\varrho_1, \ldots, M_n\sigma@\varrho_n) \in \mathsf{inst}_r[\mathcal{D}] \text{ such that } \varrho \text{ is the unique subset-maximal interval satisfying } M'\sigma@(\varrho_1 \cap \ldots \cap \varrho_n) \models M\sigma@\varrho\}. \tag{2.6}$$

The set of facts derived *from* $\mathcal{D}$ *by one-step application of* Π *is*

$$\Pi[\mathcal{D}] = \bigcup_{r \in \Pi} r[\mathcal{D}]. \tag{2.7}$$

Notice that, in the definition above, to determine instances for rules and facts derived by rules, one needs to compute subset-maximal intervals consisting of time points satisfying a particular property. Formally, given a set of intervals $S = \{\varrho_1, \varrho_2, \ldots, \varrho_n\}$ where each interval $\varrho_i = [a_i, b_i]$. An interval $\varrho_i = [a_i, b_i]$ is subset-maximal if there is no interval $\varrho_j = [a_j, b_j]$ in S such that $a_j \leq a_i$ and $b_i \leq b_j$ with $\varrho_i \neq \varrho_j$. Such computations are straight forward, for example, if a particular property holds in the time point 1, in all time points between 2 (inclusive) and 3 (exclusive), and for all time points bigger than 5.5, then the subset-maximal intervals consisting of time points satisfying this property are $[1, 1]$, $[2, 3)$, and $(5, \infty)$.

By Definition 6, the set $\mathcal{N}$ of facts derived by Procedure 1 in Line 3 in the first iteration of the loop on our running example is $\Pi_{\mathsf{ex}}[\mathcal{D}_{\mathsf{ex}}] = \{R_1(c_1, c_2)@[1, 2]$, $R_4(c_2)@[0, 2], R_5(c_2)@[2, 2]\}$. The partial materialisation $\mathcal{D}^1_{\mathsf{ex}}$ that will be passed to the next materialisation step is obtained in Line 4 as $\mathcal{D}^1_{\mathsf{ex}} = \mathsf{coal}(\mathcal{D}_{\mathsf{ex}} \cup \Pi_{\mathsf{ex}}[\mathcal{D}_{\mathsf{ex}}])$ where, as we described in Section 2.2.1, $\mathsf{coal}(\mathcal{D})$ is the unique dataset obtained by exhaustively coalescing facts in $\mathcal{D}$. In our example, $R_1(c_1, c_2)@[0, 1]$ and $R_1(c_1, c_2)@[1, 2]$ will be coalesced into $R_1(c_1, c_2)@[0, 2]$. Thus,

$$\begin{aligned}\mathcal{D}^1_{\mathsf{ex}} = \{&R_1(c_1, c_2)@[0, 2], \quad R_2(c_1, c_2)@[1, 2], \quad R_3(c_2, c_3)@[2, 3],\\ &R_4(c_2)@[0, 2], \quad R_5(c_2)@[0, 1], \quad R_5(c_2)@[2, 2]\}.\end{aligned}$$

In the second round, rules are applied to $\mathcal{D}^1_{\mathsf{ex}}$. The application of r_1 derives $R_1(c_1, c_2)@[1, 3]$ (from $R_1(c_1, c_2)@[0, 2]$) and the application of r_2 rederives a

redundant fact $R_5(c_2)@[2,2]$. In contrast to the previous step, Rule r_4 can now be applied to derive the new fact $R_6(c_2)@[2,2]$. Finally, r_3 derives the new fact $R_4(c_2)@[2,3]$ and rederives $R_4(c_2)@[0,2]$. After coalescing, the second step of materialisation yields the following partial materialisation:

$$\mathcal{D}^2_{\mathsf{ex}} = \{R_1(c_1,c_2)@[0,3], \quad R_2(c_1,c_2)@[1,2], \quad R_3(c_2,c_3)@[2,3], \\ R_4(c_2)@[0,3], \quad R_5(c_2)@[0,1], \quad R_5(c_2)@[2,2], \quad R_6(c_2)@[2,2]\}.$$

In the third materialisation step, rules are applied to $\mathcal{D}^2_{\mathsf{ex}}$, and derive the new fact $R_1(c_1,c_2)@[1,4]$ using Rule r_1, as well as all facts which were already derived in the second materialisation step (with the only exception of $R_1(c_1,c_2)@[1,3]$). After coalescing we obtain:

$$\mathcal{D}^3_{\mathsf{ex}} = \{R_1(c_1,c_2)@[0,4], \quad R_2(c_1,c_2)@[1,2], \quad R_3(c_2,c_3)@[2,3], \\ R_4(c_2)@[0,3], \quad R_5(c_2)@[0,1], \quad R_5(c_2)@[2,2], \quad R_6(c_2)@[2,2]\}.$$

At this point, the algorithm detects that $\mathcal{D}^3_{\mathsf{ex}}$ entails the input fact $R_1(c_1,c_2)@[4,4]$ and stops.

Procedure 1 is both sound and complete. To show this, for each $k \in \mathbb{N}$, let $\mathcal{N}_k$ and $\mathcal{D}_k$ denote the contents of, respectively, $\mathcal{N}$ and $\mathcal{D}$ in Procedure 1 upon the completion of the kth iteration of the loop. Then we prove soundness by showing inductively on $k \in \mathbb{N}$, that $\mathfrak{I}_{\mathcal{D}_k} \subseteq T^k_\Pi(\mathfrak{I}_\mathcal{D})$.

Theorem 7 (Soundness) *Consider Procedure 1 running on input Π and $\mathcal{D}$. Upon the completion of the kth (for some $k \in \mathbb{N}$) iteration of the loop of Procedure 1, it holds that $\mathfrak{I}_{\mathcal{D}'} \subseteq T^k_\Pi(\mathfrak{I}_\mathcal{D})$.*

Proof For each $k \in \mathbb{N}$, we let $\mathcal{N}_k$ and $\mathcal{D}_k$ denote the contents of, repsectively, $\mathcal{N}$ and $\mathcal{D}$ in Procedure 1 upon the completion of the kth iteration of the loop. Thus, it suffices to show, inductively on $k \in \mathbb{N}$, that $\mathfrak{I}_{\mathcal{D}_k} \subseteq T^k_\Pi(\mathfrak{I}_\mathcal{D})$.

In the base case, we have $\mathcal{D}_0 = \mathcal{D}$. Moreover, $T^0_\Pi(\mathfrak{I}_\mathcal{D}) = \mathfrak{I}_\mathcal{D}$, and so, $\mathfrak{I}_{\mathcal{D}_0} \subseteq T^0_\Pi(\mathfrak{I}_\mathcal{D})$, as required. For the inductive step, we assume that $\mathfrak{I}_{\mathcal{D}_k} \subseteq T^k_\Pi(\mathfrak{I}_\mathcal{D})$, for some $k \in \mathbb{N}$, and that the procedure enters the $k+1$st iteration of the loop. If the $k+1$st iteration of the loop breaks in Line 7, then $\mathcal{D}_{k+1} = \mathcal{D}_k$. By the inductive assumption we have $\mathfrak{I}_{\mathcal{D}_k} \subseteq T^k_\Pi(\mathfrak{I}_\mathcal{D})$, and therefore, $\mathfrak{I}_{\mathcal{D}_{k+1}} \subseteq T^k_\Pi(\mathfrak{I}_\mathcal{D})$, and thus, $\mathfrak{I}_{\mathcal{D}_{k+1}} \subseteq T^{k+1}_\Pi(\mathfrak{I}_\mathcal{D})$. Now, assume that the $k+1$st iteration of the loop does not break in Line 7. Hence, by Lines 4 and 8, we have $\mathcal{D}_{k+1} = \mathsf{coal}(\mathcal{D}_k \cup \mathcal{N}_{k+1})$. To show that $\mathfrak{I}_{\mathcal{D}_{k+1}} \subseteq T^{k+1}_\Pi(\mathfrak{I}_\mathcal{D})$, we assume that $\mathfrak{I}_{\mathcal{D}_{k+1}} \models M@t$, for some relational fact $M@t$. Hence, we have $\mathcal{D}_k \models M@t$ or $\mathcal{N}_{k+1} \models M@t$.

- Case 1: $\mathcal{D}_k \models M@t$. By the inductive assumption, $\mathfrak{I}_{\mathcal{D}_k} \subseteq T^k_\Pi(\mathfrak{I}_\mathcal{D})$, so $T^k_\Pi(\mathfrak{I}_\mathcal{D}) \models M@t$. Clearly, $T^k_\Pi(\mathfrak{I}_\mathcal{D}) \subseteq T^{k+1}_\Pi(\mathfrak{I}_\mathcal{D})$, and so, $T^{k+1}_\Pi(\mathfrak{I}_\mathcal{D}) \models M@t$.

- Case 2: $\mathcal{N}_{k+1} \models M@t$. By Line 3 of Procedure 1, we obtain that $\mathcal{N}_{k+1} = \Pi(\mathcal{D}_k)$. Thus, by Expression (2.7) from Definition 6, there exists a rule $r \in \Pi$, say of the form $M' \leftarrow M_1 \wedge \cdots \wedge M_n$, such that $r[\mathcal{D}_k] \models M@t$. Thus, by Expression (2.6) from Definition 6, there are a substitution σ and some intervals $\varrho_1, \ldots, \varrho_n$ such that $(M_1\sigma@\varrho_1, \ldots, M_n\sigma@\varrho_n) \in \mathsf{inst}_r[D_k]$ and $M'\sigma@(\varrho_1 \cap \cdots \cap \varrho_n) \models M@t$. Since $(M_1\sigma@\varrho_1, \ldots, M_n\sigma@\varrho_n)$ belongs to $\mathsf{inst}_r[\mathcal{D}_k]$, by Expression (2.6) from Definition 6, we obtain $\mathcal{D}_k \models M_i@\varrho_i$, for each $i \in \{1, \ldots, n\}$. Therefore, by the definition of T_Π, we obtain that $T_\Pi(\mathfrak{I}_{\mathcal{D}_k}) \models M'\sigma@(\varrho_1 \cap \cdots \cap \varrho_n)$ and so $T_\Pi(\mathfrak{I}_{\mathcal{D}_k}) \models M@t$. Finally, by the inductive assumption, $\mathfrak{I}_{\mathcal{D}_k} \subseteq T^k_\Pi(\mathfrak{I}_\mathcal{D})$, so $T_\Pi(\mathfrak{I}_{\mathcal{D}_k}) \subseteq T^{k+1}_\Pi(\mathfrak{I}_\mathcal{D})$, and therefore we obtain that $T^{k+1}_\Pi(\mathfrak{I}_\mathcal{D}) \models M@t$.

□

By Theorem 7, if the least model $\mathfrak{I}_{\mathcal{D}'}$ of $\mathcal{D}'$ is not a model of a $\bot$-rule in Π, then the canonical interpretation $\mathfrak{C}_{\Pi,\mathcal{D}}$ is also not a model of such a rule; thus, if the algorithm returns inconst, then Π and $\mathcal{D}$ are inconsistent. Next, to show completeness, we prove inductively that $T^k_\Pi(\mathfrak{I}_\mathcal{D}) \subseteq \mathfrak{I}_{\mathcal{D}_k}$.

Theorem 8 (Completeness) *Consider Procedure 1 running on input Π and $\mathcal{D}$. For each $k \in \mathbb{N}$, upon the completion of the kth iteration of the loop of Procedure 1, it holds that $T^k_\Pi(\mathfrak{I}_\mathcal{D}) \subseteq \mathfrak{I}_{\mathcal{D}'}$.*

Proof We use the same symbols $\mathcal{N}_k$ and $\mathcal{D}_k$ as in the proof of Theorem 7 and we define Δ_k analogously. We will show, inductively on natural numbers k, that $T^k_\Pi(\mathfrak{I}_\mathcal{D}) \subseteq \mathfrak{I}_{\mathcal{D}_k}$. The base case holds trivially, because we have $\mathcal{D}_0 = \mathcal{D}$, and so, $T^0_\Pi(\mathfrak{I}_\mathcal{D}) = \mathfrak{I}_{\mathcal{D}_0}$. For the inductive step assume that $T^k_\Pi(\mathfrak{I}_\mathcal{D}) \subseteq \mathfrak{I}_{\mathcal{D}_k}$, for some $k \in \mathbb{N}$. Moreover, let $T^{k+1}_\Pi(\mathfrak{I}_\mathcal{D}) \models M@t$, for some relational fact $M@t$. We will show that $\mathfrak{I}_{\mathcal{D}_{k+1}} \models M@t$. We have either $T^k_\Pi(\mathfrak{I}_\mathcal{D}) \models M@t$ or $T^k_\Pi(\mathfrak{I}_\mathcal{D}) \not\models M@t$.

- Case 1: $T^k_\Pi(\mathfrak{I}_\mathcal{D}) \models M@t$. Then, by the inductive assumption, $\mathfrak{I}_{\mathcal{D}_k} \models M@t$. Since $\mathcal{D}_{k+1} \models \mathcal{D}_k$, we obtain that $\mathfrak{I}_{\mathcal{D}_{k+1}} \models M@t$.
- Case 2: $T^k_\Pi(\mathfrak{I}_\mathcal{D}) \not\models M@t$. Since $T^{k+1}_\Pi(\mathfrak{I}_\mathcal{D}) \models M@t$, there exist a rule $r \in \Pi$, say of the form $M' \leftarrow M_1 \wedge \cdots \wedge M_n$, and a time point t' such that an application of r at t' yields $M@t$. More precisely, it means that there is a substitution σ such that $T^k_\Pi(\mathfrak{I}_\mathcal{D}) \models M_i\sigma@t'$, for each $i \in \{1, \ldots, n\}$, and $M'\sigma@t' \models M@t$. Therefore, by the fact that $T^k_\Pi(\mathfrak{I}_\mathcal{D}) \subseteq \mathfrak{I}_{\mathcal{D}_k}$ and by Expression (2.6) from Definition 6, there need to exist some intervals $\varrho_1, \ldots, \varrho_n$ such that $t' \in \varrho_1 \cap \cdots \cap \varrho_n$ and $(M_1\sigma@\varrho_1, \cdots, M_n\sigma@\varrho_n) \in \mathsf{inst}_r[\mathcal{D}_k]$. Thus, as $M'\sigma@t' \models M@t$, we obtain by Definition 6 that $r[\mathcal{D}_k] \models M@t$, and so, $\Pi[\mathcal{D}_k] \models M@t$. Finally, recall that $\mathcal{D}_{k+1} = \mathsf{coal}(\mathcal{D}_k \cup \mathcal{N}_{k+1})$

and $\mathcal{N}_{k+1} = \Pi[\mathcal{D}_k]$. Therefore, the fact that $\Pi[\mathcal{D}_k] \models M@t$ implies that $\mathfrak{I}_{\mathcal{D}_{k+1}} \models M@t$.

□

If Π and $\mathcal{D}$ are inconsistent, the canonical intepretation $\mathfrak{C}_{\Pi,\mathcal{D}}$ is not a model of some $\bot$-rule r in Π; thus, there is an ordinal α such that $T^{\alpha}_{\Pi}(\mathfrak{I}_{\mathcal{D}})$ is also not a model of r, and Theorem 8 ensures that the algorithm returns inconst, as required.

Furthermore, if a fixpoint is reached without encountering an inconsistency, then our procedure ensures that the input fact is not entailed and also that we have obtained a representation of the full canonical interpretation which can be kept in memory and subsequently exploited for checking entailment of any other fact. The procedure is, however, not always terminating as reaching a fixpoint may require infinitely many rounds of rule applications.

2.3.2 Translation to LTL

The discretisation of the rational timeline described in Section 2.2.4 was exploited by Brandt et al. (2018) to reduce reasoning in DatalogMTL to reasoning in LTL. The reduction transforms a program Π and a dataset $\mathcal{D}$ into an LTL formula $\varphi_{\Pi,\mathcal{D}}$ such that Π and $\mathcal{D}$ are consistent if and only if $\varphi_{\Pi,\mathcal{D}}$ is LTL-satisfiable. Here, LTL is a propositional modal logic interpreted over the integer time points and equipped with temporal operators $\bigcirc_P$ for *at the previous time point*, $\Box_P$ for *always in the past*, $\mathcal{S}$ for *since*, $\bigcirc_F$ for *at the next time point*, $\Box_F$ for *always in the future*, and $\mathcal{U}$ for *until*. An LTL formula φ is *satisfiable* if it holds at time point 0 in some LTL model.

Since, in contrast to DatalogMTL, the language of LTL is propositional, the first step in the translation is to ground Π with all constants occurring in Π or $\mathcal{D}$. Then, every relational atom $P(\mathbf{c})$ occurring in the grounding of Π with constants from

Π and $\mathcal{D}$ is translated into a propositional symbol $P^{\mathbf{c}}$. Moreover, the MTL operators occurring in Π are rewritten using LTL operators. Both the grounding of the input program Π and the translation of the MTL operators (with binary-encoded numbers) into sequences of LTL operators yield an exponential blow-up. For instance, assume that Π mentions an atom $\boxminus_{[0,60)} A(x)$, both Π and $\mathcal{D}$ mention constants $c_1, \ldots, c_n$, and that interval $(0, 60)$ contains m intervals after discretising the timeline. Then $\boxplus_{(0,60)} A(x)$ is translated to an LTL-formula containing n conjuncts (one conjunct for each c_i), each of the form $\bigcirc_F A^{c_i} \wedge \bigcirc_F \bigcirc_F A^{c_i} \wedge \cdots \wedge \underbrace{\bigcirc_F \cdots \bigcirc_F}_{m} A^{c_i}$. Consequently, $\varphi_{\Pi,\mathcal{D}}$ is exponentially large. Since satisfiability checking in LTL is PSpace-complete, this approach provides a (worst-case optimal) ExpSpace reasoning procedure for DatalogMTL.

2.3.3 Automata-based Reasoning

An alternative approach to exploit the time discretisation of DatalogMTL is to directly apply automata-based techniques, without the need of constructing an LTL-formula. Such automata-based constructions are well-studied in the context of LTL, where checking satisfiability of a formula φ reduces to checking non-emptiness of a generalised non-deterministic Büchi automaton where states are sets of formulas relevant to φ, the alphabet consists of sets of propositions, and the transition relation and accepting conditions ensure that words accepted by the automaton are exactly the models of φ (Baier and Katoen, 2008).

This technique can be adapted to the DatalogMTL setting (Wałęga, Cuenca Grau, Kaminski, and Kostylev, 2019); each state now represents ground metric atoms holding at all time points within a fragment of the timeline called a *window*.

Definition 9 *Let ϱ be an interval with endpoints on the* ruler [10]. *A* window *over ϱ*

[10] The *ruler* (for Π and $\mathcal{D}$) is the set of time points of the form $x + n \cdot \gcd(\Pi)$ for x a rational

is a set of W *of generalised facts* $A@\varrho'$ *with* $A \in \mathit{glit}(\Pi, \mathcal{D})$ [11] *and ruler-intervals* $\varrho' \subseteq \varrho$ *for which there exists a ruler-interpretation* $\mathfrak{I}$*, called corresponding* corresponding *to* W*, such that for all ruler-intervals* $\varrho' \subseteq \varrho$ *and* $A \in \mathit{glit}(\Pi, \mathcal{D})$*, we have* $\mathfrak{I} \models A@\varrho'$ *if and only if* $A@\varrho' \in W$*.*

Since the timeline can be discretised in DatalogMTL, each window can be finitely represented as a sequence consisting of sets of metric atoms which hold in the consecutive intervals from the window. Additionally, for such a sequence to be a state of the automaton, it is required that the involved metric atoms are *locally consistent*; for example, if $\boxplus_{[0,\infty)}A$—which states that A holds always in the future—holds in some interval ϱ, then $\boxplus_{[0,\infty)}A$ needs to hold also in all the intervals in the window which are to the right of ϱ in the timeline. The remaining components of the automaton are defined analogously to the case of LTL.

More specifically, given DatalogMTL program Π and dataset $\mathcal{D}$, the approach consists in constructing two generalised Büchi automata, which are responsible for recognising fragments of models of Π and $\mathcal{D}$ that are located on the timeline to the left and to the right of $\mathcal{D}$, respectively. Thus, if both of these automata have non-empty languages, there exists a common model of Π and $\mathcal{D}$, that is, Π and $\mathcal{D}$ are consistent. These automata have the same states, alphabet, and initial states, but they differ on the transition function and accepting conditions (the latter two components are symmetric for the left- and right-automata). In particular, states Q of these automata are windows (of the length computed based on the numbers occurring in the temporal operators of Π) and alphabet Σ consists of all sets of metric atoms which are relevant for Π and $\mathcal{D}$ (i.e., use predicates and

in $\mathcal{D}$ (or 0 if $\mathcal{D} =$) and n an integer. The greatest common divisor $\gcd(\Pi)$ of a program Π is the greatest rational number which divides all rational numbers which are endpoints of interval in Π to integer values (if Π has no numbers, we take $\gcd(\Pi) = 1$ for definiteness).

[11] $\mathrm{glit}(\Pi, \mathcal{D})$ is the set of ground atoms consisting $\top$, all groundings of literals in Π by constants in Π and $\mathcal{D}$, and atoms $\boxminus_{[0,+\infty)}\alpha$, $\boxminus_{(0,+\infty)}\alpha$, $\boxplus_{[0,+\infty)}\alpha$ and $\boxplus_{(0,+\infty)}\alpha$, for α an atom in $\mathcal{D}$.

constants occurring in Π and $\mathcal{D}$). The transition function δ is a partial function from $Q \times \Sigma$ to Q. For the right-moving automaton δ takes as an input a current window W and an alphabet symbol σ (i.e. a set of metric atoms), and returns the next window obtained by shifting W by one interval on the discritised time-line to the right (see Section 2.2.4 for the description of time discretisatrion) and putting in this new interval all atoms from σ. For the left-moving automaton, δ is defined similarly but the new windows is shifted to the left. Accepting conditions for the right-moving automaton capture the meaning of 'future' operators $\boxplus$ and $\mathcal{U}$, whereas for the left-moving automaton the accepting conditions capture the meaning of 'past' operators $\boxminus$ and $\mathcal{S}$. In particular, if $\boxplus_\varrho A$ occurs in some window, the automaton needs to guarantee that both A and $\boxplus_\varrho A$ will occur infinitely often successive states in the run. If $A\mathcal{U}_\varrho B$ occurs in a window, the automaton needs to guarantee that B will occur in future (in distance bounded by ϱ) and that A occurs everywhere up to this occurrence of this B.

In conclusion, consistency checking in DatalogMTL reduces to checking non-emptiness of (pairs of) automata. Importantly, this reduction provides an optimal PSpace upper bound for data complexity (Wałęga, Cuenca Grau, Kaminski, and Kostylev, 2019). In particular, automata states are polynomially large in the size of the dataset since windows can be chosen so that the number of intervals in each window is polynomially large. Moreover, the number of ground metric atoms that can hold in each of these intervals is also polynomially bounded. Thus, each state is polynomially representable and non-emptiness of the automata can be checked with the standard 'on-the-fly' approach (Baier and Katoen, 2008) in PSpace.

3 | Combining Materialisation and Automata in Full DatalogMTL

In this chapter, we present a scalable approach for deciding both consistency and fact entailment in full DatalogMTL. Contributions of our approach are as follows:

- an optimised materialisation procedure using a seminaïve strategy which efficiently applies the immediate consequence operator while minimising repeated inferences,
- a realisation of the automata-based reasoning approach, and
- an effective way of combining materialisation with automata-based reasoning aiming at using materialisation-based reasoning whenever possible.

In the remainder of this chapter we discuss each of the key components of our approach in detail. For convenience of presentation, and without loss of generality, we will assume that input programs do not contain rules whose body is vacuously satisfied (i.e., satisfied in the empty interpretation). Observe that each such rule can be formulated as a fact and added to the input dataset instead; for example, rule $P \leftarrow \top$ is equivalent to a fact $P@(-\infty, \infty)$.

3.1 Seminaïve Materialisation

We present a seminaïve materialisation procedure for DatalogMTL. Analogously to the classical seminaïve algorithm for plain Datalog (Abiteboul, Hull, and Vianu, 1995), the main idea behind our procedure is to keep track of newly derived facts in each materialisation step by storing them in a set Δ, and to ensure that each rule application in the following materialisation step involves at least one fact in

Δ. In this way, the procedure considers each instance (for a rule and dataset, as introduced in Definition 6) *at most once* throughout its entire execution, and so, such a procedure is said to enjoy the *non-repetition* property. The same fact, however, can still be derived multiple times by *different* instances; this redundancy is difficult to prevent and is not addressed by the standard seminaïve strategy.

We aim to lift the seminaïve rule evaluation strategy to the setting of DatalogMTL. As we have seen in Section 2.3.1, a rule instance can be considered multiple times in the naïve materialisation procedure; for example, $(\diamondminus_{[0,1]} R_5(c_2)@[0,2])$ of Rule r_3 in Example 5 is considered by the naïve materialisation procedure in Section 2.3.1 both in the first and second materialisation steps to derive $R_4(c_2)@[0,2]$ since the naïve procedure cannot detect that the fact $R_5(c_2)@[0,1]$ used to instantiate r_3 in the second step had previously been used to instantiate r_2.

Challenges Preventing such redundant computations, however, involves certain challenges. First, by including in Δ just newly derived facts as in Datalog, we may overlook information obtained by coalescing newly derived facts with previously derived ones. Second, restricting application to relevant rule instances requires consideration of the semantics of metric operators in rule bodies.

Procedure 2 extends the seminaïve strategy to the setting of DatalogMTL while overcoming the aforementioned difficulties. Analogously to the naïve approach, each iteration of the main loop captures a single materialisation step consisting of a round of rule applications (Line 3) followed by the coalescing of relevant facts (Line 4); as before, dataset $\mathcal{D}'$ stores the partial materialisation resulting from each iteration and is initialised as the input dataset, whereas the dataset $\mathcal{N}$ stores the facts obtained as a result of rule application and is initialised as empty. Following the rationale behind the seminaïve strategy for Datalog, newly derived information in each materialisation step is now stored as a dataset Δ, which is initialised as

Procedure 2: Seminaïve($\Pi, \mathcal{D}, P(\mathbf{c})@\varrho$)

Input: A program Π, a dataset $\mathcal{D}$, and a fact $P(\mathbf{c})@\varrho$
Output: Either inconst, or a pair of a truth value and a dataset

1 Initialise $\mathcal{N}$ to $\emptyset$, and both Δ and $\mathcal{D}'$ to $\mathcal{D}$;
2 **loop**
3 $\quad \mathcal{N} := \Pi[\mathcal{D}' \vdots \Delta]$;
4 $\quad \mathcal{C} := \mathsf{coal}(\mathcal{D}' \cup \mathcal{N})$;
5 $\quad \Delta := \{M@\varrho \in \mathcal{C} \mid$
$\quad\quad M@\varrho$ entails some fact in $\mathcal{N}$ which is not entailed by $\mathcal{D}'\}$;
6 $\quad$ **if** $\mathcal{D}'$ satisfies the body of some $\bot$-rule in Π **then return** inconst;
7 $\quad$ **if** $\mathcal{D}' \models P(\mathbf{c})@\varrho$ **then return** $(\mathsf{true}, \mathcal{D}')$;
8 $\quad$ **if** $\Delta = \emptyset$ **then return** $(\mathsf{false}, \mathcal{D}')$;
9 $\quad \mathcal{D}' := \mathcal{C}$;

the input dataset $\mathcal{D}$ and which is suitably maintained in each iteration (Line 5); furthermore, Procedure 2 ensures in Line 3 that only rule instances for which it is essential to involve facts from Δ (as formalised in the following definition) are taken into account during rule application.

Definition 10 *Let r be a rule of the form $M' \leftarrow M_1 \wedge \cdots \wedge M_n$, for some $n \geq 1$, and let $\mathcal{D}$ and Δ be datasets. The set of instances for r and $\mathcal{D}$ relative to Δ is defined as follows:*

$$\mathsf{inst}_r[\mathcal{D} \vdots \Delta] = \big\{(M_1\sigma@\varrho_1, \ldots, M_n\sigma@\varrho_n) \in \mathsf{inst}_r[\mathcal{D}] \mid \mathcal{D} \setminus \Delta \not\models M_i\sigma@\varrho_i, \text{ for some } i \in \{1, \ldots, n\}\big\}. \quad (3.1)$$

The set $r[\mathcal{D} \vdots \Delta]$ of facts derived by r from $\mathcal{D}$ relative to Δ is defined analogously to $r[\mathcal{D}]$ in Definition 6, with the exception that $\mathsf{inst}_r[\mathcal{D}]$ is replaced with $\mathsf{inst}_r[\mathcal{D} \vdots \Delta]$ in Expression (2.6). Finally, the set $\Pi[\mathcal{D} \vdots \Delta]$ of facts derived from $\mathcal{D}$ by one-step seminaïve application of Π is defined as $\Pi[\mathcal{D}]$ in Expression (2.7), by replacing $r[\mathcal{D}]$ with $r[\mathcal{D} \vdots \Delta]$.

In each materialisation step, Procedure 2 exploits Definition 10 to identify as relevant those rule instances with some 'new' element (i.e., one that cannot be entailed without the facts in Δ). The facts derived by such relevant rule instances in each iteration are stored in set $\mathcal{N}$ (Line 3). Note that all facts stored in sets $\mathcal{C}$, $\mathcal{N}$, and $\mathcal{D}'$ are relational and hence the entailment checks needed to compute the set Δ can be performed syntactically by checking containment between intervals; for instance a relational fact $R_4(c_2)@[0,2]$ entails fact $R_4(c_2)@[0,1]$ because of the inclusion $[0,1] \subseteq [0,2]$, but it does not entail the fact $R_4(c_2)@[1,3]$ since $[1,3] \not\subseteq [0,2]$.

As in the naïve approach, rule application is followed by a coalescing step where the partial materialisation is updated with the facts derived from rule application (Line 4). In contrast to the naïve approach, however, Procedure 2 needs to maintain set Δ to ensure that it captures only new facts. This is achieved in Line 5, where a fact in the updated partial materialisation is considered new if it entails a fact in $\mathcal{N}$ that was not already entailed by the previous partial materialisation. The procedure terminates in Line 6 if an inconsistency has been derived, in Line 7 if the input fact has been derived, or in Line 8 if Δ is empty. Otherwise, the procedure carries over the updated partial materialisation and the set of newly derived facts to the next materialisation step.

We next illustrate the application of the procedure to $\mathcal{D}_{\mathsf{ex}}$ and Π_{ex} and our example fact $R_1(c_1, c_2)@4$ from Example 5. In the first materialisation step, all input facts are considered as newly derived (i.e., $\Delta = \mathcal{D}$) and hence $\mathcal{N} = \Pi[\mathcal{D}' : \Delta] = \Pi[\mathcal{D}']$ and the result of coalescing coincides with the partial materialisation computed by the naïve procedure (i.e., $\mathcal{C} = \mathcal{D}^1_{\mathsf{ex}}$). Then, the procedure identifies as new all facts in $\mathcal{D}^1_{\mathsf{ex}}$ which are not entailed by $\mathcal{D}_{\mathsf{ex}}$, namely:

$$\Delta = \{R_1(c_1, c_2)@[0,2], \quad R_4(c_2)@[0,2], \quad R_5(c_2)@[2,2]\}.$$

In the second step, rule evaluation is performed in Line 3; since Δ is used for this, the procedure no longer considers the redundant instance $(R_2(c_1, c_2)@[0, 2], \boxplus_{[1,2]} R_3(c_2, c_3)@[1, 2])$ of r_2, since $\boxplus_{[1,2]} R_3(c_2, c_3)@[1, 2]$ is entailed already by $\mathcal{D}^1_{\mathsf{ex}} \setminus \Delta$. Similarly, the redundant instance $(\diamondminus_{[0,1]} R_5(c_2)@[0, 2])$ of Rule r_3 is not considered, as $\mathcal{D}^1_{\mathsf{ex}} \setminus \Delta \models \diamondminus_{[0,1]} R_5(c_2)@[0, 2]$. In contrast, all non-redundant facts derived by the naïve strategy are also derived by the seminaïve procedure and after coalescing dataset $\mathcal{C} = \mathcal{D}^2_{\mathsf{ex}}$, the set Δ is now updated as follows:

$$\Delta = \{R_1(c_2, c_2)@[0, 3], \qquad R_4(c_2)@[0, 3], \qquad R_6(c_2)@[2, 2]\}.$$

In particular, note that Δ contains the coalesced fact $R_4(c_2)@[0, 3]$ instead of fact $R_4(c_2)@[2, 3]$ derived from rule application. Datasets Δ and $\mathcal{D}' = \mathcal{D}^2_{\mathsf{ex}}$ are passed on to the third materialisation step, where all redundant computations identified in Section 2.3.1 are avoided with the only exception of fact $R_6(c_2)@[2, 2]$, which is re-derived using the instance of r_4 which consists of the three facts: $R_1(c_2, c_2)@[0, 3]$, $\boxminus_{[0,2]} R_4(c_2)@[2, 3]$, and $R_5(c_2)@[2, 2]$. Note that this is a new instance which was not used in previous iterations, and hence it does not violate the non-repetition property. Note also that, as with the naïve strategy, our seminaïve procedure terminates on our running example in this materialisation step since input fact $R_1(c_1, c_2)@[4, 4]$ has been derived.

We conclude this section by establishing correctness of our procedure, similarly as we did in Theorems 7 and 8 for the naïve strategy from Procedure 1.

Theorem 11 (Soundness) *Consider Procedure 2 running on input Π and $\mathcal{D}$. Upon the completion of the kth (for some $k \in \mathbb{N}$) iteration of the loop of Procedure 2, it holds that $\mathfrak{I}_{\mathcal{D}'} \subseteq T^k_{\Pi}(\mathfrak{I}_{\mathcal{D}})$.*

Proof The proof relies on the observation that rule instances processed by semi-

naïve evaluation are also processed by the naïve evaluation. In particular, directly by Definition 10 we obtain that $\mathsf{inst}_r[\mathcal{D}' \,\vdots\, \Delta] \subseteq \mathsf{inst}_r[\mathcal{D}']$, for each r, $\mathcal{D}'$, and Δ. Hence, in each loop iteration, the sets $\mathcal{N}$ and $\mathcal{C}$ in Procedure 2 are subsets of the corresponding sets in Procedure 1. Consequently, the same holds for the set $\mathcal{D}'$, and so, soundness follows from Theorem 7. □

Completeness is proved by induction on the number k of iterations of the main loop. In particular, we show that if $T_\Pi^{k+1}(\mathfrak{I}_\mathcal{D})$ satisfies a new fact $M@t$, then there must be a rule r and an instance in $\mathsf{inst}_r[\mathcal{D}_k \,\vdots\, \Delta_k]$ witnessing the derivation of $M@t$. Note that since $\mathsf{inst}_r[\mathcal{D}_k \,\vdots\, \Delta_k] \subseteq \mathsf{inst}_r[\mathcal{D}_k]$, it requires strengthening the argument from the proof of Theorem 8, where it was sufficient to show that such an instance is in $\mathsf{inst}_r[\mathcal{D}_k]$. This will imply that each fact satisfied by $T_\Pi^{k+1}(\mathfrak{I}_\mathcal{D})$ is derived in the $k+1$st iteration of our procedure.

Theorem 12 (Completeness) *Consider Procedure 2 with input program Π and input dataset $\mathcal{D}$. For each $k \in \mathbb{N}$, upon the completion of the kth iteration of the loop of Procedure 2, it holds that $T_\Pi^k(\mathfrak{I}_\mathcal{D}) \subseteq \mathfrak{I}_{\mathcal{D}'}$.*

Proof We will use $\mathcal{N}_k$, Δ_k, and $\mathcal{D}'_k$, for the contents of, respectively, $\mathcal{N}$, Δ, and $\mathcal{D}'$ in Procedure 2 upon the completion of the kth iteration of the loop. Now, as in the proof of Theorem 8, we will show inductively on natural numbers k, that $T_\Pi^k(\mathfrak{I}_\mathcal{D}) \subseteq \mathfrak{I}_{\mathcal{D}_k}$.

The only difference with respect to the proof of Theorem 8 lies in Case 2, where we assume that $T_\Pi^k(\mathfrak{I}_\mathcal{D}) \not\models M@t$ and $T_\Pi^{k+1}(\mathfrak{I}_\mathcal{D}) \models M@t$, for some relational fact $M@t$. By the inductive assumption, as in the proof of Theorem 8, there needs to be an instance $(M_1\sigma@\varrho_1, \cdots, M_n\sigma@\varrho_n) \in \mathsf{inst}_r[\mathcal{D}_k]$ of some rule r of the form $M' \leftarrow M_1 \wedge \cdots \wedge M_n$, and a time point $t' \in \varrho_1 \cap \cdots \cap \varrho_n$ such that $M'\sigma@t' \models M@t$. Now, we argue that this instance is not only in $\mathsf{inst}_r[\mathcal{D}_k]$, but

also in $\mathsf{inst}_r[\mathcal{D}_k \vdots \Delta_k]$, namely that $(M_1\sigma@\varrho_1, \cdots, M_n\sigma@\varrho_n) \in \mathsf{inst}_r[\mathcal{D}_k \vdots \Delta_k]$. To this end, by Definition 10, it suffices to show that there is $i \in \{1, \ldots, n\}$ such that $\mathcal{D}_k \setminus \Delta_k \not\models M_i\sigma@\varrho_i$. We will consider two cases, namely, when $k = 0$ and when $k > 0$.

- Case 2.1: $k = 0$. By the initialisation (Line 1) of the of Procedure 2, $\Delta_0 = \mathcal{D}$, so $\mathcal{D}_0 \setminus \Delta_0 = \emptyset$. Recall that we assumed that input programs do not have rules with vacuously satisfied bodies, so there needs to exist $i \in \{1, \ldots, n\}$ such that the empty interpretation does not satisfy $M_i\sigma@\varrho_i$, that is, $\mathcal{D}_0 \setminus \Delta_0 \not\models M_i\sigma@\varrho_i$.
- Case 2.2: $k > 0$. By Lines 9 and 4, we have $\mathcal{D}_k = \mathsf{coal}(\mathcal{D}_{k-1} \cup \mathcal{N}_k)$. Moreover, by the definition of Δ in Line 5, we obtain $\mathcal{D}_k = \mathsf{coal}(\mathcal{D}_{k-1} \cup \Delta_k)$. Thus $\mathfrak{I}_{\mathcal{D}_k} = \mathfrak{I}_{\mathcal{D}_{k-1}} \cup \mathfrak{I}_{\Delta_k}$, so $\mathfrak{I}_{\mathcal{D}_k} \setminus \mathfrak{I}_{\Delta_k} = \mathfrak{I}_{\mathcal{D}_{k-1}} \setminus \mathfrak{I}_{\Delta_k}$, and therefore $\mathfrak{I}_{\mathcal{D}_k} \setminus \mathfrak{I}_{\Delta_k} \subseteq \mathfrak{I}_{\mathcal{D}_{k-1}}$. Hence, to show that for some $i \in \{1, \ldots, n\}$ we have that $\mathcal{D}_k \setminus \Delta_k \not\models M_i\sigma@\varrho_i$, it suffices to show that $\mathcal{D}_{k-1} \not\models M_i\sigma@\varrho_i$. Suppose towards a contradiction that $\mathcal{D}_{k-1} \models M_i\sigma@\varrho_i$, for all $i \in \{1, \ldots, n\}$. By Theorem 11, $T_\Pi^{k-1}(\mathfrak{I}_\mathcal{D}) \models M_i\sigma@\varrho_i$, for all $i \in \{1, \ldots, n\}$. Thus, $T_\Pi^k(\mathfrak{I}_\mathcal{D}) \models M@t$, which raises a contradiction.

□

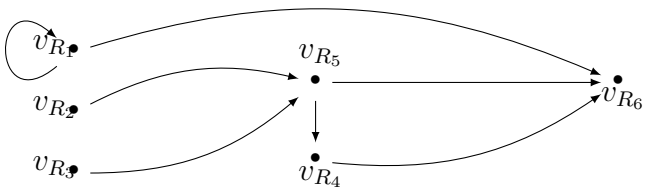
3.2 Optimised Seminaïve Evaluation

Although the seminaïve procedure enjoys the non-repetition property, it can still re-derive facts already obtained in previous materialisation steps; as a result, it can incur in a potentially large number of redundant computations. In particular, as discussed in Section 3.1, fact $R_6(c_2)@[2, 2]$ is re-derived using Rule r_4 in the third materialisation step of our running example, and (if the procedure did not

terminate) it would also be re-derived in all subsequent materialisation steps (by different instances of Rule r_4).

In this section, we present an optimised variant of our seminaïve procedure which further reduces the number of redundant computations performed during materialisation. The main idea is to disregard rules during the execution of the procedure as soon as we can be certain that their application will never derive new facts in subsequent materialisation steps. In our example, Rule r_4 can be discarded after the second materialisation step as its application will only continue to re-derive fact $R_6(c_2)@[2,2]$ in each subsequent materialisation step.

To this end, we will exploit the distinction between recursive and non-recursive predicates in a program. In the case of our Example 5, predicates R_2, R_3, R_4, and R_5 are non-recursive, whereas R_1 and R_6 are recursive, since in the dependency graph of the program Π_{ex}—depicted below—only vertices corresponding to R_1 and R_6 can be reached by paths containing a cycle. Hence, Rules r_2 and r_3 are non-recursive in Π_{ex} since their heads mention non-recursive predicates, whereas r_1 and r_4 are recursive.



Now, in contrast to recursive predicates, for which new facts can be derived in each materialisation step, the materialisation of non-recursive predicates will be completed after linearly many materialisation steps; from then on, the rules will no longer derive any new facts involving these predicates. This observation can be exploited to optimise our seminaïve evaluation as follows. Assume that the pro-

cedure has fully materialised all non-recursive predicates in the input program. At this point, we can safely discard all non-recursive rules; furthermore, we can also discard a recursive rule r with a non-recursive body atom M if the current partial materialisation does not entail any grounding of M (in this case, r cannot apply in further materialisation steps). An additional optimisation applies to forward-propagating programs, where rules cannot propagate information 'backwards' along the timeline; in this case, we can compute the maximal time points for which each non-recursive body atom in r may possibly hold, select the minimum t_r amongst such values, and discard r as soon as we can determine that the materialisation up to time point t_r has been fully completed.

In the case of our Example 5, materialisation of the non-recursive predicates R_2, R_3, R_4, and R_5 is complete after two materialisation steps. Hence, at this point we can disregard non-recursive Rules r_2 and r_3 and focus on the recursive forward-propagating Rules r_1 and r_4. Furthermore, the maximum time point at which the non-recursive predicates R_4 and R_5 from r_4 can hold are 3 and 2, respectively, and hence $t_{r_4} = 2$; as upon the completion of the second materialisation step we can be certain that the remaining body atom of r_6, namely R_1, has been materialised up to t_{r_6}, we can discard the rule r_6. In subsequent materialisation steps we can apply only Rule r_1, thus avoiding many redundant computations.

Procedure 3 implements these ideas by extending our seminaïve materialisation from Lines 3–8. In particular, in each materialisation step, the procedure checks whether all non-recursive predicates have been fully materialised (Line 9), in which case it removes all non-recursive rules from the input program as well as all recursive rules with an unsatisfied non-recursive body atom (Lines 10–12). It also sets the flag to 1, which activates the additional optimisation for forward propagating programs, which is applied in Lines 13–18.

Procedure 3: OptimisedSeminaïve($\Pi, \mathcal{D}, P(\mathbf{c})@\varrho$)

Input: A program Π, a dataset $\mathcal{D}$, and a fact $P(\mathbf{c})@\varrho$

Output: Either inconst, or a pair of a truth value and a dataset

Initialise $\mathcal{N}$ to $\emptyset$, Δ and $\mathcal{D}'$ to $\mathcal{D}$, Π' to Π, $flag$ to 0, and S_r (for each $r \in \Pi$) to the set of body atoms in r that are non-recursive in Π;

loop

- $\mathcal{N} := \Pi'[\mathcal{D}' : \Delta]$;
- $\mathcal{C} := \mathsf{coal}(\mathcal{D}' \cup \mathcal{N})$;
- $\Delta := \{M@\varrho \in \mathcal{C} \mid M@\varrho$ entails some fact in $\mathcal{N}$ but which is not entailed by $\mathcal{D}'\}$;
- **if** $\mathcal{D}'$ satisfies the body of some $\bot$-rule in Π **then return** inconst;
- **if** $\mathcal{D}' \models P(\mathbf{c})@\varrho$ **then return** $(\mathsf{true}, \mathcal{D}')$;
- **if** $\Delta = \emptyset$ **then return** $(\mathsf{false}, \mathcal{D}')$;
- **if** $flag = 0$ *and the datasets* $\mathcal{D}'$ *and* $\mathcal{C}$ *satisfy the same facts with non-recursive in* Π' *predicates* **then**
  - Set $flag$ to 1 and Π' to the recursive fragment of Π;
  - **for** *each* $r \in \Pi'$ **do**
    - **if** *there is* $M \in S_r$ *such that* $\mathcal{D}' \not\models M\sigma@t$*, for each substitution* σ *and time point* t **then** $\Pi' := \Pi' \setminus \{r\}$;
- **if** $flag = 1$ *and* Π' *is forward propagating* **then**
  - **for** *each* $r \in \Pi'$ **do**
    - **for** *each* $M \in S_r$ **do**
      - $t^M_{max} :=$ maximum right endpoint amongst all intervals ϱ satisfying $\mathcal{D}' \models M\sigma@\varrho$, for some substitution σ;
    - $t_r :=$ minimum value in $\{t^M_{max} \mid M \in S_r\}$;
    - **if** *for each* $M@\varrho \in \mathcal{C}$ *with* ϱ *overlapping* $(-\infty, t_r]$*, there is* $M@\varrho' \in \mathcal{D}'$ *with* $\varrho \cap (-\infty, t_r] \subseteq \varrho'$ **then** $\Pi' := \Pi' \setminus \{r\}$;
- $\mathcal{D}' := \mathcal{C}$;

We conclude this section by establishing correctness of our procedure. We first observe that, as soon as the algorithm switches the flag to 1, we can be certain that all facts with non-recursive predicates have been fully materialised.

Lemma 13 *Consider Procedure 3 running on input program* Π *and dataset* $\mathcal{D}$*. Whenever* $flag = 1$*, dataset* $\mathcal{D}'$ *satisfies all facts over a non-recursive predicate*

in Π *that are entailed by* Π *and* $\mathcal{D}$.

Proof In Procedure 3, $flag$ is initialised to 0 and once it is changed to 1 (in Line 10) its value cannot be reverted. Moreover, the consecutive contents of $\mathcal{D}'$ are obtained by coalescing the previous contents with facts in $\mathcal{N}$ (in Line 4), so once a fact is entailed by $\mathcal{D}'$, it will remain entailed by all the consecutive contents of $\mathcal{D}'$. Therefore, to prove the lemma, it suffices to consider the step of computation in which $flag$ becomes 1. Let k be this step and let $\mathcal{D}'_k$, be the contents of $\mathcal{D}'$ upon the completion of the kth iteration of the loop. Hence, we need to show that $\mathcal{D}'_k$ satisfies all the facts which are satisfied in $\mathfrak{C}_{\Pi,\mathcal{D}}$ and which mention non-recursive predicates in Π.

For this, we observe that before $flag$ changes to 1, Procedure 3 works as Procedure 2 so, by Theorems 11 and 12, we have $\mathfrak{I}_{\mathcal{D}'_k} = T^k_\Pi(\mathfrak{I}_\mathcal{D})$. Hence, as $\mathfrak{C}_{\Pi,\mathcal{D}} = T^{\omega_1}_\Pi(\mathfrak{I}_\mathcal{D})$, it suffices to show by a transfinite induction that, for each ordinal $\alpha \geq k$, the interpretations $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ and $T^\alpha_\Pi(\mathfrak{I}_\mathcal{D})$ satisfy the same facts with non-recursive predicates in Π. For the base case assume that $T^{k+1}_\Pi(\mathfrak{I}_\mathcal{D}) \models M@t$, for some relational fact $M@t$ whose predicate is non-recursive in Π. Hence, there is a rule $r \in \mathsf{ground}(\Pi, \mathcal{D})$ and a time point t' such that $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ entails each body atom of r at t', and the head of r holding at t' entails $M@t$. Thus the head atom of r has a non-recursive predicate in Π; therefore, by the definition, each body atom in r also mentions only non-recursive predicates in Π. Now, as $T^k_\Pi(\mathfrak{I}_\mathcal{D}) \not\models M@t$, we obtain that $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ and $T^{k-1}_\Pi(\mathfrak{I}_\mathcal{D})$ do not satisfy the same relational facts with non-recursive predicates in Π, which directly contradicts the condition from Line 9. The inductive step for a successor ordinal α uses the same argument; indeed, if $T^\alpha_\Pi(\mathfrak{I}_\mathcal{D}) \models M@t$ and $T^{\alpha-1}_\Pi(\mathfrak{I}_\mathcal{D}) \models M@t$, for some relational fact $M@t$ whose predicate is non-recursive in Π, then $T^\alpha_\Pi(\mathfrak{I}_\mathcal{D})$ and $T^{\alpha-1}_\Pi(\mathfrak{I}_\mathcal{D})$ do not satisfy the same relational facts with non-recursive predicates, contradicting the inductive

assumption. The inductive step for the transfinite ordinal α holds trivially as $T^{\alpha}_{\Pi}(\mathfrak{I}_{\mathcal{D}}) = \bigcup_{\beta<\alpha} T^{\beta}_{\Pi}(\mathfrak{I}_{\mathcal{D}})$. □

This lemma shows us that once the $flag$ is changed to 1, Procedure 3 can safely ignore all the non-recursive rules, as they can no longer be used to derive any new facts. Thus, such rules are deleted from Π in Line 12. The deletion of rules in Line 18, on the other hand, is based on a simple observation that a derivation of a new fact $M@t$ by a forward-propagating program is based only on facts holding at time points smaller-or-equal to t. Thus, if for a forward-propagating program Π' (condition in Line 13) it holds that $\mathfrak{I}_{\mathcal{D}'}$ and $T_{\Pi'}(\mathfrak{I}_{\mathcal{D}'})$ satisfy the same facts within $(-\infty, t_r]$ (condition in Line 18), then $\mathfrak{I}_{\mathcal{D}'}$ and $T^{\alpha}_{\Pi'}(\mathfrak{I}_{\mathcal{D}'})$ will coincide over $(-\infty, t_r]$, for each ordinal number α. By the construction, t_r is the maximum time point in which the body of rule r can hold, so if the above conditions are satisfied, we can safely delete r from the program in Line 18. Consequently, we obtain the following result.

Lemma 14 *Whenever Procedure 3 removes a rule r from Π' in either Line 12 or 18, $\mathfrak{C}_{\Pi',\mathcal{C}} = \mathfrak{C}_{\Pi'\setminus\{r\},\mathcal{C}}$.*

Finally, using Lemma 14 together with the soundness and completeness of our seminaïve evaluation (established in Theorems 11 and 12), we can show soundness and completeness of the optimised version of the procedure.

Theorem 15 (Soundness and Completeness) *Consider Procedure 3 with input program Π and input dataset $\mathcal{D}$. For each $k \in \mathbb{N}$, upon the completion of the kth iteration of the loop of Procedure 2, it holds that $T^{k}_{\Pi}(\mathfrak{I}_{\mathcal{D}}) = \mathfrak{I}_{\mathcal{D}'}$.*

Proof If for both Lines 12 and 18, the condition in the 'if' statement never applies, then Procedure 3 works in the same way as Procedure 2. Hence, by Theorems 11 and 12, we get $T^{k}_{\Pi}(\mathfrak{I}_{\mathcal{D}}) = \mathfrak{I}_{\mathcal{D}'}$. Otherwise, the loop from Procedure 3

works similarly as Procedure 2, except that it can delete in Lines 12 and 18 some rules. By Lemma 14, however, such rules can be safely deleted from the program, without losing the properties established in Theorems 11 and 12. □

Note that our optimisation for forward-propagating programs in Lines 13–18 can be adopted in a straightforward way for backwards-propagating programs, as these cases are symmetric.

We conclude this section by recalling that Procedure 3 is non-terminating, as it is possible for the main loop to run for an unbounded number of iterations. In our approach, we will also exploit a terminating (but incomplete) variant of the procedure, which we call OptimisedSeminaïveHalt. This variant is obtained by adding an additional termination condition just before Line 19 which breaks the loop and returns Π' and $\mathcal{D}'$ (without a truth value) if the flag has been set to 1. Thus, such a variant of the procedure ensures termination with a (partial) materialisation once all the non-recursive predicates in the input program have been fully materialised, and also returns a subset of relevant rules in the program.

3.3 Sound and Complete Reasoning Algorithm

We proposed a sound and complete reasoning algorithm combining optimised seminaïve materialisation and automata construction. Our algorithm delegates the bulk of the computation to the materialisation component and resorts to automata-based techniques only as needed to ensure termination and completeness.

Our approach is summarised in Algorithm 4. When given as input a program Π, a dataset $\mathcal{D}$, and a fact $P(\mathbf{c})@\varrho$, the algorithm starts by identifying the subset Π' of the input rules and the subset $\mathcal{D}'$ of the input data that are relevant to deriving the input fact $P(\mathbf{c})@\varrho$ (Lines 1 and 2). This optimisation is based on the observation

Algorithm 4: Sound and complete reasoning algorithm

Input: A program Π, a dataset $\mathcal{D}$, and a fact $P(\mathbf{c})@\varrho$
Output: Either inconst, or a pair of a truth value and a dataset

1 $\Pi' \coloneqq$ the set of P-relevant rules in Π;
2 $\mathcal{D}' \coloneqq$ the set of facts in $\mathcal{D}$ which mention predicates occurring in Π';
3 Out $\coloneqq$ OptimisedSeminaïveHalt$(\Pi', \mathcal{D}', P(\mathbf{c})@\varrho)$;
4 **if** *Out* $= (\Pi'', \mathcal{D}'')$ **then** assign $\Pi' \coloneqq \Pi''$ and $\mathcal{D}' \coloneqq \mathcal{D}''$;
5 **else return** Out;
6 Run two threads in parallel:
7 **Thread 1:**
8 **return** OptimisedSeminaïve$(\Pi', \mathcal{D}', P(\mathbf{c})@\varrho)$;
9 **Thread 2:**
10 $\Pi'', \mathcal{D}'' \coloneqq$ EntailToInconsist$(\Pi', \mathcal{D}', P(\mathbf{c})@\varrho)$;
11 $b \coloneqq$ AutomataProcedure$(\Pi'', \mathcal{D}'')$;
12 **return** the negation of $(b, \mathcal{D}')$;

that a program Π and dataset $\mathcal{D}$ entail a fact of the form $P(\mathbf{c})@\varrho$ if and only if so do the subset Π^P of P-relevant rules in Π and the subset $\mathcal{D}^P$ of facts in $\mathcal{D}$ mentioning only predicates from Π^P. Then, the algorithm proceeds according to the following steps.

1. Pre-materialise Π' and $\mathcal{D}'$ using the terminating (but possibly incomplete) version of the seminaïve materialisation procedure OptimisedSeminaïve-Halt discussed at the end of Section 3.2. If the procedure terminates because an inconsistency has been found, or the input fact has been derived, or a fix-point has been reached, then we can terminate and report output (Line 5); otherwise, we have obtained a partial materialisation where non-recursive predicates have been fully materialised and we can proceed.

2. Run two threads in parallel. In the first (possibly non-terminating) thread, continue performing optimised semi-naïve materialisation until the input fact (or an inconsistency) is derived or a fixpoint is reached. In turn, in the second thread we resort to automata-based techniques applied to the partial

materialisation obtained in the first stage of the algorithm; in particular, this second thread applies in Line 10 the reduction from fact entailment to inconsistency checking as described in Section 2.2.2, and then applies in Line 11 the automata-based decision procedure of Wałęga, Cuenca Grau, Kaminski, and Kostylev (2019), as described in Section 2.3.3. As soon as a thread returns a result, the other thread is terminated.

It follows directly from the results proved earlier in this section and the theoretical results of Wałęga, Cuenca Grau, Kaminski, and Kostylev (2019) that Algorithm 4 is sound, complete, and terminating. Furthermore, the algorithm is designed so that, on the one hand, materialisation is favoured over automata-based reasoning and, on the other hand, the automata construction relies on a recursive program that is as small as possible as well as on a partial materialisation that is as complete as possible. The favourable computational behaviour of this approach will be confirmed by our experiments.

3.4 Evaluation

We have implemented Algorithm 4 in a system called MeTeoR. The architecture of MeTeoR is illustrated in Figure 3.1, which mainly consists four parts: *Input*, *Parser*, *Reasoning techniques* and *Reasoning services*. In particular, our *Data Parser* support the following three types of forms for representing a fact:

- $A@t$, facts containing no term where t could be an integer or a decimal;
- $A(\mathbf{s})@t$ or $A(\mathbf{s})@[t,t]$, facts with punctual intervals where $\mathbf{s}$ is a tuple of terms and t could be an integer or a decimal;
- $A(\mathbf{s})@<t_1,t_2>$, facts with non-punctual intervals, where < is (or [, > is) or], $\mathbf{s}$ is a tuple of terms and t could be an integer or a decimal.

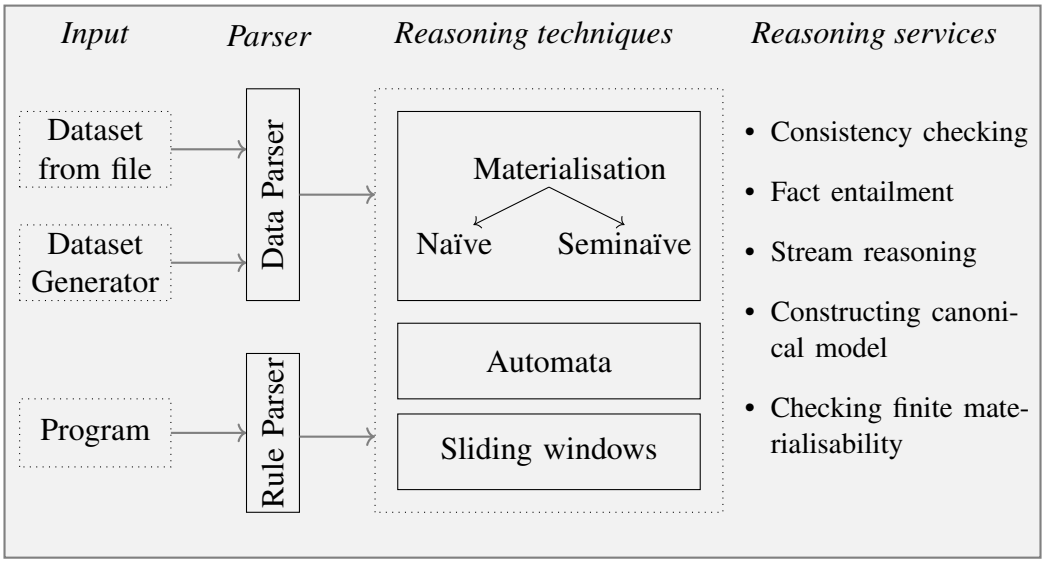


Figure 3.1: Illustration of MeTeoR's Architecture

To guarantee that our *Rule Parser* is able to accurately parse the input program, we define some keywords to denote six MTL operators, respectively:

$\diamondminus_{<t_1,t_2>}$ `Diamondminus`$< t_1, t_2 >$ or `SOMETIME`$< -t_2, -t_1 >$;

$\boxminus_{<t_1,t_2>}$ `Boxminus`$< t_1, t_2 >$ or `ALWAYS`$< -t_2, -t_1 >$;

$\diamondplus_{<t_1,t_2>}$ `Diamondplus`$< t_1, t_2 >$ or `SOMETIME`$< t_1, t_2 >$;

$\boxplus_{<t_1,t_2>}$ `Boxplus`$< t_1, t_2 >$ or `ALWAYS`$< t_1, t_2 >$;

$\mathcal{S}_{<t_1,t_2>}$ `Since`$[t_1, t_2]$

$\mathcal{U}_{<t_1,t_2>}$ `Until`$[t_1, t_2]$

Besides, for the sake of typing convenience, we define `,` and `:-` to denote the conjunction notation $\wedge$ and the derivative arrow $\leftarrow$ expressed in the rule form (2.2), respectively. Our system is implemented in Python 3.8, and does not depend on other third-party libraries. In particular, we provide an online demo website, which can also be accessed via https://meteor.cs.ox.ac.uk/. More

details about on how to use MeToeR can be found in the document https://github.com/wdimmy/MeTeoR/blob/main/README.md. Next, we describe some key implementation choices made in MeTeoR and report the results of an extensive empirical evaluation on available benchmarks.

3.4.1 Key Engineering Implementation Details

A scalable implementation of materialisation-based reasoning requires suitable representation and storage schemes for facts. In MeTeoR, we associate to each ground relational atom a list of intervals sorted by their left endpoints, which provides a compact account of all facts mentioning a given relational atom. Each ground relational atom is further indexed by a composite key consisting of its predicate and its tuple of constants. This layout is useful for fact coalescing and fact entailment checking; for instance, to check if a fact $M@\varrho$ is entailed by a dataset $\mathcal{D}$, it suffices to find and interval ϱ' such that $M@\varrho' \in \mathcal{D}$ and $\varrho \subseteq \varrho'$; this can be achieved by first scanning the sorted list of intervals for M using the index and checking if ϱ is a subset of one of these intervals. Additionally, each ground relational atom is also indexed by each of its term arguments to facilitate joins. Finally, when a fact is inserted into the dataset, the corresponding list of intervals is sorted to facilitate subsequent operations. MeTeoR also implements an optimised version of (temporal) joins, which are required to evaluate rules with multiple body atoms. A naïve implementation of the join of metric atoms $M_1, \ldots, M_n$ occurring in the body of a rule would require computing all intersections of intervals $\varrho_1, \ldots, \varrho_n$ such that $M_i@\varrho_i$ occurs in the so-far constructed dataset, for each $i \in \{1, \ldots, n\}$. Since each M_i may hold in multiple intervals in the dataset, the naïve approach is inefficient in practice. In contrast, we implemented a variant of the classical sort-merge join algorithm: we first sort all n lists of intervals corresponding to $M_1, \ldots, M_n$, and then scan the sorted lists to compute the in-

tersections, which improves performance. Our approach to temporal joins can be seen as a generalisation of the idea sketched by Brandt et al. (2018, Section 5), which deals with two metric atoms at a time but has not been put into practice to the best of our knowledge.

3.4.2 Baselines and Machine Configuration

We compared MeTeoR with two baselines: the query rewriting approach proposed by Brandt et al. (2018) and the reduction to LTL reasoning proposed by Brandt et al. (2018) which was later implemented by Yang (2022). We could not compare our approach to the temporal exension of the Vadalog system recently proposed by Bellomarini, Blasi, Nissl, and Sallinger (2022) as the system is not accessible.

The query rewriting approach by Brandt et al. (2018) is based on rewriting a target predicate P with respect to an input program Π into an SQL query that, when evaluated over the input dataset $\mathcal{D}$, provides the set of all facts with maximal intervals over P entailed by Π and $\mathcal{D}$. The only implementation of this approach that we are aware of is the extension of the Ontop system described by Kalaycı, Xiao, Ryzhikov, Kalayci, and Calvanese (2018); this implementation, however, is no longer available[1] and hence we have produced our own implementation. Following Brandt et al. (2018), we used temporary SQL tables[2] (instead of subqueries) to compute the extensions of predicates appearing in Π on the fly, which has two implications. First, we usually have not just one SQL query but a set of queries for computing the final result; second, similarly to MeTeoR, the approach essentially works bottom-up rather than top-down. The implementation by Brandt et al. (2018) provides two variants for coalescing: the first one uses standard SQL

[1] Personal communication with the authors.

[2] A temporary table in SQL is a special type of table that exists temporarily and is used to store data temporarily during the execution of SQL statements or procedures.

queries by Zhou, Wang, and Zaniolo (2006), whereas the second one implements coalescing explicitly. For our baseline we adopted the standard SQL approach, which is less dependent on implementation details. Finally, we chose Postgresql 10.18 as the underpinning database for all our baseline experiments.

As shown by Brandt et al. (2018), DatalogMTL reasoning can be reduced to LTL reasoning by means of an exponential translation. This approach has been implemented by Yang (2022) using BLACK (Geatti, Gigante, and Montanari, 2019) as the LTL reasoner of choice, and we used their implementation as a baseline.

3.4.3 Benchmarks

The *Temporal LUBM Benchmark* is an extension of the Lehigh University Benchmark (Guo, Pan, and Heflin, 2005) with temporal rules and data, which we developed (Wang, Hu, Wałęga, and Cuenca Grau, 2022). LUBM's data generator, which provides means for generating datasets of different sizes, has been extended to randomly assign an interval to each generated fact; the rational endpoints of each interval belong to a range, which is a parameter. In addition, the plain Datalog rules from the OWL 2 RL fragment of LUBM have been extended with 29 rules involving recursion and mentioning all metric operators in DatalogMTL. The resulting DatalogMTL program is denoted by Π_L.

The *Weather Benchmark* is based on freely available data with meteorological observations in the US (Maurer, Wood, Adam, Lettenmaier, and Nijssen, 2002)[3]; in particular, we used a dataset $\mathcal{D}_W$ consisting of 397 million temporal facts spanning over the years 1949–2010, and then considered smaller subsets of this dataset. Similar data was used in experiments of Brandt et al. (2018), however the format of data has slightly changed, so we adapted their (non-recursive) program (consist-

[3] www.engr.scu.edu/~emaurer/gridded_obs/index_gridded_obs.html

ing of 4 rules) for detecting US states affected by excessive heat and heavy wind, so that the program matches the data. The obtained program is still non-recursive and, in what follows, it is denoted by Π_W.

The *iTemporal Benchmark* provides a generator for DatalogMTL programs and datasets (Bellomarini, Nissl, and Sallinger, 2022).[4] The benchmark can generate different types of DatalogMTL programs and equips them with datasets. We used iTemporal to generate 10 recursive programs $\Pi_I^1, \ldots, \Pi_I^{10}$ with different structures and containing 20-30 rules each, as well as corresponding datasets of various size.

3.4.4 Experiments

Automata vs LTL. Although MeTeoR utilises automata to ensure completeness and termination, we could have alternatively relied on the translation to LTL implemented by Yang (2022) together with BLACK (Geatti, Gigante, and Montanari, 2019) as the LTL reasoner of choice. To determine which choice is favourable, we considered programs $\Pi_I^1, \ldots, \Pi_I^{10}$ from the iTempora benchmark, together with datasets containing $1,000$ facts each. For each pair of a dataset and a program, we constructed 10 query facts; these facts were generated by first randomly choosing a predicate that appears in a program or in a dataset; second, constants are randomly chosen among those present in the program and in the dataset; third, an interval for the fact is generated. Next, we performed fact entailment by first reducing to inconsistency checking and then either applying the automata-based approach or constructing an LTL-formula and checking its satifiability with BLACK. Average running times per 10 query facts for both approaches are reported in Figure 3.2 (left), where each point represents the running times for the automata-based approach (vertical coordinate) and LTL-based approach (horizontal coordinate) for a single program and dataset. We can observe that

[4] https://github.com/kglab-tuwien/iTemporal.git

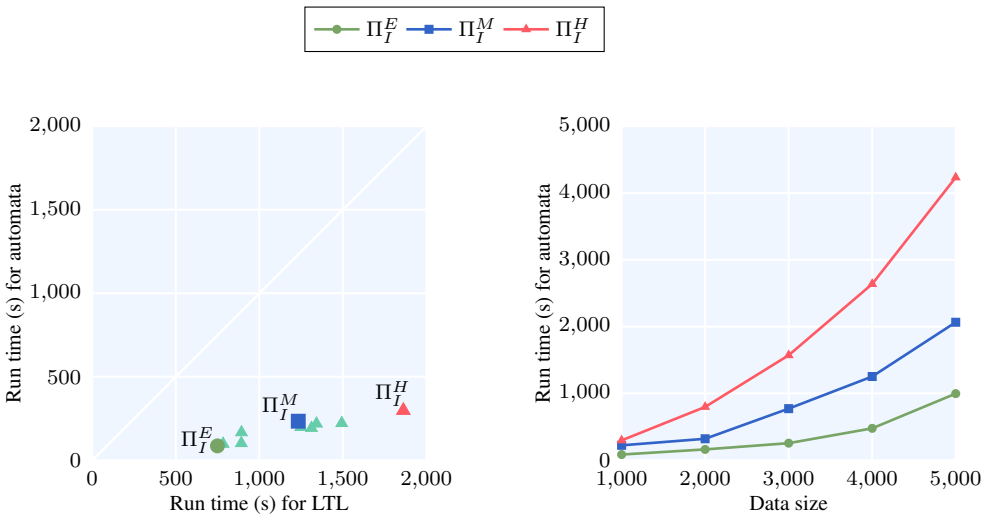


Figure 3.2: Comparison of automata- and LTL-based approaches (left) and scalability of the automata approach (right). Π_I^E, Π_I^M and Π_I^H, represent an 'easy' program, a 'medium' program and a 'hard' program, respectively

the automata-based approach is consistently about an order of magnitude faster, which justifies our choice over an LTL-based reasoner.

Scalability of automata We have conducted scalability experiments to understand the practical limitations of the automata-based approach. Based on the results from the previous experiment, we chose an 'easy' program Π_I^E, a 'medium' program Π_I^M, and a 'hard' program Π_I^H amongst programs $\Pi_I^1, \ldots, \Pi_I^{10}$, as depicted in Figure 3.2 (left). We used iTemporal to generate increasing size datasets for these programs and we randomly generated query facts as in the previous experiment. The runtimes of automata-based approach for such inputs are summarised in Figure 3.2 (right), where we observe that the automata-based procedure is able to solve non-trivial problems, but struggles to scale beyond datasets with a few thousand temporal facts.

Comparison of materialisation strategies We compared the naïve, seminaïve, and optimised seminaïve materialisation procedures described in Sections 2.3.1, 3.1, and 3.2, respectively. To this end, for each of the programs Π_i^E, Π_i^M, and Π_i^H we used iTemporal to generate a dataset containing one million facts. In Figure 3.3 we present running time and memory consumption (measured as the maximum number of facts stored in memory) over the first 30 materialisation steps in each case. Note that Theorems 11, 12, and 15 ensure that the naïve, seminaïve, and optimised seminaïve materialisation procedures will compute exactly the same output after any fixed number of materialisation steps. As we can see, the seminaïve procedure consistently outperforms the naïve approach both in terms of running time and memory consumption, especially as materialisation progresses. In turn, the optimised seminaïve approach disregards rules after 6th, 7th, and 8th materialisation steps for Π_I^E, Π_I^M, and Π_I^H, respectively; from then onwards, the optimised procedure outperforms the basic seminaïve procedure. We can note that after several steps of materialisation the memory usage stabilises; the number of facts in memory stops increasing but their intervals keep growing. One possible reason is that we executed the coalescing operation after each materialization step, allowing multiple facts to be combined into a single representation, thus avoiding the extra memory cost. However, if the derived facts cannot be coalesced, the memory usage will continue to increase as more facts are derived. From this perspective, the behavior observed in Figure 3.3 appears to be specific to the programs used in this experiment.

Scalability of optimised seminaïve materialisation Our previous experiments provide evidence of the superiority of optimised seminaïve materialisation over the two alternative approaches. We further evaluated the scalability of optimised seminaïve materialisation as the size of the data increases. To this end, we consid-

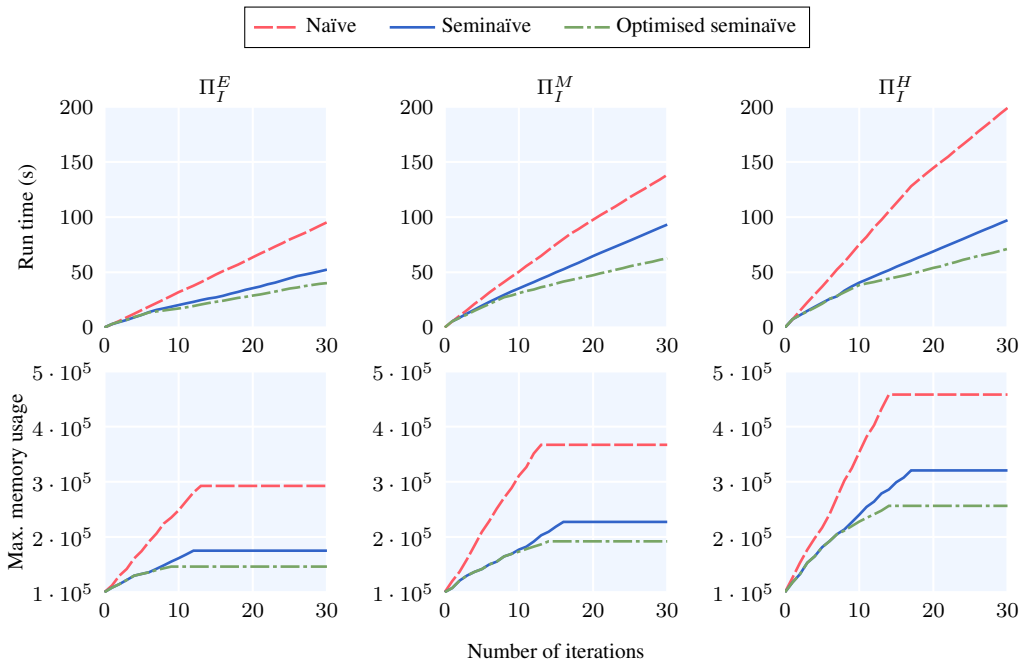


Figure 3.3: Comparison of time and memory consumption in various strategies

ered the three iTemporal programs Π_I^E, Π_I^M, and Π_I^H as well as program Π_L from the Temporal LUBM Benchmark, and Π_W from the Weather Benchmark. In Figure 3.4 we show running times and memory consumption after the first 10 materialisation steps for datasets with up to 10 million temporal facts. We note here that the program Π_W is non-recursive and its full materialisation is obtained already after 3 materialisation steps; all other programs are recursive. Our results suggest that the optimised seminaïve materialisation procedure exhibits favourable scalability behaviour as data size increases.

Comparison with query rewriting We compared the performance of MeTeoR with the query rewriting baseline. As the latter does not support recursive programs, we considered non-recursive programs only. For this, we generated non-recursive subsets of the program Π_L from the Temporal LUBM Benchmark and two subsets of the program Π_W from the Weather Benchmark. In particular, we

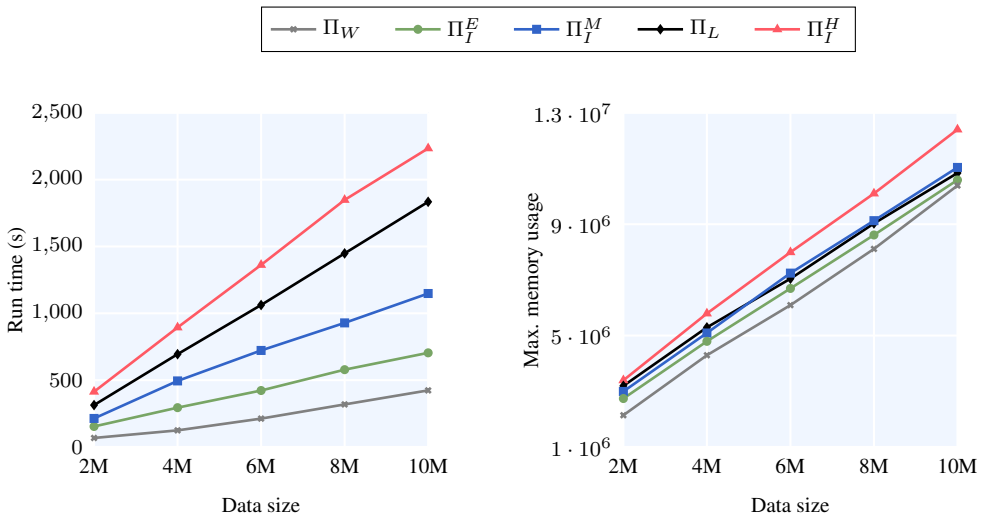


Figure 3.4: Scalability of optimised seminaïve materialisation

considered fragments Π_L^1 (with 5 rules), Π_L^2 (10 rules), and Π_L^3 (21 rules) of Π_L with rules relevant (see the end of Section 2.2.1 for the precise definition of this notion) to predicates $ResearchAssistant$, $Lecturer$, and $FullProfessorCandidate$, respectively. For the Weather Benchmark we used fragments Π_W^1 and Π_W^2 (each with 2 rules) of Π_W, with rules relevant to predicates $HeatAffectedState$ and $HeavyWindAffectedState$, respectively. For each program we generated a collection of datasets with $100,000$ facts each and compared the runtime of MeTeoR with the baseline in the task of computing all entailed facts over the predicates mentioned above. Note that since the programs are non-recursive, MeTeoR runs only (optimised seminaïve) materialisation without exploiting automata. Figure 3.5 presents results, where each point represents the running times for MeTeoR (vertical coordinate) and the baseline (horizontal coordinate) for particular pairs of a program and a dataset; MeTeoR consistently outperformed the baseline.

Usage of materialisation and automata in MeTeoR Our previous experiments provide strong indication that the optimised seminaïve materialisation algorithm

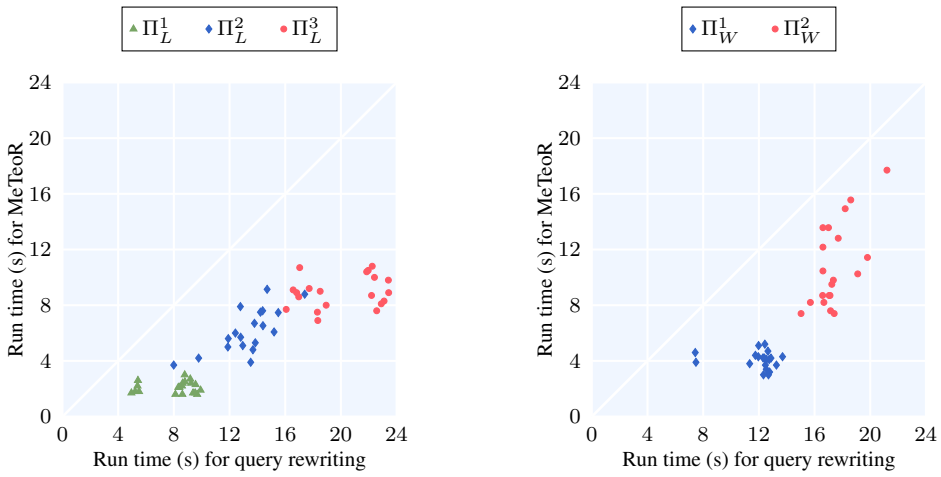


Figure 3.5: Comparison between MeTeoR and the query rewriting baseline

implemented in MeTeoR is more scalable than the automata-based component and that it can deal with complex rules and datasets containing millions of temporal facts. Next, we assess whether Algorithm 4 is indeed effective in delegating most of the reasoning workload to the scalable materialisation component. To this end, we considered programs $\Pi_I^1, \ldots, \Pi_I^{10}$ generated by iTemporal together with their corresponding datasets with 1000 facts each. For each pair of a program and a dataset we randomly generated 100 query facts and checked, for which facts entailment can be checked by MeTeoR using materialisation (i.e., Algorithm 4 terminates in either Line 3 or Line 8) and for which by the automata component (i.e., Algorithm 4 terminates in Line 12). In 99.7% of the cases MeTeoR decided entailment using materialisation only with an average runtime of $0.17s$ and a standard deviation of $0.089s$. In turn, the automata component was used in only 0.3% cases, with an average runtime of 181.9s and a standard deviation of 0.089s.

Although these results are obviously biased by the way iTemporal generates programs and datasets, they do support our hypothesis that focusing on the materialisation component in MeTeoR can provide a efficient reasoning mechanism.

4 | Pure Materialisation-based Reasoning in Interval-bounded DatalogMTL

In the previous chapter, we proposed a complete and sound reasoning algorithm in the full DatalogMTL language, which offers an effective way of combining the scalability of materialisation-based reasoning and the completeness guaranteed by automata-based procedure. While the construction of automata guarantees completeness in situations where materialisation might not terminate, it does come with a notable drawback: the constructed automata are of exponential size. As demonstrated by experiments (Wang, Hu, Wałęga, and Cuenca Grau, 2022), this can lead to a significant performance degradation. Consequently, an alternative strategy is to explore the design of reasoning algorithms for DatalogMTL that rely exclusively on materialisation, eliminating the need for automata construction.

In this chapter, we investigate a pure materialisation-based reasoning algorithm, where partial representations of the canonical interpretation are acquired through iterative rounds of rule applications. We focus on bounded DatalogMTL programs and datasets, where ∞ is not mentioned as an endpoint of any interval; this is a natural and expressive fragment, in which reasoning is as hard as in the unrestricted language (Wałęga, Zawidzki, and Cuenca Grau, 2021). The exclusion of ∞ provides the possibility to determine the definite depth of a rule which determines the time points that can be 'affected' by an application of this rule. As we will show next, within the context of DatalogMTL with bounded intervals, we are able to obtain the periodic structure of a canonical interpretation based on a partial materialisation, instead of executing a potentially infinite number of materialization steps.

A comprehensive examination of the theoretical underpinnings could be found in

Wałęga, Zawidzki, Wang, and Cuenca Grau (2023). This chapter is primarily dedicated to expounding upon the implementation details, underscoring the pragmatic realization of the aforementioned theoretical framework.

4.1 Periodic Structures

Given a bounded program Π and a dataset $\mathcal{D}$, saturation conditions state the properties that a partial materialisation $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ needs to satisfy so that the full canonical interpretation can be recovered. An interpretation satisfying these conditions is called *saturated*. For convenience of presentation, we fix for the remainder of this section an arbitrary bounded program Π, bounded dataset $\mathcal{D}$, and natural number k. Moreover, for an interpretation $\mathfrak{I}$ and an interval ϱ, we let the *projection* $\mathfrak{I}\,|_\varrho$ of $\mathfrak{I}$ over ϱ be the interpretation that coincides with $\mathfrak{I}$ on ϱ and makes all relational atoms false outside ϱ. We say that an interpretation $\mathfrak{I}'$ is a *shift* of $\mathfrak{I}$ if there is a rational number q such that $\mathfrak{I} \models M@\varrho$ if and only if $\mathfrak{I}' \models M@(\varrho + q)$, for each (relational) fact $M@\varrho$. Furthermore, for a rule r, we define its *depth*, written as $\mathsf{depth}(r)$, as the sum of right endpoints of all intervals occurring in the operators of r (or 0 if r mentions no intervals), and we let $\mathsf{depth}(\Pi)$ be the maximum depth of its rules. As shown in the following proposition, the depth of a bounded rule determines the time points that can be 'affected' by an application of this rule.

Proposition 16 *For every interpretation $\mathfrak{I}$, time point t, and bounded rule r, it holds that $\mathfrak{I}$ satisfies r at t if and only if so does $\mathfrak{I}\,|_{[t-\mathsf{depth}(r),t+\mathsf{depth}(r)]}$.*

Having introduced these basic concepts, we are now ready to define our notion of a saturated interpretation.

Definition 17 *Interpretation $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ is* saturated *if there exist closed intervals ϱ_1, ϱ_2, ϱ_3, and ϱ_4 of length $2\mathsf{depth}(\Pi)$, whose endpoints are located on the $(\Pi, \mathcal{D})$-*

ruler and satisfy $\varrho_1^+ < \varrho_2^+ < t_{\mathcal{D}}^-$ *and* $t_{\mathcal{D}}^+ < \varrho_3^- < \varrho_4^-$*, and s.t. the following properties hold:*

- $T_{\Pi}^k(\mathfrak{I}_{\mathcal{D}})$ *satisfies* Π *in* $[\varrho_1^-, \varrho_4^+]$*;*
- $T_{\Pi}^k(\mathfrak{I}_{\mathcal{D}}) \mid_{\varrho_1}$ *and* $T_{\Pi}^k(\mathfrak{I}_{\mathcal{D}}) \mid_{\varrho_3}$ *are shifts of* $T_{\Pi}^k(\mathfrak{I}_{\mathcal{D}}) \mid_{\varrho_2}$ *and* $T_{\Pi}^k(\mathfrak{I}_{\mathcal{D}}) \mid_{\varrho_4}$*, respectively.*

Any pair of intervals $[\varrho_1^-, \varrho_2^-)$ *and* $(\varrho_3^+, \varrho_4^+]$*, for* ϱ_1*,* ϱ_2*,* ϱ_3*,* ϱ_4 *as above, will be referred to as* periods $(\varrho_{\mathsf{left}}, \varrho_{\mathsf{right}})$ *of* $T_{\Pi}^k(\mathfrak{I}_{\mathcal{D}})$.

Intuitively, a saturated interpretation contains a 'central fragment' $[\varrho_1^-, \varrho_4^+]$ which satisfies $\mathcal{D}$ and such that a single application of T_{Π} does not derive any new facts within this fragment (i.e., the interpretation satisfies Π within $[\varrho_1^-, \varrho_4^+]$). In the left segment of this 'central fragment' there are two intervals ϱ_1 and ϱ_2—each of length $2\mathsf{depth}(\Pi)$—in which $T_{\Pi}^k(\mathfrak{I}_{\mathcal{D}})$ satisfies the same relational facts modulo a shift. Analogously, in the right segment of the 'central fragment' there are intervals ϱ_3 and ϱ_4 also with repeating contents.

It is worth emphasising that in the definition of a saturated interpretation, we consider only a single materialisation step, which can be effectively checked. As shown in Theorem 19, rather surprisingly, this condition is enough to guarantee that a saturated interpretation can be unfolded into the canonical interpretation.

We next define the unfolding of a saturated interpretation relatively to a pair $(\varrho_{\mathsf{left}}, \varrho_{\mathsf{right}})$ of its periods. Although there can be many such pairs of intervals, the key properties of the unfolding are independent of the choice of periods.

Definition 18 *The* $(\varrho_{\mathsf{left}}, \varrho_{\mathsf{right}})$*-unfolding, of a saturated interpretation* $T_{\Pi}^k(\mathfrak{I}_{\mathcal{D}})$ *with periods* $(\varrho_{\mathsf{left}}, \varrho_{\mathsf{right}})$*, is the interpretation* $\mathfrak{C}$ *such that:*

- $\mathfrak{C} \mid_{[\varrho_{\mathsf{left}}^-, \varrho_{\mathsf{right}}^+]} = T_{\Pi}^k(\mathfrak{I}_{\mathcal{D}}) \mid_{[\varrho_{\mathsf{left}}^-, \varrho_{\mathsf{right}}^+]}$,

– $\mathfrak{C}\,|_{\varrho_{\mathsf{left}}-n\cdot|\varrho_{\mathsf{left}}|}$ *is a shift of* $T^k_\Pi(\mathfrak{I}_\mathcal{D})\,|_{\varrho_{\mathsf{left}}}$*, for any* $n \in \mathbb{N}$*,*

– $\mathfrak{C}\,|_{\varrho_{\mathsf{right}}+n\cdot|\varrho_{\mathsf{right}}|}$ *is a shift of* $T^k_\Pi(\mathfrak{I}_\mathcal{D})\,|_{\varrho_{\mathsf{right}}}$*, for any* $n \in \mathbb{N}$*.*

Unfolding is unique and can be obtained from the least interpretation coinciding with $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ on $[\varrho^-_{\mathsf{left}}, \varrho^+_{\mathsf{right}}]$ by subsequently copying the fragment of this interpretation spanning ϱ_{left} infinitely many times to the left and the fragment of this interpretation spanning ϱ_{right} infinitely many times to the right on the timeline.

Theorem 19 *The* $(\varrho_{\mathsf{left}}, \varrho_{\mathsf{right}})$*-unfolding* $\mathfrak{C}$ *of a saturated interpretation* $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ *with periods* $(\varrho_{\mathsf{left}}, \varrho_{\mathsf{right}})$ *coincides with the canonical interpretation* $\mathfrak{C}_{\Pi,\mathcal{D}}$*.*

Theorem 19 states that the unfolding of a saturated interpretation around any pair of its periods coincides with the canonical interpretation. Thus, we can refer to an unfolding of a saturated interpretation without explicitly referring to its periods.

Lastly, by establishing a bound k_{max}, depending on Π and $\mathcal{D}$, it ensures that $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ is saturated for some $k \leq k_{\mathsf{max}}$. Intuitively, k_{max} is the number of facts that we can construct by combining atoms relevant to Π and $\mathcal{D}$ with $(\Pi, \mathcal{D})$-intervals contained in a sufficiently large interval ϱ. The interval ϱ is chosen so that $[t^-_\mathcal{D}, t^+_\mathcal{D}] \subseteq \varrho$, and such that its fragments $[\varrho^-, t^-_\mathcal{D})$ and $(t^+_\mathcal{D}, \varrho^+]$ to the left and right of $\mathcal{D}$, respectively, contain so many intervals of length $2\mathsf{depth}(\Pi)$ with endpoints on the $(\Pi, \mathcal{D})$-ruler, that their contents need to repeat. These repetitions guarantee the existence of intervals $\varrho_1, \ldots, \varrho_4$ from Definition 17.

Theorem 20 *There exists* $k \leq k_{\mathsf{max}}$ *such that* $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ *is saturated, where the bound* k_{max} *is defined as follows. Let* A *be the number of ground relational atoms in the grounding of* Π *with constants from* Π *and* $\mathcal{D}$*, let* B *be the number of* $(\Pi, \mathcal{D})$*-intervals within* $[t^-_\mathcal{D}, t^-_\mathcal{D} + 2\mathsf{depth}(\Pi)]$*, and let* $\varrho = [t_{-w} - 2\mathsf{depth}(\Pi), t_w + 2\mathsf{depth}(\Pi)]$*, where* $\cdots < t_{-2} < t_{-1} < t^-_\mathcal{D}$ *and* $t^+_\mathcal{D} < t_1 < t_2 < \ldots$ *are se-*

Algorithm 5: Reasoning via Periods Detection

Input: a bounded program Π, a bounded dataset $\mathcal{D}$, and a relational fact $M@\varrho$

Output: a Boolean value True or False

```
1 D_now := D;
2 loop
3     if D_now ⊨ M@ϱ then return True;
4     (ϱ_left, ϱ_right) := Periods(Π, D, D_now);
5     if ϱ_left ≠ ∅ and ϱ_right ≠ ∅ then
6         if Entails(Π, D, D_now, ϱ_left, ϱ_right, M@ϱ) then return True;
7         else return False;
8     D_now := ApplyRules(Π, D_now);
```

quences of consecutive time points on the $(\Pi, \mathcal{D})$*-ruler, whil* $w = 1 + B \cdot (2^A)^B$*. Then* k_{max} *is the product of* A *and the number of* $(\Pi, \mathcal{D})$*-intervals contained in* ϱ*.*

The bound k_{max} from Theorem 20 is doubly exponential in the size of Π and $\mathcal{D}$, but only exponential in the size of $\mathcal{D}$. This bound shows us how much time is needed to perform reasoning, which is consistent with the computational complexity of reasoning in DatalogMTL, namely ExpSpace in combined complexity and PSpace in data complexity (Wałęga, Cuenca Grau, Kaminski, and Kostylev, 2019).

4.2 Reasoning via Periods Detection

The process of obtaining the *periods* $(\varrho_{\mathsf{left}}, \varrho_{\mathsf{right}})$ of $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ is represented as Periods, whose input are a bounded program Π, a bounded dataset $\mathcal{D}$, and a dataset $\mathcal{D}_{\mathsf{now}}$ representing $T^k_\Pi(\mathfrak{I}_\mathcal{D})$, for some $k \in \mathbb{N}$. The materialisation-based reasoning procedure is shown in Algorithm 5, which checks if a bounded program Π and a dataset $\mathcal{D}$ entail a relational fact $M@\varrho$.

The algorithm successively applies the rules of Π to the dataset (Line 8) until one of two stopping conditions hold. The first condition (Line 3) detects if the

materialisation $\mathcal{D}_{\mathsf{now}}$ constructed thus far entails the input fact $M@\varrho$, in which case the algorithm reports that the input fact is entailed. The second condition checks if $\mathcal{D}_{\mathsf{now}}$ is saturated (i.e., represents a saturated interpretation). To this end, the algorithm computes in Line 4 two non-empty intervals which are periods of $\mathcal{D}_{\mathsf{now}}$ (if $\mathcal{D}_{\mathsf{now}}$ is saturated), or introduces two empty intervals (if $\mathcal{D}_{\mathsf{now}}$ is not saturated). If $\mathcal{D}_{\mathsf{now}}$ is saturated, the algorithm exploits $\mathcal{D}_{\mathsf{now}}$ and its periods to check whether $M@\varrho$ is entailed (Lines 6–7).

In what follows, we fix an arbitrary input Π, $\mathcal{D}$, and $M@\varrho$ to Algorithm 5 and assume that the notions of a saturated interpretation and its periods are relative to Π and $\mathcal{D}$. We also let the projection $\mathcal{D}' \restriction_{\varrho'}$ of a dataset $\mathcal{D}'$ over an interval ϱ' be the dataset obtained by intersecting all intervals in facts from $\mathcal{D}'$ with ϱ'.

In each iteration of the loop, Algorithm 5 applies ApplyRules (Line 8), which implements a single round of rule applications to $\mathcal{D}_{\mathsf{now}}$.

First Stopping Condition In Line 3, Algorithm 5 checks whether the materialisation $\mathcal{D}_{\mathsf{now}}$ constructed so far entails the input fact $M@\varrho$. Since facts in $\mathcal{D}_{\mathsf{now}}$ are stored in a coalesced form, to check if $\mathcal{D}_{\mathsf{now}} \models M@\varrho$, it suffices to scan $\mathcal{D}_{\mathsf{now}}$ and verify whether there is $M@\varrho' \in \mathcal{D}_{\mathsf{now}}$with $\varrho \subseteq \varrho'$.

Second Stopping Condition Algorithm 5 calls the Periods procedure in Line 4 to check, in an iteration $k+1$ of the loop, if $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ is saturated. The procedure searches for intervals $\varrho_1, \ldots, \varrho_4$ satisfying the conditions in Definition 17 and returns the periods of the interpretation $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ if it is saturated or a pair of empty intervals otherwise. To this end, the procedure performs one round of rule applications to $\mathcal{D}_{\mathsf{now}}$, computing $\mathcal{D}_{\mathsf{next}}$. Then, it computes an interval ϱ_{max} as either the empty interval (if $\mathcal{D}_{\mathsf{now}}$ and $\mathcal{D}_{\mathsf{next}}$ do not coincide on $[t^-_\mathcal{D}, t^+_\mathcal{D}]$), or otherwise as the maximal interval containing $[t^-_\mathcal{D}, t^+_\mathcal{D}]$ and such that $\mathcal{D}_{\mathsf{now}}$ and $\mathcal{D}_{\mathsf{next}}$ coincide on

ϱ_{max}. Interval ϱ_{max} is next used to search for ϱ_1 and ϱ_2, namely all pairs of intervals contained in ϱ_{max}, located to the left of $t^-_{\mathcal{D}}$, with endpoints on the $(\Pi, \mathcal{D})$-ruler, and of lengths $2\mathsf{depth}(\Pi)$, are compared. The first pair of such intervals with the same contents in $\mathcal{D}_{\mathsf{now}}$ is set as ϱ_1 and ϱ_2; otherwise ϱ_1 and ϱ_2 are empty intervals. Intervals ϱ_3 and ϱ_4 are similarly computed. Finally, the procedure outputs a pair of intervals $([\varrho_1^-, \varrho_2^-), (\varrho_3^+, \varrho_4^+])$, or $(\emptyset, \emptyset)$ if any of the intervals $\varrho_1, \ldots, \varrho_4$ is empty.

Fact Entailment Checking after Saturation After the construction of a saturated dataset $\mathcal{D}_{\mathsf{now}}$ (denoting $T^k_\Pi(\mathfrak{I}_\mathcal{D})$, for some $k \in \mathbb{N}$) with periods $(\varrho_{\mathsf{left}}, \varrho_{\mathsf{right}})$, Algorithm 5 calls the procedure Entails (Line 6) to check whether the input fact $M@\varrho$ holds in the unfolding of $T^k_\Pi(\mathfrak{I}_\mathcal{D})$.

Based on the theoretical results described in the previous section, it follows that the algorithm is sound and complete:

Theorem 21 *Algorithm 5 outputs* True *if* $(\Pi, \mathcal{D}) \models M@\varrho$*, otherwise it outputs* False*. Moreover, the algorithm terminates after at most* $k_{\mathsf{max}} + 1$ *(c.f. Theorem 20) iterations of its main loop.*

We conclude this section with an example illustrating the execution of Algorithm 5, namely, how to obtain the periodic structure.

Example 22 *Consider* $\Pi = \{\boxplus_{[0,1]} P \leftarrow P, \boxminus_{[1,1]} Q \leftarrow Q\}$, $\mathcal{D} = \{P@0, Q@1.5\}$, *and a query fact* $M@t = Q@{-4.5}$*. After 5 iterations of the loop in Algorithm 5,* $\mathcal{D}_{\mathsf{now}}$ *consists of facts* $P@[0, 5]$ *and* $Q@t$*, for all* $t \in \{-3.5, -2.5, -1.5, -0.5, 0.5, 1.5\}$*. The stopping condition from Line 3 does not hold, but the condition in Line 5 does. Indeed,* $\mathcal{D}_{\mathsf{now}}$ *is saturated and its periods computed in Line 4 are* $\varrho_{\mathsf{left}} = [-3.5, -2.5)$ *and* $\varrho_{\mathsf{right}} = (4.0, 4.5]$*. The unfolding of* $\mathcal{D}_{\mathsf{now}}$ *is the canonical interpretation of* Π *and* $\mathcal{D}$ *in Figure 4.1. In Line 6, Algorithm 5 can*

exploit $\mathcal{D}_{\mathsf{now}}$, ϱ_{left}, *and* ϱ_{right} *to detect entailment of any fact. In particular, the input fact* $Q@-4.5$ *is entailed, and so, Algorithm 5 returns* True *in Line 6.*

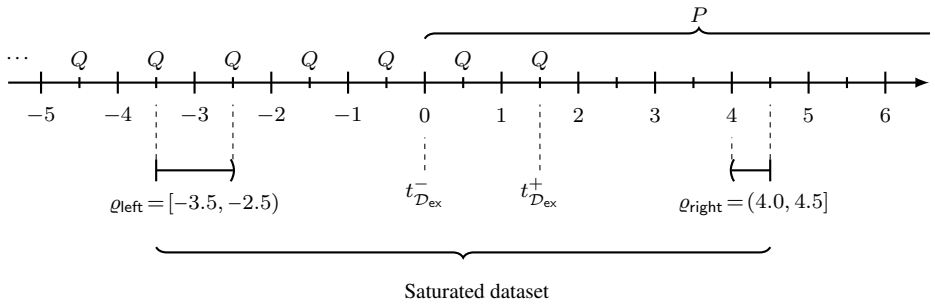


Figure 4.1: Canonical interpretation $\mathfrak{C}_{\Pi,\mathcal{D}}$ of Π and $\mathcal{D}$

4.3 Evaluation

We have implemented Algorithm 5 and conducted two experiments. The first experiment compares the pure materialisation-based reasoning approach with the combined approach introduced in the last chapter. In particular, we did not consider other reasoners, such as Ontop (Kalaycı et al., 2018), as they support only non-recursive programs, where canonical interpretations are always finite. The second experiment tests the scalability of our implementation on datasets of increasing size. We used the as benchmarks Temporal LUBM and and iTemporal.

Comparison with the baseline We compared our implementation with MeTeoR, which combines materialisation with automata-based reasoning techniques. For the LUBM benchmark, Figure 4.2 (left), we considered a dataset $\mathcal{D}$ with 5 million facts and query facts $F@t$, where F is a fixed relational atom (about the predicate *FullProfessor*) and $t \in \{-500, -400, \ldots, 500\}$. Reasoning with $\mathcal{D}$ and the fragment Π of the benchmark's program that is relevant for F is non-trivial and materialisation does not terminate for them.

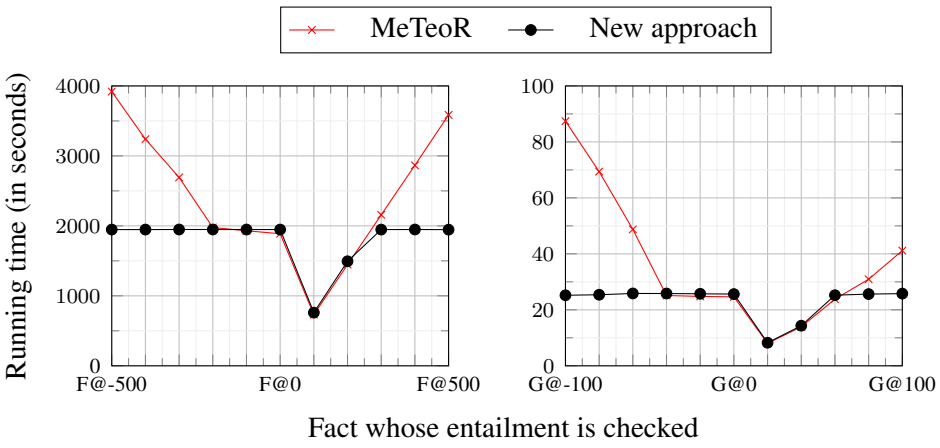


Figure 4.2: Comparison with MeTeoR on the LUBM (on the left) and iTemporal (on the right) benchmarks

Facts $F@t$ with $t > 0$ are entailed by Π and $\mathcal{D}$, so such facts are decided by MeTeoR using materialisation only. Thus, the larger t, the larger the number of materialisation steps the baseline performs; as shown in Figure 4.2, materialisation times increase linearly with t. In our new approach, we observe a similar linear growth up until $t = 300$, where saturation is achieved. This indicates that saturation checks during materialisation come with negligible overhead. Once saturation is reached, our approach does not require further materialisation steps and fact entailment can be decided based on the saturated dataset. Thus, fact entailment running times are roughly identical for all facts with $t \geq 300$, while for MeTeoR they continue to grow with t.

Facts $F@t$, with $t < 0$ are not entailed, hence MeTeoR must resort to automata-based techniques. Due to the optimisations implemented in MeTeoR, automata-based reasoning works well for facts with time points that lie close to the dataset, but performance degrades as t becomes located further away. In contrast, our approach proceeds as in the previous case; it will materialise in linear running time until a saturated dataset is obtained, at which point running times stabilise

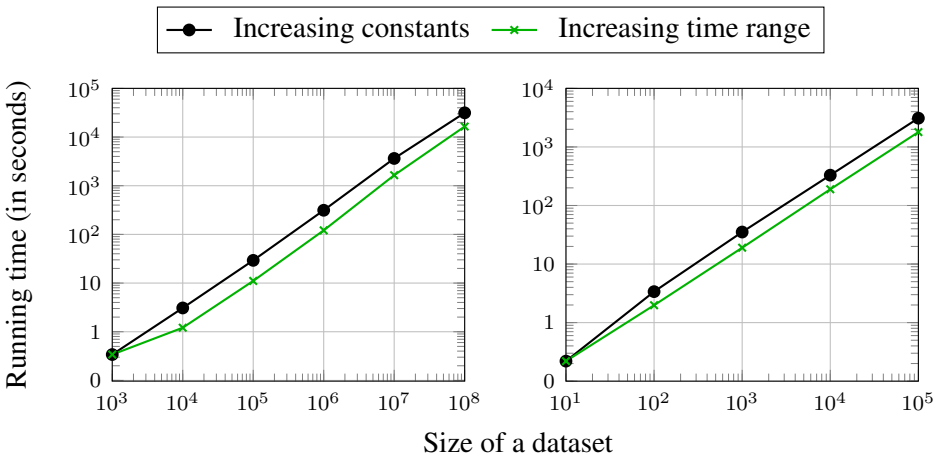


Figure 4.3: Scalability of our approach tested on the LUBM (on the left) and iTemporal (on the right) benchmarks

and become independent of t.

In the case of the iTemporal benchmark, we considered a dataset with 1000 facts and facts $G@t$ with a specific atom G and $t \in \{-100, -80, \ldots, 100\}$; we observed the same type of behaviour as for LUBM (see Figure 4.2, right).

Scalability in data size In the second experiment, we analysed how saturation times increase with the size of the input dataset. We considered the same programs as in the first experiment, together with sequences of datasets of increasing size. For the LUBM benchmark, we generated two sequences, each containing 6 datasets. In the first sequence all the intervals of temporal facts are contained within the range $[0, 50]$ and we increase the size of datasets by introducing atoms with new constants. In the second sequence, the number of constants is the same but the number of facts increases; facts occupy increasing ranges of time, namely, $[0, 5 \cdot 10^i]$, for $i \in \{1, \ldots, 6\}$. In both sequences, the datasets have 10^3, 10^4, 10^5, 10^6, 10^7, and 10^8 facts, respectively.

As depicted in Figure 4.3 (left), in both cases running times grow proportionally

with data size; this suggests that our approach is scalable with respect to both the number of constants and the temporal range of the dataset. We observe that the running times for the second sequence of facts are around twice smaller than for the first sequence (except the first dataset which is the same in both sequences), so increasing the number of constants seems to have a larger impact on our approach than increasing the temporal domain. We performed a similar experiment for the iTemporal benchmark (see Figure 4.3, right), which confirmed our observations from the LUBM experiment. One possible reason is that increasing the time range increases the likelihood of merging multiple derived facts into a single fact. In contrast, facts with different constants cannot be merged. Therefore, increasing the number of distinct constants has a greater impact on our approach than extending the temporal domain in terms of running time.

5 | Stream Reasoning in Forward-propagating DatalogMTL

Stream reasoning, the problem of query answering over data streams in the presence of background knowledge, has received a great deal of attention in the literature (Barbieri, Braga, Ceri, Della Valle, and Grossniklaus, 2009; Anicic, Fodor, Rudolph, and Stojanovic, 2011; Dell'Aglio, Valle, Calbimonte, and Corcho, 2014; Özçep, Möller, and Neuenstadt, 2014; Margara, Urbani, van Harmelen, and Bal, 2014; Zaniolo, 2012; Ronca et al., 2018; Beck, Dao-Tran, and Eiter, 2018; Ronca, Kaminski, Grau, and Horrocks, 2022). Different from the standard query answering, where the input data is typically assumed to be static and finite, streaming data is regarded as an infinite and unceasing sequence of facts coursing through the system, demanding incremental processing. Metric temporal logic (MTL) has been proposed as a suitable formalism for specifying and reasoning in real time over such complex events (Heintz and Doherty, 2004; Thati and Roşu, 2005; Doherty, Kvarnström, and Heintz, 2009; Ničković and Piterman, 2010; Baldor and Niu, 2012; Ho, Ouaknine, and Worrell, 2014; Basin, Klaedtke, and Zălinescu, 2018). However, MTL satisfiability checking cannot be used to capture analysis tasks involving recursive propagation of information; query answering in a temporal rule language such as DatalogMTL is better suited to that effect.

Example 23 *Consider a network where nodes are monitoring different signals. A node receiving readings for a signal with sufficiently high frequency flags the signal by sending a message to all neighbouring nodes in the network, which will then also monitor the signal for a fixed period of time. The analyst is interested in tracking which nodes are monitoring which signals at any point in time. This generic monitoring task involves analysing how information propagates recur-*

sively over time throughout the topology of the network, and can be captured by the following DatalogMTL *rules:*

$$Flag(x, z) \leftarrow Monit(x, z) \wedge \boxminus_{[0,4]} \diamondminus_{[0,2]} Signal(z), \tag{5.1}$$

$$\boxplus_{[0,3]} Monit(x, z) \leftarrow Flag(y, z) \wedge Connect(x, y). \tag{5.2}$$

Rule (5.1) states that a node monitoring a signal will flag it whenever the signal has been received continuously over an interval of length at least 4 *with gaps between consecutive readings of length at most* 2*. In turn, Rule (5.2) ensures that each signal* z*, flagged by a node* y*, will be monitored in the future by all nodes* x *directly connected to* y *in the network for a period of length at least* 3*.*

Wałęga, Kaminski, and Cuenca Grau (2019) introduced a sound and complete stream reasoning algorithm designed for the *forward-propagating* DatalogMTL programs. *Forward-propagating* rules are syntactically restricted to preclude propagation of derived information towards past time points. In this chapter, our primary emphasis lies in introducing the expanded contributions derived from Wałęga, Kaminski, and Cuenca Grau (2019), specifically centering on the implementation of a streaming reasoning system. Our experiments show that our implemented stream reasoning system is able to keep in memory only a very limited number of facts at each point in time. Initially, memory consumption increases rapidly as the first window is populated and new facts are derived; however, as soon as the window starts sliding and the algorithm starts forgetting old facts that are no longer relevant, memory consumption stabilises and remains essentially constant from then onwards.

5.1 Preliminaries

In this section, we provide a recapitulation of key notions to streaming reasoning. Notably, some definitions used in this chapter have been previously established.

Time A set $\mathbb{T} \subseteq \mathbb{Q}$ of time points is discrete if each non-maximal $t \in \mathbb{T}$ has a *successor* $\mathsf{succ}_{\mathbb{T}}(t)$ in $\mathbb{T}$; that is, if $\mathsf{succ}_{\mathbb{T}}(t) \in \mathbb{T}$, $t < \mathsf{succ}_{\mathbb{T}}(t)$, and there is no $t' \in \mathsf{succ}_{\mathbb{T}}(t)$ such that $t < t' < t < \mathsf{succ}_{\mathbb{T}}(t)$. For definiteness, we let $\mathsf{succ}_{\mathbb{T}}(t) = \infty$ if t is the maximal element of $\mathbb{T}$. The *greatest common divisor* of a set $\mathbb{T} \subseteq \mathbb{Q}$ of rational numbers, written as $\mathrm{gcd}(\mathbb{T})$, is the greatest rational number which divides all the numbers in $\mathbb{T}$ to integer values (or it is not defined, if such divisor does not exist). The existence of a gcd for a set of time points ensures existence of a minimal distance between them. Indeed, in (infinite) sets without a gcd, such as $\{0, \frac{1}{2}, \frac{3}{4}, \frac{7}{8}, \ldots\}$, there is no minimal distance between numbers.

Interval operations As described in Section 2.2.1, an *interval* ϱ is a non-empty subset of $\mathbb{Q}$, where ϱ^- and ϱ^+ are the *left* and *right endpoints* of ϱ, respectively. For two intervals ϱ and ϱ' we define the following operations, which will be useful to concisely formulate our algorithms:

$$\begin{aligned}
\varrho + \varrho' &= \{t + t' \mid t \in \varrho \text{ and } t' \in \varrho'\},\\
\varrho - \varrho' &= \{t - t' \mid t \in \varrho \text{ and } t' \in \varrho'\},\\
\varrho \oplus \varrho' &= \{t \mid (\{t\} - \varrho') \subseteq \varrho\};
\end{aligned}$$

for example, $[0, 10] + [2, 4] = [0 + 2, 10 + 4] = [2, 14]$, $[0, 10] - [2, 4] = [0 - 4, 10 - 2] = [-4, 8]$, whereas $[0, 10] \oplus [2, 4] = [0 + 4, 10 + 2] = [4, 12]$.

5.2 Streams and Stream Queries

Streaming data is often represented as an unbounded sequence of timestamped facts. Similarly to Brandt et al. (2018), we represent timestamps as non-negative rational numbers; this generalises other models of streaming data in the literature where timestamps are assumed to be natural numbers (Beck, Dao-Tran, and Eiter, 2018; Ronca et al., 2018). We also assume that the timeline of a stream is discrete as opposed to dense—that is, it is always possible to tell which is the next time point for which data is available in the stream.

Definition 24 *A* stream *is a function $\mathcal{S}$ assigning a (possibly empty) finite set of ground relational atoms to each $t \in \mathbb{Q}_{\geq 0}$. The* temporal domain $\mathbb{T}_{\mathcal{S}}$ *of $\mathcal{S}$ is the set of time points t satisfying $\mathcal{S}(t) \neq \emptyset$; we require that the temporal domain of every stream is a discrete set. Moreover, we say that $\mathcal{S}$ is* regular *if $\gcd(\mathbb{T}_{\mathcal{S}})$ is well-defined. The* object domain $\mathbb{O}_{\mathcal{S}}$ *of $\mathcal{S}$ is the set of all constants occurring in the values of $\mathcal{S}$. By slight abusing notation, we identify a stream $\mathcal{S}$ with the (possibly infinite) set of facts $\{M@t \mid t \in \mathbb{T}_{\mathcal{S}}$ and $M \in \mathcal{S}(t)\}$.*

A stream query defines a transformation of an input stream into an output stream by means of logical entailment.

Definition 25 *A* stream query *is a pair (Π, Q) consisting of a program Π and a predicate Q. The* answer *to (Π, Q) over a stream $\mathcal{S}$, relative to a discrete set $\mathbb{T} \subseteq \mathbb{Q}_{\geq 0}$, is the stream $\{Q(\mathbf{c})@t \mid t \in \mathbb{T}$ and $(\Pi, \mathcal{S}) \models Q(\mathbf{c})@t\}$.*

A query (Π, Q) is forward-propagating (c.f. Section 2.2.3) if so is Π. Note that set $\mathbb{T}$ in the previous definition specifies the time points for which answers are required in the output; thus, $\mathbb{T}$ contains the temporal domain of the answer stream.

5.3 Stream Reasoning Algorithm

Algorithm 6: Generic stream reasoning algorithm

Parameters: A stream query (Π, Q) with a forward-propagating program Π in normal form, a discrete set $\mathbb{T} \subseteq \mathbb{Q}_{\geq 0}$, and $\mathsf{step} \in \mathbb{Q}_{>0}$

Input: A stream $\mathcal{S}$

1 $t := -\mathsf{step}$; // current time point
2 $\mathcal{W} := \emptyset$; // window contents
3 $\mathcal{H} := \emptyset$; // history contents
4 $t_\Pi :=$ maximal rational endpoint in intervals of Π (or 0 if Π does not mention rational numbers);
5 **loop**
6 **if** $\mathsf{succ}_\mathcal{S}(t) \leq t + \mathsf{step}$ **then**
7 $t_\mathsf{next} := \mathsf{succ}_\mathcal{S}(t)$;
8 $\mathcal{W} := \mathcal{W} \cup \mathcal{S}(t_\mathsf{next})$;
9 **else**
10 $t_\mathsf{next} := t + \mathsf{step}$;
11 $\mathcal{W} := \mathsf{ApplyRules}^{str}(\Pi, \mathcal{W}, \mathcal{H}, t, t_\mathsf{next})$;
12 **for** *each* $Q(\mathbf{c})@\varrho \in \mathcal{W}$ **do**
13 **for** *each* $t' \in \varrho \cap (t, t_\mathsf{next}] \cap \mathbb{T}$ **do**
14 **stream out** $Q(\mathbf{c})@t'$;
15 $t := t_\mathsf{next}$;
16 $(\mathcal{W}, \mathcal{H}) := \mathsf{Forget}(\mathcal{W}, \mathcal{H}, t, t_\Pi)$;

In this section, we present a generic stream reasoning algorithm applicable to forward-propagating stream queries. Our procedure (Algorithm 6) takes as input a stream $\mathcal{S}$ and outputs, also as a stream, the answers to a standing forward-propagating query (Π, Q). The query is assumed to be in normal form (c.f. Section 2.2.3) and fixed, as a parameter. The algorithm is also parametrised with a discrete set $\mathbb{T}$, which specifies the temporal domain of the answer stream, and a value step which controls the incrementality of query processing.

To simplify the presentation, and without loss of generality, we assume that the input stream $\mathcal{S}$ mentions only predicates occurring in Π (facts involving other

predicates are irrelevant for answering the query) and that all constants from Π occur in $\mathcal{S}$ (this can be easily achieved by extending $\mathcal{S}$ with facts explicitly mentioning these constants).

Algorithm 6 is initialised in Lines 1–4; Line 1 initialises the current time point t in the input stream to a value smaller than all time points in $\mathbb{T}_{\mathcal{S}}$ (note that we initialise t as $-\mathsf{step}$, but we could use any value smaller than the first time point in the input stream), and Line 4 sets t_Π to the maximum rational number occurring as an interval endpoint in program Π. Moreover, Lines 2 and 3 initialise the memory contents, which consist of the following two parts.

- The *window* $\mathcal{W}$—a set of relational facts storing information about atoms holding in the most recent interval (in particular, in the interval $[t - t_\Pi, t]$).
- The *history* $\mathcal{H}$—a set of relational atoms storing information about atoms which hold further back into the past than the recent window covered by $\mathcal{W}$, without specifying exactly when.

The core of the algorithm is an infinite loop, where each iteration consists of the steps detailed next, and where the current time point t is incremented at the end of each iteration. In Lines 6–10, the algorithm processes the next fragment of the input stream by either moving to the next time point in the stream containing data and loading the relevant facts into the window memory, or moving forward by step (if there is no data point between t and $t + \mathsf{step}$). In Line 11, the algorithm updates $\mathcal{W}$ by applying to them rules from Π which derive new information within the interval $(t, t_{\mathsf{next}}]$. In Lines 12–14 the algorithm outputs all answers to the query in-between t and t_{next}. Finally, in Line 16 the algorithm trims the window $\mathcal{W}$ by forgetting information that is no longer relevant and updates the history $\mathcal{H}$.

Next, we describe in detail the procedures $\mathsf{ApplyRules}^{str}$ and Forget from Lines 11 and 16, respectively. Procedure $\mathsf{ApplyRules}^{str}$ is presented in Algorithm 7. It takes

as input a program Π, a window $\mathcal{W}$, a history $\mathcal{H}$, and two time points t and t_{next}, to compute updated $\mathcal{W}$. In particular, $\mathsf{ApplyRules}^{str}$ in the stream reasoning setting is to successively applies rules of Π to $\mathcal{W}$ and $\mathcal{H}$ (Lines 2–3) to derive all the facts that hold within the interval $(t, t_{\mathsf{next}}]$. If no more rules can be applied (Line 17), the procedure terminates and returns updated $\mathcal{W}$ (Line 18). The rules in Π of Form (2.3) are applied in Lines 3–7; rules of Form (2.4) in Lines 8–10; and rules of Form (2.5) are applied in Lines 11–13. In Lines 14–16, facts in $\mathcal{W}$ mentioning the same atoms are coalesced by merging together facts over adjacent intervals.

Example 26 *Consider Algorithm 7 running on a program* Π *consisting of the following rules:*

$$P(z) \leftarrow \diamondminus_{[0,2]} Signal(z), \tag{5.3}$$

$$P'(z) \leftarrow \boxminus_{[0,4]} P(z), \tag{5.4}$$

$$Flag(x, z) \leftarrow Monit(x, z) \wedge P'(z), \tag{5.5}$$

empty history $\mathcal{H}$*, window* $\mathcal{W}$ *with facts*

$$Signal(s_1)@96.3, \qquad P(s_1)@[97, 100],$$
$$Signal(s_1)@98, \qquad Monit(n, s_1)@101,$$
$$Signal(s_1)@100,$$

and time points $t = 100$*,* $t_{\mathsf{next}} = 101$*, and* $t_\Pi = 4$*.*

Algorithm 7 extends $\mathcal{W}$ *with new facts. In Lines 3–7, Rule (5.3) is applied to the fact* $Signal(s_1)@100$*, and so,* $P(s_1)@(100, 101]$ *is added to* $\mathcal{W}$*.*

Facts $P(s_1)@[97, 100]$ *and* $P(s_1)@(100, 101]$ *are coalesced in Lines 14–16, and so the fact* $P(s_1)@[97, 101]$ *is added to* $\mathcal{W}$*. In Lines 8–10, Rule (5.4) is applied to*

Algorithm 7: ApplyRules^{str}

Input: A forward-propagating program Π in normal form, a set $\mathcal{W}$ of facts, a set $\mathcal{H}$ of atoms, and time points t, t_{next}

Output: A set of facts

```
1  repeat
2      for each r ∈ ground(Π, W ∪ H) do
3          if r is of the form M' ← ⟐_ϱ2 M then
4              for each M@ϱ1 ∈ W such that ϱ = (ϱ1 + ϱ2) ∩ (t, t_next] is
                non-empty do
5                  Add M'@ϱ to W;
6              if ϱ2 is unbounded and M ∈ H then
7                  Add M'@(t, t_next] to W;
8          if r is of the form M' ← ⊟_ϱ2 M then
9              for each M@ϱ1 ∈ W such that ϱ = (ϱ1 ⊕ ϱ2) ∩ (t, t_next] is
                non-empty do
10                 Add M'@ϱ to W;
11         if r is of the form M' ← M1 ∧ ··· ∧ Mn then
12             for all M1@ϱ1, ..., Mn@ϱn ∈ W such that ϱ = ϱ1 ∩ ··· ∩ ϱn is
                non-empty do
13                 Add M'@ϱ to W;
14     for each M@ϱ1, M@ϱ2 ∈ W do
15         if ϱ = ϱ1 ∪ ϱ2 is an interval then
16             Add M@ϱ to W;
17 until no more elements are added to W;
18 return W;
```

the fact $P(s_1)@[97, 101]$*, adding* $P'(s_1)@101$ *to* $\mathcal{W}$*.*

Next, Rule (5.5) is applied to facts $P'(s_1)@101$ *and* $Monit(n, s_1)@101$ *in Lines 11–13, and* $Flag(n, s_1)@101$ *is added to* $\mathcal{W}$*, as a result.*

We next describe the procedure Forget specified by Algorithm 8, which trims the window memory $\mathcal{W}$ by forgetting information that is no longer relevant and adds corresponding atoms to $\mathcal{H}$. In Lines 1–7, the algorithm 'slides' the window by

Algorithm 8: Forget

Input: A set $\mathcal{W}$ of facts, a set $\mathcal{H}$ of atoms, and time points t, t_Π
Output: A pair consisting of a set of facts and a set of atoms

```
1  for each M@ϱ ∈ W do
2      if ϱ ⊈ [t − t_Π, t] then
3          Add M to H;
4          if ϱ ∩ [t − t_Π, t] = ∅ then
5              Delete M@ϱ from W;
6          else
7              Replace M@ϱ by M@ϱ ∩ [t − t_Π, t] in W;
8  for each M@ϱ1, M@ϱ2 ∈ W with ϱ1 ⊊ ϱ2 do
9      Delete M@ϱ1 from W;
10 return (W, H);
```

deleting from $\mathcal{W}$ old information that falls outside $[t - t_\Pi, t]$ and by extending the history $\mathcal{H}$ with corresponding ground atoms (not mentioning intervals); intuitively, such atoms hold outside the sliding window, that is, somewhere within the range $(-\infty, t - t_\Pi)$. Then, in Lines 8 and 9, the algorithm eliminates redundancy by deleting from $\mathcal{W}$ all relational facts subsumed by others.

Example 27 *Consider Algorithm 8 running on an input consisting of the window $\mathcal{W}$ computed by* ApplyRules *in Theorem 26 and the time points $t = 101$ and $t_\Pi = 4$. In Lines 1–7 the algorithm forgets all facts that hold to the left of the sliding window (i.e., to the left of $t - t_\Pi = 97$). The fact $Signal(s_1)@96.3$ lies to the left of the sliding window (as $96.3 < 97$); hence, its ground atom $Signal(s_1)$ is added to $\mathcal{H}$ in Line 3 and $Signal(s_1)@96.3$ is removed from $\mathcal{W}$ in Line 5.*

In Lines 8 and 9, the facts $P(s_1)@[97, 100]$ and $P(s_1)@(100, 101]$ are deleted from $\mathcal{W}$, as they are both subsumed by $P(s_1)@[97, 101]$, which is also in $\mathcal{W}$.

The following theorems by Wałęga, Kaminski, and Cuenca Grau (2019) establish soundness and completeness of Algorithm 6.

Theorem 28 (Soundness) *Each fact streamed out by Algorithm 6 is in the answer to* (Π, Q) *over* $\mathcal{S}$ *relative to* $\mathbb{T}$.

Theorem 29 (Completeness) *Each fact in the answer to* (Π, Q) *over* $\mathcal{S}$ *relative to* $\mathbb{T}$ *is streamed out by Algorithm 6.*

5.4 Evaluation

We implemented Algorithm 6 on top of our MeTeoR system, which we have named MeTeoR-Str. To assess the performance of MeTeoR-Str, we conducted evaluations using the Temporal LUBM Benchmark and one of the tasks from the Hackathon Challenge at the 2021 Stream Reasoning Workshop[1].In particular, since we consider stream reasoning within the context of the DatalogMTL language, many open-source tools like StreamRule (Mileo, Abdelrahman, Policarpio, and Hauswirth, 2013) and LARS (Beck, Dao-Tran, and Eiter, 2018) are not directly applicable. MeTeoR was developed for general temporal reasoning in DatalogMTL and was not optimized for stream reasoning, so we use it as our baseline. This choice allows us to effectively demonstrate the superiority of our proposed optimized stream reasoning algorithm. In the future, it will be valuable to compare our algorithm with other stream reasoning tools, potentially by converting datasets and rules to be compatible with these existing tools.

Temporal LUBM Benchmark In particular, we noticed that several rules in the program Π of the Temporal LUBM Benchmark are not forward-propagating; hence, we chose two query predicates P_{nr} and P_r whose *P-relevant* (see the definition in Section 2.2.1) rules set with respect Π are named Π_{nr} and Π_r, respectively. In particular, Π_{nr} is non-recursive forwarding-propagating program

[1]https://streamreasoning.org/events/srw2021/

while Π_r is recursive forward-propagating programs. We generated timestamped datasets with time points ranging from 0 to 300, resulting in four temporal datasets: $\mathcal{D}_5$, $\mathcal{D}_{10}$, $\mathcal{D}_{15}$, and $\mathcal{D}_{20}$ consisting of 5, 10, 15, and 20 million facts, respectively. Notably, the four generated datasets have only punctual intervals. The distribution of facts in these datasets over various time points is depicted in Figure 5.1.

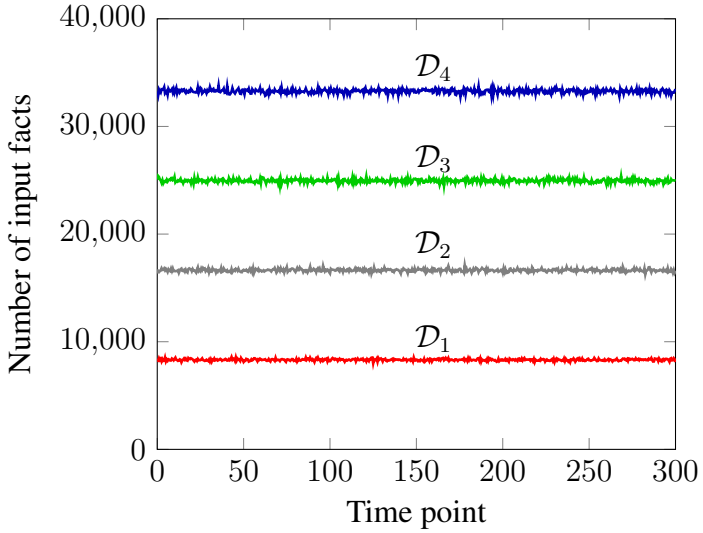


Figure 5.1: Temporal distribution of facts in input streams

For each dataset $\mathcal{D}_i$, with $i \in \{5, 10, 15, 20\}$, and each query (Π_j, P_j), with $j \in \{nr, r\}$, we computed the set of all query answers within the time interval $[0, 300]$. We did this in two different ways:

1. We passed program Π_j and dataset $\mathcal{D}_i$ as an input to MeTeoR and computed all facts $P_j@t$, with $t \in [0, 300]$, which are entailed by Π_j and D_i.

2. We ran MeTeoR-Str on Π_j and dataset $\mathcal{D}_i$ 'sliced' as a stream, in which facts were passed on as input in 0.5 time increments.

In both cases, we reported memory usage as the maximal number of temporal facts stored in memory at any point during the execution of the algorithm.

The results are summarised in Table 5.1, and clearly suggest that MeTeoR-Str uses significantly less memory than MeTeoR (approximately a 40-fold reduction in memory consumption in most cases). This is thanks to the sliding window mechanism, which allows Algorithm 6 to keep in memory only a limited number of facts by forgetting 'old' facts that are no longer relevant.

		$\mathcal{D}_5$	$\mathcal{D}_{10}$	$\mathcal{D}_{15}$	$\mathcal{D}_{20}$
Π_{nr}	MeTeoR	41612	83196	124717	165812
	MeTeoR-Str	**1006**	**2015**	**2841**	**3738**
Π_{r}	MeTeoR	92243	186087	279019	369664
	MeTeoR-Str	**2829**	**5699**	**8353**	**10945**

Table 5.1: Maximal number of facts stored by MeTeoR and MeTeoR-Str

The results from Table 5.1 also indicate that memory usage in Algorithm 6 increases linearly with the size of the input datasets $\mathcal{D}_5$–$\mathcal{D}_{20}$ passed on as streams. This is due to the fact that larger datasets were generated by increasing the density of the input streams, rather than by expanding the overall temporal interval over which facts are generated. Indeed, we have monitored memory usage of Algorithm 6 as the stream is processed and the window slides accordingly; the results for the non-recursive and the recursive benchmark queries are given in Figure 5.2 and Figure 5.3, respectively. The figures show that, initially, memory consumption increases rapidly as the first window is populated and materialised and no facts have been fogotten yet; however, as soon as the algorithm starts forgetting old facts, memory consumption stabilises and remains roughly constant from then onwards. This is a crucial property of stream reasoning algorithms, and suggests that the algorithm is able to process datasets of *any size* in a streaming fashion.

Hackathon Benchmark The Hackathon Challenge, organised at the Stream Reasoning Workshop 2021[2] (Schneider, Alvarez-Coello, Le-Tuan, Nguyen-Duc,

[2] https://streamreasoning.org/events/srw2021/

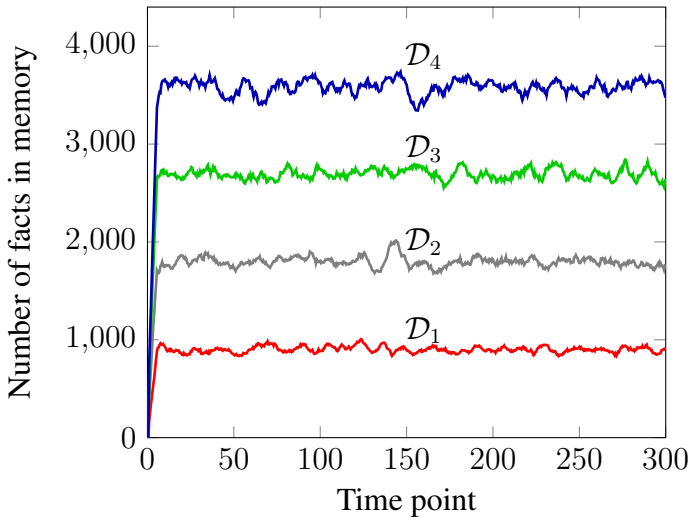


Figure 5.2: Memory usage of MeTeoR-Str running on the non-recursive query (Π_{nr}, P_{nr}) and different input streams

and Le-Phuoc, 2022) provides a stream generator together with several reasoning tasks. We considered the scenario where input streams contain data from Eclipse Simulation of Urban MObility (SUMO)[3] describing road vehicles in a traffic jam, and the task is to detect vehicles that make a short stop (less than 5 seconds).

We use input streams $\mathcal{S}_1$ and $\mathcal{S}_2$, where $\mathcal{S}_2$ contains a significantly larger number of facts than $\mathcal{S}_1$. Both streams provide new data every second indicating vehicles' positions, speeds, accelerations, etc. We used the initial fragments of these streams for the first 200 seconds; in this fragment $\mathcal{S}_1$ contains 23,828 facts, whereas $\mathcal{S}_2$ contains 80,124 facts. The distribution of facts in $\mathcal{S}_1$ and $\mathcal{S}_2$ over various time points is depicted in Figure 5.4.

To solve the task we have constructed a stream query $(\Pi, \mathit{ShortStop})$, with a DatalogMTL program Π consisting of the following rules, where $\mathit{Speed}_{=0}(x)$ and $\mathit{Speed}_{\neq 0}(x)$ mean that a vehicle x has a zero or non-zero speed, respectively,

[3] http://www.eclipse.org/sumo/

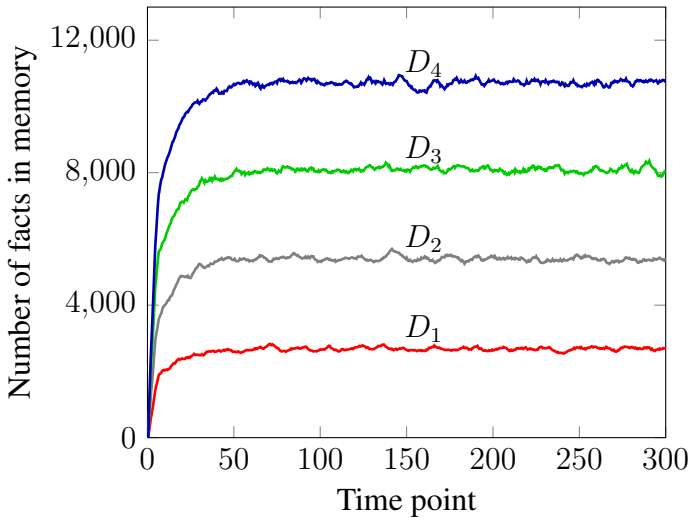


Figure 5.3: Memory usage of MeTeoR-Str running on the recursive query (Π_r, P_r) and different input streams

whereas $NotOnMap(x)$ indicates that a vehicle x is not on the map at the moment:

$$Speed_{\neq 0}(x) \leftarrow NotOnMap(x),$$

$$ShortStop(x) \leftarrow Speed_{\neq 0}(x) \wedge \boxminus_1 Speed_{=0}(x) \wedge \boxminus_2 Speed_{\neq 0}(x),$$

$$ShortStop(x) \leftarrow Speed_{\neq 0}(x) \wedge \boxminus_1 Speed_{=0}(x) \wedge \boxminus_2 Speed_{=0}(x) \wedge \boxminus_3 Speed_{\neq 0}(x),$$

$$ShortStop(x) \leftarrow Speed_{\neq 0}(x) \wedge \boxminus_1 Speed_{=0}(x) \wedge \boxminus_2 Speed_{=0}(x) \wedge \boxminus_3 Speed_{=0}(x) \wedge \boxminus_4 Speed_{\neq 0}(x),$$

$$ShortStop(x) \leftarrow Speed_{\neq 0}(x) \wedge \boxminus_1 Speed_{=0}(x) \wedge \boxminus_2 Speed_{=0}(x) \wedge \boxminus_3 Speed_{=0}(x) \wedge \boxminus_4 Speed_{=0}(x) \wedge \boxminus_5 Speed_{\neq 0}(x).$$

By the first rule, vehicles not present on a map are treated as having non-zero speed. The remaining four rules define the concept of a short stop; in particular, they state that a vehicle makes a short stop if it has zero speed in exactly one, two, three, or four consecutive time points of the input stream.

We report reasoning times and memory consumption of MeTeoR-Str on Figures

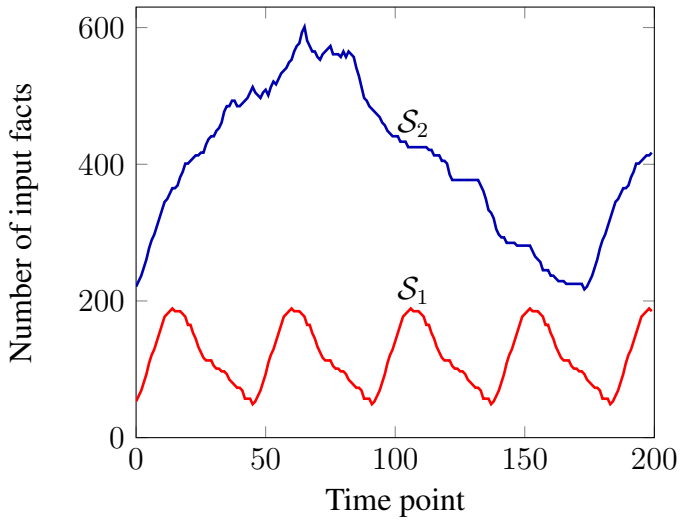


Figure 5.4: Temporal distribution of facts in input streams

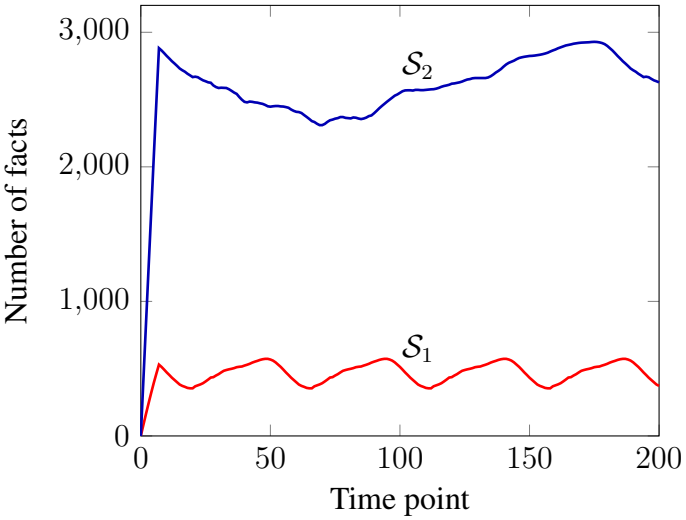


Figure 5.5: Memory usage of MeTeoR-Str running on the query $(\Pi, \mathit{ShortStop})$ and the input streams $\mathcal{S}_1$ and $\mathcal{S}_2$

5.5 and 5.6, respectively. Our results on Hackathon data are consistent with those on LUBM benchmark: memory consumption of MeTeoR-Str rapidly increases in the beginning until the first sliding window is populated with materialised facts, and then remains essentially stable with fluctuations depending on the number of facts arriving in the input streams (see Figure 5.5). These fluctuations are much

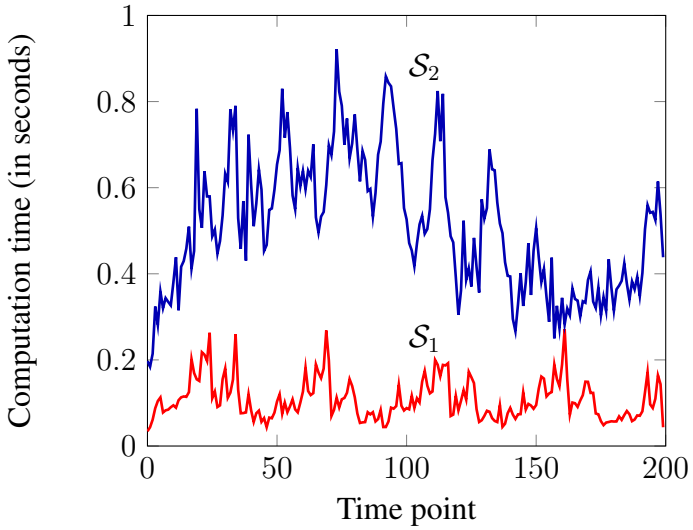


Figure 5.6: Time consumption of MeTeoR-Str running on the query $(\Pi, \mathit{ShortStop})$ and the input streams $\mathcal{S}_1$ and $\mathcal{S}_2$

more visible than in LUBM benchmark since the distribution of facts in the input streams is much less uniform for Hackathon benchmark (c.f. Figure 5.4). For the same reasons, we can also observe fluctuations in reasoning times (Figure 5.6). Note, however, that maximal reasoning time for each window remains below one second; since the input streams generate new facts each second, it means that MeTeoR-Str is able to perform reasoning in real time as data arrives.

6 | Extracting Temporal Rules Using Datalog Rule Learners

In the preceding chapters, our research has focused on practical reasoning algorithms in DatalogMTL, assuming the availability of a temporal dataset and relevant DatalogMTL rules. In this chapter, we shift our focus to scenarios in which the temporal data lacks these predefined temporal rules. Temporal rules provide deep insights into the dynamics of continuously evolving data, and can facilitate making well-informed predictions. However, obtaining high-quality temporal rules presents practical hurdles due to the scarcity of expert knowledge and the substantial human effort required. Our aim is to develop a method for automatically extracting temporal rules from the provided temporal data.

Unlike extracting static rules, the search space for temporal rules is significantly larger due to the inclusion of the time dimension, particularly when considering the whole rational timeline. Therefore, one big challenge is how to devise a pruning algorithm to efficiently reduce the search complicity in the time space. In this chapter, we present a framework for temporal rule learning from datasets, which capitalises on the availability of increasingly mature Datalog rule learners. Our approach is based on the idea of splitting a temporal dataset into windows, extracting static rules from each window with an off-the-shelf Datalog rule learner, and then combining the obtained static rules into temporal rules corresponding to the whole dataset. Temporal rules generated by our approach are expressed in DatalogMTL and are assigned time-sensitive confidence scores. We have implemented our approach in a system MTLearn compatible with any Datalog rule learner, as well as with a range of strategies for scoring the output temporal rules. Results on the task of temporal link prediction show that our proposed approach

is highly competitive, achieve performance comparable to that of state-of-the-art machine learning models for both the extrapolation and the interpolation settings, while at the same time providing interpretable results.

6.1 Temporal Link Prediction

We identify a *temporal knowledge graph* (tKG) with a temporal dataset $\mathcal{D}$ consisting of temporal facts of the form $R(c_1, c_2)@t$. In the context of link prediction, it is assumed that an incomplete tKG $\mathcal{D}$ is given and the aim is to predict which temporal facts hold in the (unknown) completion $\mathcal{D}^*$ of $\mathcal{D}$. We focus on answering prediction queries of the form $R(x, b)@t$ or $R(a, x)@t$ where x is a variable, whereas a, b and t are fixed constants and a time point, respectively. Hence, given $\mathcal{D}$ and a prediction query q, we focus on the task of finding variable assignments making the resulting temporal fact true in $\mathcal{D}^*$.

Let 0 and $t_{\max}$ be the least and largest time points mentioned in the input tKG $\mathcal{D}$; for simplicity of presentation we will assume that 0 is the minimal time point in all datasets. A time point t in the prediction query satisfies $t > t_{\max}$. link prediction tasks is to extend neural architectures and embedding techniques with a temporal dimension. Prominent models include RE-Net (Jin, Qu, Jin, and Ren, 2020), TTransE (Leblay and Chekol, 2018), TA-DisMult (Garcia-Duran, Dumančić, and Niepert, 2018), CyGNet (Zhu, Chen, Fan, Cheng, and Zhang, 2021), TIMEPLEX (Jain, Rathi, Chakrabarti, et al., 2020), and TComplEx (Lacroix, Obozinski, and Usunier, 2020).

6.2 Our Method

This section introduces our approach for extracting DatalogMTL rules by exploiting a readily available Datalog rule learning tool. At a high level, the execution of our algorithm on a temporal dataset $\mathcal{D}$ consists of the following steps.

1. Dataset $\mathcal{D}$ is transformed into a sequence of sets $\mathcal{F}_t$, where each such set consists of non-temporal facts obtained from facts in $\mathcal{D}$ which mention a given time point t.

2. Using a sliding window of size w (a configurable parameter), the algorithm processes w such datasets in ascending order of their time stamps. At each step, a Datalog rule learner is applied to extract a static Datalog program for the window, with each rule in the extracted programs accompanied by a confidence score.

3. The extracted Datalog programs are then converted into DatalogMTL programs. These temporal programs for each window are then combined into a single DatalogMTL program, wherein each rule is assigned a confidence score based on a predefined scoring strategy.

Assumptions Our approach requires that the input dataset $\mathcal{D}$ consists of punctual facts of the form $P(\mathbf{c})@t$, with t a natural number—a standard assumption in the literature on temporal KGs (Jin, Qu, Jin, and Ren, 2020; Zhu et al., 2021; Han, Chen, Ma, and Tresp, 2021; Liu, Ma, Hildebrandt, Joblin, and Tresp, 2022); this allows us to partition the facts in $\mathcal{D}$ according to their time stamps for further processing. In addition, we require that the least time point occurring in a dataset is 0. This is a technical assumption which allows us to initiate our sliding window approach from a convenient origin; clearly, this is without loss of generality, since

time points in the data can always be shifted by a constant value to ensure that the minimum time point in any fact is set to zero.

Our approach does not impose any additional limitations on Datalog rules, beyond those already dictated by the off-the-shelf Datalog rule learner employed as a black-box. However, we do impose restrictions on the use of metric temporal operators within the generated DatalogMTL rules. Specifically, the output rules are constrained to only include the operator $\boxminus$, and the use of this operator is restricted to rule bodies. The constraint primarily arises because static rule learners can only mine rules from static datasets. This necessitates transforming temporal facts into atemporal ones. Consequently, our approach is limited to punctual facts, as facts with a time range, such as $[1, 2]$, cannot be transformed into predicate subscripts. Additionally, our focus is on the forecasting task, which involves predicting future events based solely on past events. Therefore, we are not permitted to use future operators like $\boxplus$. Rules satisfying such requirements are *forward-propagating* (c.f. Section 2.2.3), in the sense that the derivation of a fact at time t can usually depend only on facts holding at time points previous to t in the timeline.

Rule Extraction Algorithm Next, we provide a detailed description of each of the steps mentioned at the start of this section. Specifically, we have formalised our rule extraction approach using Algorithm 9, and we will show its execution using the example in Figure 6.1.

The first step of our approach is implemented in Lines 1 and 2 of the algorithm. Here, we construct each set of facts $\mathcal{F}_t$ by converting every temporal fact $P(\mathbf{c})@t$ in the input dataset $\mathcal{D}$ into a non-temporal fact $P_t(\mathbf{c})$. To accomplish this, we introduce a new predicate P_t associated with P and t throughout the algorithm's execution. In the left-hand side of Figure 6.1, we can observe the conversion

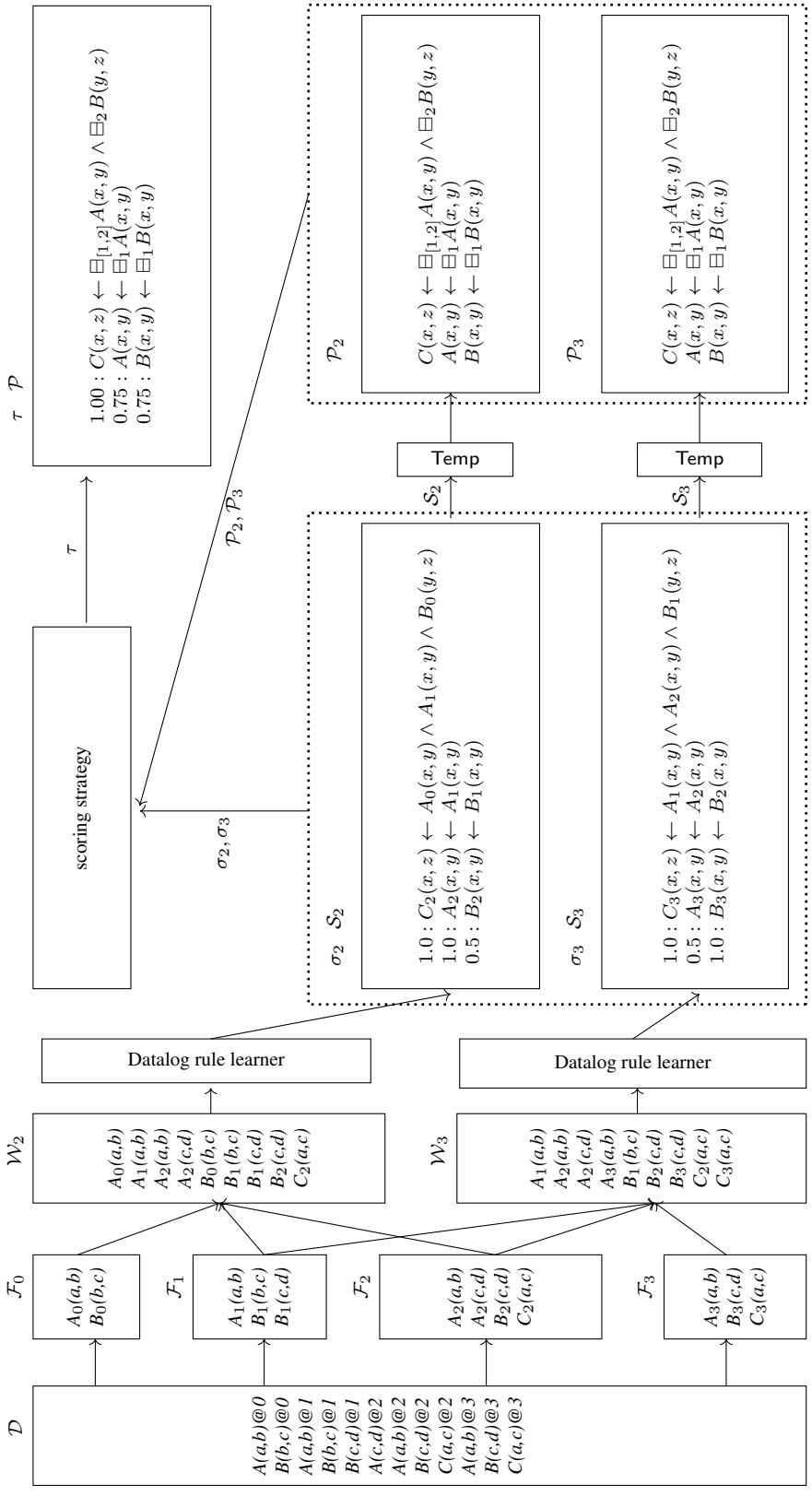


Figure 6.1: Execution of Algorithm 9 on the running example, with window size $w = 3$ and a scoring strategy 'average'; the input temporal dataset is depicted on the left-hand-side of the figure and the output set of DatalogMTL rules together with their confidence scores are in the top-right

process of the input dataset depicted in the leftmost block. It is transformed into sets of non-temporal facts $\mathcal{F}_0$, $\mathcal{F}_1$, $\mathcal{F}_2$, and $\mathcal{F}_3$ through the introduction of fresh predicates A_i, B_i, and C_i for each time point $0 \leq i \leq 3$ occurring in $\mathcal{D}$.

Algorithm 9: Temporal rule extraction

Input: A dataset $\mathcal{D}$ satisfying our assumptions
Parameters: A window size $w \in \mathbb{N}$, a Datalog rule learner L, and a scoring strategy
Output: A set of temporal rules and their scores

1 $t_{\max} \coloneqq$ the maximal time point in $\mathcal{D}$;
2 $\mathcal{F}_t \coloneqq \{(P_t(\mathbf{c}) \mid P(\mathbf{c})@t \in \mathcal{D}\}$, for each $0 \leq t \leq t_{\max}$;
3 **for** *each* $t \in \{w-1, \ldots, t_{\max}\}$ **do**
4 $\quad \mathcal{W}_t \coloneqq \mathcal{F}_{(t-w)+1} \cup \cdots \cup \mathcal{F}_t$;
5 $\quad (\mathcal{S}_t, \sigma_t) \coloneqq \mathsf{L}(\mathcal{W}_t)$;
6 $\quad \mathcal{P}_t \coloneqq \{\mathsf{Temp}(r) \mid r \in \mathcal{S}_t$ with head predicate associated to $t\}$;
7 $\mathcal{P} = \mathcal{P}_w \cup \cdots \cup \mathcal{P}_{t_{\max}}$;
8 Construct $\tau : \mathcal{P} \mapsto [0, 1]$ using the scoring strategy;
9 **return** $(\mathcal{P}, \tau)$;

The loop in Lines 3–6 considers, one by one, all *windows* $\mathcal{W}_t$ consisting of w consecutive datasets ending with $\mathcal{F}_t$, namely sequences $\mathcal{F}_{(t-w)+1}, \ldots, \mathcal{F}_t$ (Line 4). In Figure 6.1 we let $w = 3$, so the loop will consider two windows $\mathcal{W}_2$ and $\mathcal{W}_3$ corresponding to the time points within intervals $[0, 2]$ and $[1, 3]$ respectively.

For each such window $\mathcal{W}_t$, Algorithm 9 applies the off-the-shelf Datalog rule learner L to compute a set $\mathcal{S}_t$ of Datalog rules together with their scoring function σ_t (Line 5). Datalog rule learners typically generate a large number of rules, resulting in a potentially extensive set $\mathcal{S}_t$. However, in our specific case, we are solely interested in the subset of rules where the head predicate corresponds to the time point t. This subset of relevant rules is typically more manageable. In our example, the Datalog rule learner extracts sets $\mathcal{S}_2$ and $\mathcal{S}_3$ of relevant rules for windows $\mathcal{W}_2$ and $\mathcal{W}_3$, respectively, along with their associated confidence scores given by σ_2 and σ_3. Each of these relevant rules r is rewritten as a DatalogMTL

rule in Line 6 of the algorithm. The latter is achieved by function Temp.

Definition 30 Temp *is a function of rewriting a Datalog rule as a* DatalogMTL *rule. For each Datalog rule, it performs the following two steps:*

1. *Each maximal sequence of body atoms* $P_m(\mathbf{x}), P_{m+1}(\mathbf{x}), \ldots, P_n(\mathbf{x})$ *in* r *is replaced with* $P_{[m,n]}(\mathbf{x})$.
2. *The head atom* $P_t(\mathbf{x})$ *is replaced with* $P(\mathbf{x})$*, whereas each body atom* $P_{[m,n]}(\mathbf{x})$ *is replaced with the metric atom* $\boxminus_{[t-m,t-n]} P(\mathbf{x})$.

For example, a Datalog rule

$$C_3(x, y) \leftarrow A_1(x, y) \wedge A_2(x, y) \wedge B_1(y, z)$$

extracted for window $\mathcal{W}_3$ in our running example is first transformed into the auxiliary Datalog rule

$$C_3(x, y) \leftarrow A_{[1,2]}(x, y) \wedge B_{[1,1]}(y, z)$$

and subsequently rewritten as the final DatalogMTL rule

$$C(x, y) \leftarrow \boxminus_{[1,2]} A(x, y) \wedge \boxminus_{[2,2]} B(y, z).$$

Upon completion of the main loop, the algorithm gathers all the generated temporal rules into a DatalogMTL program $\mathcal{P}$, as indicated in Line 7. Following that, in Line 8, the algorithm computes a score ranging from 0 to 1 for each rule in $\mathcal{P}$ based on the provided scoring strategy. We explore several scoring strategies, which are further discussed in the subsequent subsection. The selection of a scoring function is treated as a parameter, and its significance in influencing the

algorithm's performance in temporal link prediction tasks will be demonstrated in the experiments section.

Scoring Strategies In this section, we describe the different strategies implemented in our approach for assigning scores to temporal rules.

The first type of strategy involves the computation of scores $\tau_t : \mathcal{P}_t \mapsto [0, 1]$ for each time point t individually; these are then combined to derive the overall scoring function τ. To be more specific, for each time point t and each rule $r \in \mathcal{P}_t$, we assign $\tau_t(r) = \sigma_t(r')$, where $r' \in \mathcal{S}_t$ represents the unique Datalog rule such that $\mathsf{Temp}(r') = r$ (uniqueness is guaranteed by the definition of Temp given in the previous subsection) and $\sigma_t(r')$ is the score assigned by the static Datalog rule learner to rule r' in window $\mathcal{W}_t$. The computation of $\tau(r)$ for a generated DatalogMTL rule r is based on the list Val_r containing the defined values among $\tau_w(r), \ldots, \tau_{t_{\max}}(r)$, where we note that for some $t \in \{w, \ldots, t_{\max}\}$ we may have $r \notin \mathcal{P}_t$ and hence $\tau_t(r)$ may be undefined. We propose four alternative ways of computing $\tau(r)$:

$$\begin{aligned}
\text{maximum:} \quad & \max(\mathit{Val}_r) \\
\text{average:} \quad & \operatorname{avg}(\mathit{Val}_r) \\
\text{weighted maximum:} \quad & \max(\mathit{Val}_r) \cdot \frac{|\mathit{Val}_r|}{t_{\max} - w + 1} \\
\text{weighted average:} \quad & \operatorname{avg}(\mathit{Val}_r) \cdot \frac{|\mathit{Val}_r|}{t_{\max} - w + 1}.
\end{aligned}$$

In our example from Figure 9, rule $A(x, y) \leftarrow \boxminus_1 A(x, y)$ is obtained in $\mathcal{P}_2$ and $\mathcal{P}_3$ with scores 1.0 and 0.5, respectively; the scoring function for $\mathcal{P} = \mathcal{P}_2 \cup \mathcal{P}_3$ computes their average, by assigning a score of 0.75 to the rule in $\mathcal{P}$.

Next, we define a scoring strategy which disregards the scores computed by the

off-the-shelf Datalog rule learner and relies instead on temporalised versions SC_T and HC_T of the standard confidence and head coverage metrics in the field of rule learning. In particular, we set $\tau(r)$ to

$$\beta \cdot SC_T(r) + (1 - \beta) \cdot HC_T(r), \tag{6.1}$$

for $\beta \in [0, 1]$ a parameter controlling the influence of SC_T and HC_T on the function τ. While confidence (SC) and head coverage (HC) metrics are widely used in the context of non-temporal Datalog rules (Meilicke, Chekol, Ruffinelli, and Stuckenschmidt, 2019; Galárraga, Teflioudi, Hose, and Suchanek, 2015), the definition of their temporal counterparts varies depending on the form of temporal rules, and different approaches adopt different definitions (Liu et al., 2022; Omran, Wang, and Wang, 2019). In the subsequent sections, we will present their definition in our specific context, maintaining a close analogy to the definitions of non-temporal SC and HC.

In the non-temporal case, for a Datalog rule r and a fixed dataset $\mathcal{D}$, SC and HC are usually given by the fractions

$$SC(r) = \frac{\text{supp}(r)}{\text{body-supp}(r)}, \qquad HC(r) = \frac{\text{supp}(r)}{\text{head-supp}(r)},$$

where the *rule support* $\text{supp}(r)$, *body support* $\text{body-supp}(r)$, and the *head support* $\text{head-supp}(r)$ are defined as follows, using the set G of all ground instances of r. The rule support, $\text{supp}(r)$, is the number of distinct ground head atoms H such that there exists in G a rule with head H and body B, both of which hold in $\mathcal{D}$. The definitions of $\text{body-supp}(r)$ and $\text{head-supp}(r)$ are analogous, but they

require only the body or only the head, respectively, to hold in $\mathcal{D}$. Formally:

$$\begin{aligned}\text{supp}(r) &= |\{H \mid (H \leftarrow B) \in G, \mathcal{D} \models H, \mathcal{D} \models B\}|,\\ \text{body-supp}(r) &= |\{H \mid (H \leftarrow B) \in G, \mathcal{D} \models B\}|,\\ \text{head-supp}(r) &= |\{H \mid (H \leftarrow B) \in G, \mathcal{D} \models H\}|.\end{aligned}$$

In the case of temporal rules, the notions of support should not only take into account the number of atoms, but also the time points in which they hold. These can be obtained, for example, by grounding the temporal variables occurring in a rule, as in TLogic (Liu et al., 2022), or alternatively by defining *dynamic* versions of SC and HC recursively via time, as in StreamLearner (Omran, Wang, and Wang, 2019). Our definition is close to the one used in TLogic, except that in DatalogMTL rules do not mention temporal variables, so they cannot be grounded per se. Instead, we define *temporalised variants* of SC and HC as

$$SC_T(r) = \frac{\text{supp}_T(r)}{\text{body-supp}_T(r)}, \qquad HC_T(r) = \frac{\text{supp}_T(r)}{\text{head-supp}_T(r)},$$

where the temporalised variants of supports count the number of facts $H@t$ and not only atoms H, namely:

$$\begin{aligned}\text{supp}_T(r) &= |\{H@t \mid (H \leftarrow B) \in G, \mathcal{D} \models H@t, \mathcal{D} \models B@t\}|,\\ \text{body-supp}_T(r) &= |\{H@t \mid (H \leftarrow B) \in G, \mathcal{D} \models B@t\}|,\\ \text{head-supp}_T(r) &= |\{H@t \mid (H \leftarrow B) \in G, \mathcal{D} \models H@t\}|,\end{aligned}$$

where t ranges over all points in the timeline.

It is worth observing that in our setting SC_T and HC_T can be equivalently defined using $r[\mathcal{D}]$ (the dataset obtained by applying r to $\mathcal{D}$) and the set $\mathcal{D}_r$ of all facts

$H@t$ in $\mathcal{D}$ such that H can be obtained by grounding the head of r:

$$SC_T(r) = \frac{|r[\mathcal{D}] \cap \mathcal{D}|}{|r[\mathcal{D}]|}, \qquad HC_T(r) = \frac{|r[\mathcal{D}] \cap \mathcal{D}|}{|\mathcal{D}_r|}.$$

Indeed, observe first that $|\mathcal{D}_r| = \text{head-supp}(r)$, by the definition. Next, note that all facts in $\mathcal{D}$ and $r[\mathcal{D}]$ are over time points (we assume that $\mathcal{D}$ has only such facts and rule r that can occur in our approach does not contain diamond operators). Thus $|r[\mathcal{D}]|$ is the number of facts which can be derived from $\mathcal{D}$ by an application of r, that is, by grounding r and checking if the obtained body holds in $\mathcal{D}$. Hence, $|r[\mathcal{D}]| = \text{body-supp}_T$. Finally, $|r[\mathcal{D}] \cap \mathcal{D}|$ is the number of facts in $r[\mathcal{D}]$ which additionally hold in $\mathcal{D}$, and so, $|r[\mathcal{D}] \cap \mathcal{D}| = \text{supp}_T(r)$.

Hence, SC_T and HC_T can be determined using any reasoner which can compute $r[\mathcal{D}]$; we obtain it using our developed MeTeoR system.

6.3 Evaluation

Next, we present the experimental setup to evaluate our implementation of Algorithm 9 in our system called MTLearn and discuss the results of our evaluation.

Tasks We have tested our approach on temporal link prediction, which is commonly used for evaluating temporal rule learners (Omran, Wang, and Wang, 2019; Liu et al., 2022; Xiong, Yang, Fekri, and Kerce, 2022). We conducted experiments across standard benchmarks (ICEWS14, ICEWS18, ICEWS0515) as well as our newly introduced synthetic benchmarks (tLUBM and iTEMP).

Benchmarks We used several standard benchmarks constructed from the Integrated Crisis Early Warning System[1] (ICEWS) dataset. These include CEWS14, ICEWS18, and ICEWS0515, which contain data about years 2014, 2018, and from 2005 to 2015, respectively. We use their splits into training, validation, and test sets as provided by Han, Chen, Ma, and Tresp (2021) and adopted also by Lin, Liu, Mao, Xu, and Cambria (2023). We also generated six synthetic benchmarks tLUBM and iTEMP_1–iTEMP_5, each consisting of a training and validation datasets, together with a set of gold-standard DatalogMTL rules. Testing data consists of all facts which can be derived by one round of application of the gold-standard rules to the union of the training and validation sets, and such that the time points in these facts are greater than time points in the training and validation sets (the latter condition is required by the extrapolation setting). We adopt the results of a single rule application based on the following consideration: during evaluation, we only consider the outcomes derived from a single rule application and use the corresponding rule score as the ranking criterion, similar to most static rule learners. Allowing multiple steps complicates score assignment, as a fact might be derived by multiple rules. We leave the exploration of multi-step derivations as future work. tLUBM is a temporalised version of LUBM (Guo, Pan, and Heflin, 2005), whereas iTEMP_1–iTEMP_5 are designed using the iTemporal[2] (Bellomarini, Nissl, and Sallinger, 2022) generator. Table 6.1 provides key statistics of these benchmarks, namely the number of entities, predicates, time points, as well as the number of temporal facts in the training, validation, and test sets, respectively. The table also provides the number of gold-standard temporal rules in the synthetic benchmarks.

[1] https://dataverse.harvard.edu/dataverse/icews
[2] https://github.com/kglab-tuwien/iTemporal.git

Table 6.1: Benchmarks' Statistics

	Entities	Predicates	Points	Train	Valid	Test	Rules
ICEWS14	7,128	230	365	63,685	13,823	13,222	–
ICEWS18	23,033	256	304	373,018	45,995	49,945	–
ICEWS0515	10,488	251	4017	322,958	69,224	69,147	–
tLUBM	29,265	28	145	389,498	50,523	61,026	46
$iTEMP_1$	1,000	10	100	40,000	5,000	5,000	10
$iTEMP_2$	1,000	14	100	40,000	5,000	5,000	12
$iTEMP_3$	1,000	18	100	40,000	5,000	5,000	14
$iTEMP_4$	1,000	20	100	40,000	5,000	5,000	16
$iTEMP_5$	1,000	22	100	40,000	5,000	5,000	20

Baseline models We used the following models as baselines. TTransE (Jiang, Liu, Ge, Sha, Chang, Li, and Sui, 2016) which is a temporal extension of the KG embedding model TransE (Bordes, Weston, Collobert, and Bengio, 2011), TA-DistMult (Garcia-Duran, Dumančić, and Niepert, 2018) which relies on recurrent neural networks to learn time-aware representations, TNTComplEx (Lacroix, Obozinski, and Usunier, 2020) which learns complex-valued temporal-aware embeddings, RE-NET (Jin, Qu, Jin, and Ren, 2020) which combines an RNN-based event encoder and a neighborhood aggregator to capture both time and KG structure, DE-SimplE (Goel, Kazemi, Brubaker, and Poupart, 2020) which proposes a diachronic entity embedding with a static segment and a time-varying segment, xERTE (Han, Chen, Ma, and Tresp, 2021) which is based on a subgraph extraction, TLogic (Liu et al., 2022) which extracts temporal rules using random walks, and TECHS (Lin et al., 2023) which exploits a graph convolutional network to embed topological structures and temporal dynamics. We could not obtain a usable version of StreamLearner.

Evaluation metrics We have adopted the standard protocol with temporal filtering (Han, Chen, Ma, and Tresp, 2021). Each dataset (tKG) is split into training,

validation, and testing sets according to the restrictions on time points imposed by the extrapolation and interpolation settings, where appropriate. We used the training and validation datasets to extract a DatalogMTL program Π and a function $\tau : \Pi \mapsto [0, 1]$ assigning scores to rules of Π.

To test performance of (Π, τ) on temporal link prediction we proceeded as follows. For each fact $R(a, b)@t$ in the test dataset, we construct a query $R(x, b)@t$, for which a is assumed to be the correct answer, and a query $R(a, x)@t$, for which b is the correct answer. Assume that the query is of the form $R(x, b)@t$ with the correct answer a. In the interpolation setting we let $\mathcal{D}$ be the union of the training and validation sets, whereas in the extrapolation setting $\mathcal{D}$ contains also all facts from the test set with time points smaller than t, which corresponds to the so-called *single-step* setting (Gastinger, Sztyler, Sharma, and Schuelke, 2022; Liu et al., 2022). Then, we compute the set of constants c such that $R(c, b)@t$ holds in the interpretation $T_\Pi(\mathcal{D})$ obtained by applying Π to $\mathcal{D}$. We let the score of each such answer c be the maximum amongst $\tau(r)$ for each $r \in \Pi$ deriving $R(c, b)@t$ from $\mathcal{D}$. Formally, the score of c is $\max_{\tau(r)}\{r \in \Pi \mid R(c, b)@t \in r[\mathcal{D}]\}$. We sort answers in descending order of scores (ties are broken by considering the scores of the next highest-score rules producing the answer, and if this does not differentiate the answers we use the alphabetic ordering). We perform time-aware filtering (Han, Chen, Ma, and Tresp, 2021; Sun, Zhong, Ma, Han, and He, 2021; Z. Li, Guan, Jin, Peng, Lyu, Zhu, Bai, Li, Guo, and Cheng, 2022; Liu et al., 2022), where we delete from the list of answers all constants $c \neq a$ such that $R(c, b)@t$ holds in the training, validation, or test dataset. The rank of an answer is given by its position on the list. The process is analogous for queries of the form $R(a, x)@t$.

We use mean reciprocal rank (MRR) and Hits@k, for $k \in \{1, 3, 10\}$, as standard metrics. MRR is defined as the mean of the reciprocals $\frac{1}{rank_i}$ over all queries q_i, where $rank_i$ is the rank of the correct answer in the list of answers to a query q_i

(if the correct answer is not on the list, we let $\frac{1}{rank_i} = 0$). Hits@k is the number of queries q_i, for which $rank_i \leq k$, divided by the number of all queries. Both metrics yield values within $[0, 1]$ with higher values indicating better performance. We report these values as percentages.

Some of our benchmarks are equipped with a set of gold-standard temporal rules, which are used for generating the validation and test sets. For such benchmarks we also introduce the *rule quality* (RQ) metric. Given a program Π extracted by MTLearn, a gold-standard program Π' from the benchmark, a union of training, validation, and test sets $\mathcal{D}$, and a test set $\mathcal{D}_T$, the value of RQ is the percentage of rules r in Π' such that $T_{\{r\}}(\mathcal{D}) \cap \mathcal{D}_T \subseteq T_{\Pi'}(\mathcal{D})$. Hence RQ indicates the percentage of rules in Π' which derive facts in $\mathcal{D}_T$ that are also derived by Π.

Main results The results for the ICEWS14, ICEWS18, and ICEWS0515 benchmarks are summarised in Table 6.2, whereas the results for the synthetic benchmarks tLUBM and iTEMP$_1$–iTEMP$_5$ are presented in Table 6.3. The results for baseline models on the benchmarks ICEWS14, ICEWS18, and ICEWS0515 are as reported by Lin et al. (2023), whereas the results on our synthetic benchmarks tLUBM and iTEMP$_1$–iTEMP$_5$ have been obtained by exploiting default hyperparameters and using the validation set to perform early stop of the training. Recall that MTLearn exploits Datalog rule learners. Such rule learners are often non-deterministic, for example due to reliance on random sampling, and so, distinct runs of MTLearn are also non-deterministic in the sense that they can lead to extraction of different temporal rules. Thus, for each experiment we performed five independent runs of MTLearn and reported mean results.

As shown in Table 6.2, TECHS, TLogic, and MTLearn obtained the highest scores on ICEWS14, ICEWS18, and ICEWS0515, with TECHS being usually slightly better than the other two models. It is worth to observe, however, that

Table 6.2: Results for the extrapolation setting with the highest scores written in bold and the second highest underlined; baseline results are provided by Lin et al. (Lin et al., 2023)

	ICEWS14				ICEWS18				ICEWS0515			
	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
TTransE	13.4	3.1	17.3	34.5	8.3	1.9	8.6	21.9	15.7	5.0	19.7	38.0
TA-DistMult	26.5	17.1	30.2	45.4	16.8	8.6	18.4	33.6	24.3	14.6	27.9	44.2
DE-SimplE	32.7	24.4	35.7	49.1	19.3	11.5	21.9	34.8	35.0	25.9	39.0	52.8
TNTComplEx	32.1	23.4	36.0	49.1	21.2	13.3	24.0	36.9	27.5	19.5	30.8	42.9
CyGNet	32.7	23.7	36.3	50.7	24.9	15.9	28.3	42.6	35.0	25.7	39.1	52.9
RE-Net	38.3	28.7	41.3	54.5	28.8	19.1	32.4	47.5	43.0	31.3	46.9	63.5
xERTE	40.8	32.7	45.7	57.3	29.3	21.0	33.5	46.5	46.6	37.8	52.3	63.9
TLogic	43.0	33.6	48.3	61.2	29.8	20.5	34.0	48.5	47.0	36.2	53.1	67.4
TECHS	**43.9**	**34.6**	**49.4**	**62.0**	**30.9**	21.8	**35.4**	**49.8**	**48.4**	**38.3**	**54.7**	**68.9**
MTLearn	42.8	33.9	48.3	60.4	28.8	**22.0**	34.8	46.7	47.5	35.6	53.4	67.1

Table 6.3: Results for the extrapolation setting on the synthetic benchmarks; the highest scores are in bold and the second highest are underlined

	tLUBM		$iTEMP_1$		$iTEMP_2$		$iTEMP_3$		$iTEMP_4$		$iTEMP_5$	
	MRR	H@10	MRR	H@10	MRR	H@10	MRR	H@3	MRR	H@10	MRR	H@10
TTransE	38.9	49.3	50.2	61.4	52.1	63.0	54.2	64.3	50.1	63.2	56.3	67.2
CyGNet	42.7	58.2	68.8	76.2	70.1	79.8	69.4	78.8	72.4	80.4	65.1	74.9
RE-Net	49.8	59.1	72.5	80.1	73.2	82.1	72.7	81.8	72.4	82.4	68.1	78.8
xERTE	59.4	70.1	78.3	84.2	89.2	93.4	73.4	83.2	76.4	84.8	84.3	90.8
TLogic	<u>64.1</u>	<u>72.4</u>	<u>85.3</u>	<u>91.1</u>	<u>90.2</u>	<u>94.1</u>	<u>85.2</u>	<u>92.9</u>	<u>79.2</u>	<u>88.1</u>	<u>90.1</u>	<u>94.9</u>
MTLearn	**65.2**	**75.1**	**87.3**	**94.2**	**93.4**	**96.7**	**89.4**	**95.2**	**81.4**	**92.8**	**92.3**	**96.2**

MTLearn and TLogic are able to also provide interpretable temporal rules—a key advantage—while providing results comparable to those obtained by TECHS. On ICEWS18 MTLearn was able to slightly outperform TECHS for H@1.

Test data in the synthetic benchmarks tLUBM and iTEMP$_1$–iTEMP$_5$ is generated by temporal rules, which may be a more appropriate setting for temporal rule learning models. As Table 6.3 shows, in this case MTLearn obtains very high scores and outperforms all the other approaches, with TLogic being the second-best model. Unfortunately, we could not gain access to TECHS's code and hence we could not test it on these benchmarks. All models typically obtained better results in synthetic benchmarks than on ICEWS-based benchmarks. One intuition is that ICEWS-based benchmarks do not include explicit temporal rules, and even in the best scenarios, ground-truth temporal rules for these benchmarks are not guaranteed to be learned. In this context, it is very challenging for rule-based approaches to learn sufficiently accurate rules for predictions. In contrast, neural models are assumed to learn from data and are better equipped to handle complex, high-dimensional data, capturing intricate patterns more effectively. However, for benchmarks with ground-truth temporal rules, the superior performance of rule-based methods, such as our approach and TLogic, can be attributed to the fact that these datasets were generated by applying temporal rules. This makes rule-based approaches more accurate for the temporal link forecasting in this specific setting.

In addition, Table 6.4 suggests that MTLearn was able to achieve very high RQ scores; interestingly, on iTEMP$_2$ we obtained a score of 100%. These results suggest a strong alignment between the benchmark gold-standard rules and the rules generated by our system.

Choice of hyperparameters Furthermore, we have analysed the impact of hyperparameters, namely the choice of Datalog rule learner, size of a window, and

Table 6.4: RQ results of MTLearn on six synthetic benchmarks.

Dataset	tLUBM	iTEMP_1	iTEMP_2	iTEMP_3	iTEMP_4	iTEMP_5
RQ	84.12	90.12	100.0	86.13	88.34	85.23

scoring strategy, on the performance of our approach.

- *Rule learners* We have tested MTLearn using AnyBURL, AMIE+, and Popper as Datalog rule learners. It turns out that using AnyBURL leads to the highest scores of MTLearn, which seems to be correlated with high performance of AnyBURL on Datalog rule learning task (Meilicke, Chekol, Ruffinelli, and Stuckenschmidt, 2019). Indeed, when using AnyBURL (with window size 5 and the scoring strategy from Equation (6.1) with $\beta = 0.5$), we observed the following improvements compared to using AMIE+: *(i)* for ICEWS18, an increase of 3.8% for MRR, of 4.3% for H@1, and of 3.1% for H@10; *(ii)* for tLUBM, an increase of 2.2% for MRR, of 3.3% for H@1, and of 2.6% for H@10. We obtained similar performance gaps when comparing MTLearn using AnyBURL and Popper: *(iii)* for ICEWS18 using AnyBURL leads to an increase of 5.3% for MRR, 4.6% for H@1, and 4.7% for H@10; *(iv)* for tLUBM using AnyBURL leads to an increase of 3.4% for MRR, of 3.5% for H@1, and of 3.7% for H@10.

- *Window size* We conducted experiments with window sizes 2–6, 8, and 10 using AnyBURL as a default rule learner and the scoring strategy from Equation (6.1) with $\beta = 0.5$. As shown in Figure 6.2, the performance of MTLearn improves overall as the window size increases; indeed, larger window sizes may allow for the discovery of rules capturing dependencies which take into account larger fragments of the timeline. For larger window sizes, however, performance levels off, indicating that further increasing the window size does not lead to significant gains. This observation aligns with

Table 6.5: Impact of scoring strategies on MTLearn; the highest scores are written in bold and the second highest are underlined

	ICEWS18		tLUBM	
	MRR	H@10	MRR	H@10
max	25.8	43.5	62.5	72.2
avg	26.4	44.4	62.1	71.4
w-max	27.7	45.2	63.1	73.1
w-avg	27.2	44.7	62.9	72.2
$\beta = 0.3$	<u>28.5</u>	<u>46.6</u>	<u>64.3</u>	**75.7**
$\beta = 0.5$	**28.8**	**46.7**	**65.1**	<u>75.1</u>
$\beta = 0.7$	27.9	46.1	63.9	74.6

the findings of StreamLearner (Omran, Wang, and Wang, 2019), which also relies on window size as a parameter for learning temporal rules.

- *Scoring strategy* We compared the impact using different scoring strategies: maximum (max), average (avg), weighted maximum (w-max), weighted average (w-avg), and the one from Equation (6.1) with varying values of β. For this experiment we used AnyBURL as a Datalog rule learner and we set the window size to 10 and 5 for ICEWS18 and tLUBM benchmarks, respectively. As shown in Table 6.5, w-max and w-avg scoring strategies outperform their non-weighted counterparts, but the best results were achieved using the strategy from Equation (6.1) with $\beta = 0.5$. It is worth observing, that although this strategy leads to the best performance, it requires most complex computations (it requires running a DatalogMTL reasoner). Therefore, whenever computational resources are limited, it maybe preferable to exploit other scoring strategies.

Conclusion and Future Work Our approach achieved comparable performance on temporal link prediction to state-of-the-art baselines, while at the same time providing interpretable rules in an expressive temporal logic programming lan-

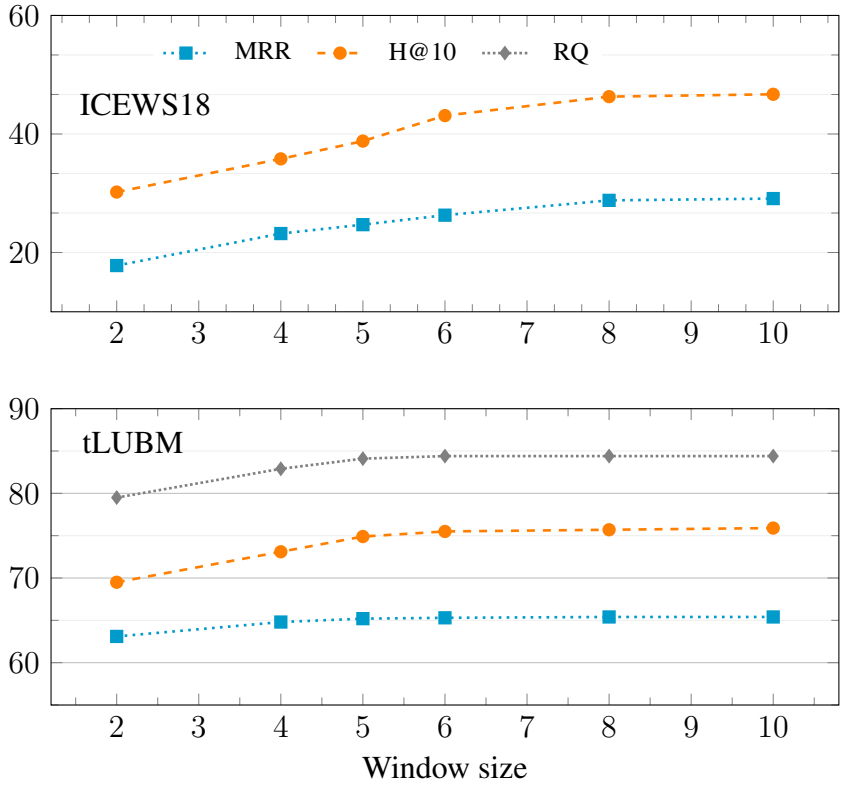


Figure 6.2: Impact of increasing window size on MTLearn

guage DatalogMTL. We expect the performance of MTLearn to improve as increasingly mature Datalog rule learners become available in the future. As future work, we aim to generate DatalogMTL rules using a wider range of operators, which can capture a richer class of temporal dependencies. Besides, another potential application of our method is that (unlike many other approaches) once trained on some benchmark, it could be also used for other benchmarks in an inductive setting where the rules are transferred to related datasets that share a common predicate vocabulary for straightforward application.

7 | Related Works

In this section, we review some prior research closely related to this thesis. In particular, we mainly discuss: temporal rule-based languages, existing approaches for temporal reasoning and temporal rule mining, respectively.

7.1 Temporal Rule-Based Languages

DatalogMTL was first introduced by Brandt et al. (2017) and it has found applications in areas such as temporal stream reasoning (Wałęga, Kaminski, and Cuenca Grau, 2019), temporal ontology-based data access (Brandt et al., 2017), specification and verification of banking agreements (Nissl and Sallinger, 2022), fact-checking economic claims (Mori, Papotti, Bellomarini, and Giudice, 2022), and the description of human movements (Raheb, Mailis, Ryzhikov, Papapetrou, and Ioannidis, 2017). The complexity of standard reasoning tasks in both the full language and its fragments has been investigated in depth (Brandt et al., 2018; Wałęga, Cuenca Grau, Kaminski, and Kostylev, 2019; Wałęga, Cuenca Grau, Kaminski, and Kostylev, 2021; Bellomarini, Nissl, and Sallinger, 2021). Alternative semantics to the standard continuous semantics over the rational timeline have also been considered; these include the semantics based on the integer timeline studied by Wałęga, Grau, Kaminski, and Kostylev (2020) and the event-based semantics by Ryzhikov, Wałęga, and Zakharyaschev (2019). DatalogMTL has also been extended with negation-as-failure under the stable model semantics, first for stratified programs (Tena Cucala, Wałęga, Cuenca Grau, and Kostylev, 2021) and subsequently for the general case (Wałęga, Tena Cucala, Kostylev, and Cuenca Grau, 2021).

There have been numerous alternative proposals for extending Datalog with tem-

poral constructs. For example, Datalog$_{1S}$ (Chomicki and Imieliński, 1988) is a prominent early extension, in which predicates are allowed to contain time terms of an additional temporal sort and a single successor function symbol over this sort. Specifically, a Datalog$_{1S}$ time term is the constant 0, a time variable, or the term $s(u)$ for u a time term. A time point t is then represented by t applications of the successor function to 0, i.e, $s(s(\dots s(0)\dots))$, and a time term $t + k$ is represented by k applications of the successor function to t, i.e, $s(s(\dots s(t)\dots))$. A stardard reasoning problem in Datalog$_{1S}$ is *fact entailment*, which is known to be PSpace-complete in data complexity. Ronca, Kaminski, Grau, and Horrocks (2022) investigated rule-based stream reasoning, focusing on a fragment of Datalog$_{1S}$ called Temporal Datalog. Temporal Datalog is a notational variant of Datalog$_{1S}$ with the additional temporal guardedness condition, namely rules do not mention explicit time points. The temporal guardedness ensures that an application of a rule to a stream is localised, in the sense that the time arguments of all facts in the stream matching the antecedent of the rule are close to each other and to the head that gets derived. As shown in Ronca, Kaminski, Grau, and Horrocks (2022), the temporal guardedness does not make standard reasoning easier, namely remaining PSpace-hard in data complexity.

More recently, temporal extensions of Datalog feature operators from LTL (Artale et al., 2015) as well as from the Halpern-Shoham logic of intervals (Kontchakov, Pandolfo, Pulina, Ryzhikov, and Zakharyaschev, 2016). Extending DatalogMTL with non-monotonic negation is closely related to temporal extensions of answer set programming (ASP) (Aguado, Cabalar, Diéguez, Pérez, Schaub, Schuhmann, and Vidal, 2021). ASP is a popular approach to solving knowledge-intense combinatorial search problems due to its performant solving engines and expressive modeling language. However, both are mainly geared towards static domains and lack native support for handling dynamic and time-dependent applications. This

shortcoming was addressed over the last decade by creating a temporal extension of ASP. The common use of ASP for temporal reasoning and the representation of action theories follows the methodology proposed by Gelfond and Lifschitz (1993). This methodology represents transition systems by adding to all time-dependent predicates an extra integer parameter: the time point T ranging over the finite interval $0, 1, \ldots, n-1$ where $n \in \mathbb{N}$ is some fixed length that can be iteratively increased. A major disadvantage of this representation is that it provides neither special language constructs nor specific inference methods for temporal reasoning, as available in LTL.

Further research (Cabalar, Kaminski, Morkisch, and Schaub, 2019; Beck, Dao-Tran, and Eiter, 2018; Cabalar, Diéguez, Schaub, and Schuhmann, 2020) have extended ASP with operators from LTL, resulting in a proper extension of the stable models semantics for the general case of temporal formulas in the syntax of LTL. For example, the solver *telingo* (Cabalar, Kaminski, Morkisch, and Schaub, 2019) is a temporal extension of the well-known ASP solver *clingo* (Gebser, Kaminski, Kaufmann, and Schaub, 2019) that uses its incremental solving capabilities. It deals with present-centered temporal programs that are expressible in the full (non-ground) input language of *clingo* extended with temporal operators from LTL. Moreover, *telingo* provides several syntactic extensions designed to enhance temporal modeling. Firstly, it enables the use of first and next operators in singular heads. Secondly, it allows the incorporation of arbitrary temporal formulas in integrity constraints. Additionally, *telingo* permits the utilization of nested temporal formulas within integrity constraints, as well as their negated forms as substitutes for temporal literals within rules. Particularly relevant is a recently introduced extension of ASP with MTL operators (Cabalar, Diéguez, Schaub, and Schuhmann, 2020; Cabalar, Diéguez, Schaub, and Schuhmann, 2022). Cabalar, Diéguez, Schaub, and Schuhmann (2020) treat time as a state counter by iden-

tifying the next time with the next state. Although this allows the restriction of temporal operators to subsequences of states, no fine-grained timing constraints are expressible. Cabalar, Diéguez, Schaub, and Schuhmann (2022) filled this gap in the context of temporal logic by associating each state with its time, as done in Metric Temporal Logic. This resulted in a non-monotonic metric temporal extension of ASP, which allows us to measure time differences between events. However, it is worth observing that all these temporal extensions of ASP are interpreted over the integer timeline.

Finally, operators from MTL have also been exploited in temporal extensions of description logics (DLs) (Gutiérrez-Basulto, Jung, and Ozaki, 2016; Baader et al., 2017; Artale and Franconi, 1998). DLs (Baader, Horrocks, and Sattler, 2004) are a well-known family of logic-based knowledge representation formalisms with a number of relevant applications. Important characteristics of DLs are high expressivity together with decidability, which guarantee that reasoning algorithms always terminate with the correct answers. However, DLs were initially developed with the aim of capturing static knowledge, so pure DLs are unable to capture the evolving behavior of dynamic domains. Schmiedel (1990) provided the first attempt to extend DLs with an interval based temporal logic introduced by Halpern and Shoham (1991). In the interval-based temporal description logics, temporal variables are constrained by means of temporal relationships based on Allen's interval algebra extended with metric constraints in order to deal with durations, absolute times and granularities of intervals. In this way, the logic is able to express abstract temporal patterns. In addition, there are other proposals that extend description logics with an explicit point-based notion of time. The resulting logics are the combination of a static DL with a tense logic (Gabbay and Guenthner, 2002). In general, whether in interval-based or point-based temporal description logics, the inclusion of temporal operators on the role side leads to undecidabil-

ity when considering the usual temporal structures (Woltert and Zakharyaschev, 1999). A promising framework is the combination of a description logic with a temporal concrete domain as pointed out by Haarslev, Lutz, and Möller (1998).

7.2 Approaches for Temporal Reasoning

Ontology-based data access (OBDA) (Calvanese, De Giacomo, Lembo, Lenzerini, and Rosati, 2007; Poggi, Lembo, Calvanese, De Giacomo, Lenzerini, and Rosati, 2008) is one of the most promising applications of Knowledge Representation in the Semantic Web domain. It is based on the idea of exposing a high-level conceptual layer in the form of an ontology on top of data sources. However, the classical OBDA frameworks and its associated ontology and query languages were primarily designed for static data and do not incorporate a temporal component to handle temporal information found in time-varying data. Consequently, numerous efforts have been made to develop temporal OBDA. The first temporal OBDA system with DatalogMTL support is Ontop-temporal (Kalaycı et al., 2018), an extension of the Ontop platform (Bagosi, Calvanese, Hardi, Komla-Ebri, Lanti, Rezk, Rodriguez-Muro, Slusnys, and Xiao, 2014) restricted to non-recursive programs and based on reasoning via rewriting into SQL. Very recently, the Vadalog reasoning system (Bellomarini, Sallinger, and Gottlob, 2018) has also been extended to support the DatalogMTL language (Bellomarini, Blasi, Nissl, and Sallinger, 2022); the algorithm implemented by this extension is, however, non-terminating in general. Fragments of DatalogMTL for which materialisation-based reasoning is guaranteed to terminate have been identified and studied by Wałęga, Zawidzki, and Cuenca Grau (2023); these fragments, however, impose restrictions which disallow programs expressing ‘recursion through time’.

Besides, there is a plethora of reasoners available for solving satisfiability check-

ing problems in LTL, with techniques involving reduction to model checking, tableau systems, and automata-based techniques. Prominent recent examples of highly-optimised LTL reasoners include nuXmv (Cavada, Cimatti, Dorigatti, Griggio, Mariotti, Micheli, Mover, Roveri, and Tonetta, 2014) and BLACK (Geatti, Gigante, and Montanari, 2019), where the latter was our LTL reasoner of choice in our experiments. BLACK was developed for satisfiability checking of Linear Temporal Logic (LTL) formulas, which supports formulas using both future and past operators, interpreted over both infinite and finite traces. At its core, BLACK uses an incremental SAT encoding of the one-pass tree-shaped tableau for LTL recently developed by Reynolds (2016), which guarantees completeness thanks to its particular pruning rule. As the underlying SAT solver, BLACK supports a variety of backends, such as MathSAT (Cimatti, Griggio, Schaafsma, and Sebastiani, 2013), Z3 (De Moura and Bjørner, 2008), MiniSAT (Eén and Sörensson, 2003), and CryptoMiniSAT (Soos, Nohl, and Castelluccia, 2009). nuXmv is the evolution of the NuSMV (Cimatti, Clarke, Giunchiglia, Giunchiglia, Pistore, Roveri, Sebastiani, and Tacchella, 2002), a widely used open source model checker, which integrates some of the most successful BDD and SAT based symbolic model checking algorithms. It extends NuSMV along two main directions: for finite-state systems it complements the basic verification techniques of NuSMV with state-of-the-art verification algorithms; for infinite-state systems, it extends the NuSMV language with new data types, namely Reals and unbounded Integers. Additionally, it provides advanced SMT-based model checking techniques while maintaining continuous optimization in terms of performance to be competitive with the state of the art. Regarding its practical significance, nuXmv has found applications in various industrial projects as a verification backend. It serves as the foundation for multiple extensions, addressing needs such as requirements analysis, contract-based design, model checking of hybrid systems, safety assess-

ment, and software model checking. In stark contrast with a plethora of available LTL reasoners that have been developed and successfully deployed in practice, the design and implementation of practical reasoning algorithms for the full DatalogMTL language, however, remains a largely unexplored area—something that has prevented its widespread adoption in applications. To the best of our knowledge, MeTeoR, which we have developed, is the first open-source reasoner for the full DatalogMTL language.

7.3 Approaches for Temporal Rule Mining

Association rules, typically represented in the form $X \leftarrow Y$, are integral in equipping decision-makers with insights that facilitate sound decision-making processes. These rules are instrumental in the generation of descriptive or predictive models from the amassed data, thereby not only offering a more nuanced explanation of the existing data but also enabling the prediction of future data trends (Han, Pei, and Tong, 2022). However, most of past studies on this topic have failed to emphasise the importance of temporal information in rule mining. Nonetheless, the temporal dimension is often a critical factor in real-world data applications, as data frequently exhibits time-based changes or is influenced by varying temporal conditions. Consequently, the mining of patterns or rules from temporal datasets has garnered substantial attention in the data mining community in recent years. This approach has been successfully applied to a wide variety of areas, such as biomedicine (Nguyen, Luo, Phung, and Venkatesh, 2018; Ageno, Català, and Pons, 2023), stock trading (Abe, Hirabayashi, Ohsaki, and Yamaguchi, 2007; Nair, Mohandas, Nayanar, Teja, Vigneshwari, and Teja, 2015; Al Galib, Alam, Hossain, and Rahman, 2012), intelligent transport systems (Wen, Zhang, Sun, Wang, and Xu, 2019; Lanka and Jena, 2014; Ganapathy and Parama-

sivam, 2019), and so on.

In contrast to non-temporal rules, which overlook temporal components and may consequently miss critical insights linked to specific time periods, temporal rules offer enhanced predictive and descriptive capabilities owing to their ability to consider time-related dynamics and variations (Ao, Luo, Wang, Zhuang, and He, 2017; Segura-Delgado, Gacto, Alcalá, and Alcalá-Fdez, 2020; Ageno, Català, and Pons, 2023; Nguyen, Luo, Phung, and Venkatesh, 2018; Liang, Xinming, Lin, and Wenliang, 2005; Nazerfard, Rashidi, and Cook, 2011). Traditional algorithms often treat the time component as a variable to represent temporal constraints, such as the temporal distance between events (Höppner, 2001; Wang and Zheng, 2020). Other methodologies integrate the time variable into the learning process to scrutinise the temporal phases in which specific rules are prevalent. For example, these methods can lead to the induction of rules that encapsulate phenomena recurring within user-specified time intervals (Y. Li, Ning, Wang, and Jajodia, 2003; Ramaswamy, Mahajan, and Silberschatz, 1998).

8 | Conclusions

8.1 Discussion of Results

In this thesis, we explore practical reasoning and rule mining algorithms in DatalogMTL, a powerful rule-based language for representing and reasoning about data and knowledge involving temporal information. We have designed and implemented three practical reasoning algorithms in DatalogMTL, which are specifically tailored to the full DatalogMTL programs, forward-propagating DatalogMTL programs, and interval-bounded DatalogMTL programs, respectively. Notably, we are the first one to present a practical reasoning algorithm for full DatalogMTL, which combines materialisation with an automata-based approach. To achieve favourable performance, our algorithm delegates most of the computations to the materialisation component, which we optimised using seminaïve reasoning strategies. In turn, the computationally-expensive automata-based approach is used only to guarantee termination and completeness of reasoning.

In light of the efficiency challenges associated with the automata-based approach, our investigation extends further into the formulation of DatalogMTL reasoning algorithms that rely solely on materialisation, dispense with automata construction altogether, and do not limit recursion. Although we focus on bounded DatalogMTL programs and datasets, where ∞ is not mentioned as an endpoint of any interval; this is a natural and expressive fragment, in which reasoning is as hard as in the unrestricted language. Furthermore, our study introduces a practical algorithm for stream reasoning, an interesting and valuable research direction in the domain of data processing and analytics. The current reasoning algorithm is limited to forward-propagating rules, which are syntactically restricted to preclude propagation of derived information towards past time points. Such rules are

motivated by the observation that received data in streaming applications typically influences only present and future events (e.g., a flag is raised only after a pattern in signal data is detected).

Finally, we introduce a framework for learning DatalogMTL rules that benefits from advancements on Datalog rule learning. Our proposed approach achieved state-of-the-art performance while at the same time providing interpretable rules.

8.2 Future Work

Our theoretical results and development of MeTeoR advances research on robust and efficient temporal reasoning engines, thereby enhancing their practical applicability. We see many exciting avenues for future research, as listed below.

Extending DatalogMTL with stratified negation Tena Cucala, Wałęga, Cuenca Grau, and Kostylev (2021) have extended DatalogMTL with stratified negation as failure—a very useful feature for applications—and provided an automata-based decision procedure. It would be interesting to extend MeTeoR to support stratified negation, which will require a non-trivial revision of our algorithm since materialisation within each separate stratum may be non-terminating.

Incremental materialisation-based reasoning in DatalogMTL Datalog systems typically materialise the entire set of outcomes derived from a Datalog program, enabling all subsequent queries to be directly evaluated against this materialisation. This method of reasoning is usually complemented with an incremental maintenance algorithm, which updates the materialisation in response to changes in the input facts. Although incremental materialisation-based reasoning has received extensive attention in the realm of Datalog (Motik, Nenov, Piro, and Horrocks, 2019b), extending these methodologies to the DatalogMTL framework

remains largely unexplored and presents an intriguing prospect.

Extension of pure materialisation-based reasoning algorithm to arbitrary intervals We have explored the design of reasoning algorithms for interval-bounded DatalogMTL, focusing exclusively on materialisation and entirely avoiding the construction of automata, while not restricting recursion. A promising direction for future research is to investigate whether the purely materialisation-based reasoning algorithm can be expanded to accommodate arbitrary intervals.

Learning DatalogMTL rules with more diverse metric temporal operators We present a framework for learning temporal rules written in DatalogMTL that benefits from the advancements on established Datalog rule learners. Our approach does not impose any additional limitations on Datalog rules, beyond those already dictated by the off-the-shelf Datalog rule learner employed as a black-box. However, we do impose restrictions on the use of metric temporal operators within the generated DatalogMTL rules. Specifically, the output rules are constrained to only include the operator $\boxminus$, and the use of this operator is additionally restricted to rule bodies. Rules satisfying such requirements are *forward-propagating*. As future work, it would be interesting and valuable to extend our methodology to learn DatalogMTL rules with more diverse metric temporal operators, thereby enriching the scope and utility of our approach.

Bibliography

Abadi, M., and Manna, Z. (1989). Temporal Logic Programming. *Journal of Symbolic Computation*, *8*(3), 277–295.

Abe, H., Hirabayashi, S., Ohsaki, M., and Yamaguchi, T. (2007). Evaluating a Trading Rule Mining Method Based on Temporal Pattern Extraction. In *The Third International Workshop on Mining Complex Data (MCD 2007) In Conjunction with ECML/PKDD* (pp. 49–58).

Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*.

Ageno, A., Català, N., and Pons, M. (2023). Acquisition of Temporal Patterns from Electronic Health Records: an Application to Multimorbid Patients. *BMC Medical Informatics and Decision Making*, *23*(1), 189.

Aguado, F., Cabalar, P., Diéguez, M., Pérez, G., Schaub, T., Schuhmann, A., and Vidal, C. (2021). Linear-Time Temporal Answer Set Programming. *Theory and Practice of Logic Programming*, *23*(1), 1–55.

Al Galib, A., Alam, M., Hossain, N., and Rahman, R. M. (2012). Study of Temporal Data Mining Technique on Dhaka Stock Exchange. *International Journal of Knowledge Engineering and Data Mining*, *2*(1), 1–13.

Alur, R., and La Torre, S. (2004). Deterministic Generators and Games for LTL Fragments. *ACM Transactions on Computational Logic*, *5*(1), 1–25.

Anderson, K. (2005). *Predicting the Weather: Victorians and the Science of Meteorology*.

Anicic, D., Fodor, P., Rudolph, S., and Stojanovic, N. (2011). EP-SPARQL: A Unified Language for Event Processing and Stream Reasoning. In *Proceedings of the 20th International Conference on World Wide Web* (pp. 635–644).

Ao, X., Luo, P., Wang, J., Zhuang, F., and He, Q. (2017). Mining Precise-

positioning Episode Rules from Event Sequences. *IEEE Transactions on Knowledge and Data Engineering*, *30*(3), 530–543.

Artale, A., and Franconi, E. (1998). A Temporal Description Logic for Reasoning about Actions and Plans. *Journal of Artificial Intelligence Research*, *9*, 463–506.

Artale, A., Kontchakov, R., Kovtunova, A., Ryzhikov, V., Wolter, F., and Zakharyaschev, M. (2015). First-order Rewritability of Temporal Ontology-mediated Queries. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence* (pp. 2706–2712).

Augusto, J. C. (2001). The Logical Approach to Temporal Reasoning. *Artificial Intelligence Review*, *16*, 301–333.

Baader, F., Borgwardt, S., Koopmann, P., Ozaki, A., and Thost, V. (2017). Metric Temporal Description Logics with Interval-Rigid Names. In *Frontiers of Combining Systems* (pp. 60–76).

Baader, F., Horrocks, I., and Sattler, U. (2004). *Description Logics*. Springer.

Bagosi, T., Calvanese, D., Hardi, J., Komla-Ebri, S., Lanti, D., Rezk, M., Rodriguez-Muro, M., Slusnys, M., and Xiao, G. (2014). The Ontop Framework for Ontology Based Data Access. In *The Semantic Web and Web Science* (pp. 67–77).

Baier, C., and Katoen, J.-P. (2008). *Principles of Model Checking*. MIT Press.

Baldor, K., and Niu, J. (2012). Monitoring Dense-time, Continuous-Semantics, Metric Temporal Logic. In *Runtime Verification* (pp. 245–259).

Barbieri, D. F., Braga, D., Ceri, S., Della Valle, E., and Grossniklaus, M. (2009). C-SPARQL: SPARQL for continuous querying. In *Proceedings of the 18th International Conference on World Wide Web* (pp. 1061–1062).

Basin, D., Klaedtke, F., and Zălinescu, E. (2018). Algorithms for Monitoring Real-time Properties. *Acta Informatica*, 309–338.

Baudinet, M., Chomicki, J., and Wolper, P. (1993). Temporal deductive databases. In *Temporal databases: Theory, design, and implementation* (pp. 294–320).

Beck, H., Dao-Tran, M., and Eiter, T. (2018). LARS: A Logic-based framework for Analytic Reasoning over Streams. *Artificial Intelligence*, *261*, 16–70.

Bellomarini, L., Blasi, L., Nissl, M., and Sallinger, E. (2022). The Temporal Vadalog System. In *International Joint Conference on Rules and Reasoning* (pp. 130–145).

Bellomarini, L., Nissl, M., and Sallinger, E. (2021). Query Evaluation in DatalogMTL–Taming Infinite Query Results. *arXiv preprint arXiv:2109.10691*.

Bellomarini, L., Nissl, M., and Sallinger, E. (2022). iTemporal: an Extensible Generator of Temporal Benchmarks. In *IEEE 38th International Conference on Data Engineering* (pp. 2021–2033).

Bellomarini, L., Sallinger, E., and Gottlob, G. (2018). The Vadalog System: Datalog-based Reasoning for Knowledge graphs. *Proceedings of the VLDB Endowment*, *11*, 975–987.

Bordes, A., Weston, J., Collobert, R., and Bengio, Y. (2011). Learning Structured Embeddings of Knowledge Bases. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 301–306).

Brandt, S., Kalaycı, E. G., Ryzhikov, V., Xiao, G., and Zakharyaschev, M. (2018). Querying Log Data with Metric Temporal Logic. *Journal of Artificial Intelligence Research*, *62*, 829–877.

Brandt, S., Kontchakov, R., Ryzhikov, V., Xiao, G., and Zakharyaschev, M. (2017). Ontology-based Data Access with a Horn Fragment of Metric Temporal Logic. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 1070–1076).

Bry, F., Eisinger, N., Eiter, T., Furche, T., Gottlob, G., Ley, C., Linse, B., Pichler,

R., and Wei, F. (2007). Foundations of Rule-based Query Answering. In *Reasoning Web* (pp. 1–153).

Cabalar, P., Diéguez, M., Schaub, T., and Schuhmann, A. (2020). Towards Metric Temporal Answer Set Programming. *Theory and Practice of Logic Programming*, *20*(5), 783–798.

Cabalar, P., Diéguez, M., Schaub, T., and Schuhmann, A. (2022). Metric Temporal Answer Set Programming over Timed Traces. In *International Conference on Logic Programming and Nonmonotonic Reasoning* (pp. 117–130).

Cabalar, P., Kaminski, R., Morkisch, P., and Schaub, T. (2019). telingo= ASP+ Time. In *International Conference on Logic Programming and Nonmonotonic Reasoning* (pp. 256–269).

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., and Rosati, R. (2007). Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *Journal of Automated Reasoning*, *39*, 385–429.

Carral, D., Dragoste, I., González, L., Jacobs, C. J. H., Krötzsch, M., and Urbani, J. (2019). VLog: A Rule Engine for Knowledge Graphs. In *International Semantic Web Conference* (pp. 19–35).

Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., and Tonetta, S. (2014). The NuXmv Symbolic Model Checker. In *Computer Aided Verification* (pp. 334–342).

Chomicki, J., and Imieliński, T. (1988). Temporal Deductive Databases and Infinite Objects. In *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (pp. 61–73).

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). Nusmv 2: An Opensource Tool for Symbolic Model Checking. In *Computer Aided Verification* (pp. 359–364).

Cimatti, A., Griggio, A., Schaafsma, B. J., and Sebastiani, R. (2013). The Math-

SAT5 SMT Solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (pp. 93–107).

Cropper, A., and Dumančić, S. (2022). Inductive Logic Programming at 30: A New Introduction. *Journal of Artificial Intelligence Research*, *74*, 765–850.

Cropper, A., and Morel, R. (2021). Learning Programs by Learning from Failures. *Machine Learning*, *110*, 801–856.

Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (2001). Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, *33*(3), 374–425.

Dell'Aglio, D., Valle, E. D., Calbimonte, J., and Corcho, Ó. (2014). RSP-QL Semantics: A Unifying Query Model to Explain Heterogeneity of RDF Stream Processing Systems. *International Journal on Semantic Web and Information Systems*, *10*(4), 17–44.

De Moura, L., and Bjørner, N. (2008). Z3: An Efficient SMT Solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (pp. 337–340).

Demri, S., Goranko, V., and Lange, M. (2016). *Temporal Logics in Computer Science: Finite-state Systems*.

Demri, S., and Schnoebelen, P. (2002). The Complexity of Propositional Linear Temporal Logics in Simple Cases. *Information and Computation*, *174*(1), 84–103.

Doherty, P., Kvarnström, J., and Heintz, F. (2009). A Temporal Logic-based Planning and Execution Monitoring Framework for Unmanned Aircraft Systems. *Autonomous Agents and Multi-Agent Systems*, *19*, 332–377.

Eén, N., and Sörensson, N. (2003). An Extensible SAT-solver. In *International Conference on Theory and Applications of Satisfiability Testing* (pp. 502–518).

Feikin, D. R., Higdon, M. M., Abu-Raddad, L. J., Andrews, N., Araos, R., Goldberg, Y., Groome, M. J., Huppert, A., O'Brien, K. L., Smith, P. G., Wilder-Smith, A., Zeger, S., Deloria Knoll, M., and Patel, M. K. (2022). Duration of Effectiveness of Vaccines Against SARS-CoV-2 Infection and COVID-19 Disease: Results of a Systematic Review and Meta-regression. *The Lancet*, 924–944.

Ferraz Costa, A., Yamaguchi, Y., Juci Machado Traina, A., Traina Jr, C., and Faloutsos, C. (2015). Rsc: Mining and Modeling Temporal Activity in Social Media. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 269–278).

Fürnkranz, J., Gamberger, D., and Lavrač, N. (2012). *Foundations of Rule Learning*. Springer Science & Business Media.

Fürnkranz, J., and Kliegr, T. (2015). A Brief Overview of Rule Learning. In *Rule Technologies: Foundations, Tools, and Applications: 9th International Symposium, RuleML* (pp. 54–69).

Gabbay, D. M., and Guenthner, F. (2002). *Handbook of philosophical logic*.

Galárraga, L., Teflioudi, C., Hose, K., and Suchanek, F. M. (2015). Fast Rule Mining in Ontological Knowledge Bases with AMIE+. *The VLDB Journal*, *24*(6), 707–730.

Galárraga, L. A., Teflioudi, C., Hose, K., and Suchanek, F. (2013). AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases. In *Proceedings of the International Conference on World Wide Web* (pp. 413–422).

Ganapathy, J., and Paramasivam, J. (2019). Prediction of Traffic Volume by Mining Traffic Sequences Using Travel Time Based PrefixSpan. *IET Intelligent Transport Systems*, *13*(7), 1199–1210.

Garcia-Molina, H., Ullman, J. D., and Widom, J. (2009). *Database Systems - The*

Complete book (2. ed.). Pearson Education.

Garcia-Duran, A., Dumančić, S., and Niepert, M. (2018). Learning sequence encoders for temporal knowledge graph completion. In *Proceedings of the conference on empirical methods in natural language processing* (pp. 4816–4821).

Gastinger, J., Sztyler, T., Sharma, L., and Schuelke, A. (2022). On the evaluation of methods for temporal knowledge graph forecasting. In *Neurips 2022 temporal graph learning workshop.*

Geatti, L., Gigante, N., and Montanari, A. (2019). A SAT-based Encoding of the One-Pass and Tree-Shaped Tableau System for LTL. In *Automated Reasoning with Analytic Tableaux and Related Methods* (pp. 3–20).

Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. (2019). Multi-shot ASP Solving with clingo. *Theory and Practice of Logic Programming*, *19*(1), 27–82.

Gelfond, M., and Lifschitz, V. (1993). Representing Action and Change by Logic Programs. *The Journal of Logic Programming*, *17*(2-4), 301–321.

Goel, R., Kazemi, S. M., Brubaker, M., and Poupart, P. (2020). Diachronic embedding for temporal knowledge graph completion. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 34, pp. 3988–3995).

Guo, Y., Pan, Z., and Heflin, J. (2005). LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics*, *3*(2-3), 158–182.

Gutiérrez-Basulto, V., Jung, J. C., and Ozaki, A. (2016). On Metric Temporal Description Logics. In *22nd European Conference on Artificial Intelligence* (pp. 837–845).

Haarslev, V., Lutz, C., and Möller, R. (1998). Foundation of Spatiotemporal Reasoning with Description Logics. In *International Conference on Principles of Knowledge Representation and Reasoning* (pp. 112–123).

Halpern, J. Y., and Shoham, Y. (1991). A Propositional Modal Logic of Time Intervals. *Journal of the ACM*, *38*(4), 935–962.

Han, J., Pei, J., and Tong, H. (2022). *Data Mining: Concepts and Techniques*. Morgan kaufmann.

Han, Z., Chen, P., Ma, Y., and Tresp, V. (2021). Explainable Subgraph Reasoning for Forecasting on Temporal Knowledge Graphs. In *International Conference on Learning Representations.*

Heintz, F., and Doherty, P. (2004). DyKnow: An Approach to Middleware for Knowledge Grocessing. *Journal of Intelligent & Fuzzy Systems*, *15*(1), 3–13.

Ho, H.-M., Ouaknine, J., and Worrell, J. (2014). Online Monitoring of Metric Temporal Logic. In *Runtime Verification* (pp. 178–192).

Höppner, F. (2001). Discovery of Temporal Patterns: Learning Rules about the Qualitative Behaviour of Time Series. In *European Conference on Principles of Data Mining and Knowledge Discovery* (pp. 192–203).

Hu, P., Motik, B., and Horrocks, I. (2022). Modular Materialisation of Datalog Programs. *Artificial Intelligence*, *308*, 103726.

Jain, P., Rathi, S., Chakrabarti, S., et al. (2020). Temporal knowledge base completion: New algorithms and evaluation protocols. In *Proceedings of the 2020 conference on empirical methods in natural language processing* (pp. 3733–3747).

Jiang, T., Liu, T., Ge, T., Sha, L., Chang, B., Li, S., and Sui, Z. (2016). Towards Time-aware Knowledge Graph Completion. In *International Conference on Computational Linguistics* (pp. 1715–1724).

Jin, W., Qu, M., Jin, X., and Ren, X. (2020). Recurrent Event Network: Autoregressive Structure Inferenceover Temporal Knowledge Graphs. In *Proceedings of the Conference on Empirical Methods in Natural Language Process-*

ing (pp. 6669–6683).

Kalaycı, E. G., Xiao, G., Ryzhikov, V., Kalayci, T. E., and Calvanese, D. (2018). Ontop-temporal: A Tool for Ontology-Based Query Answering over Temporal Data. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (pp. 1927–1930).

Kontchakov, R., Pandolfo, L., Pulina, L., Ryzhikov, V., and Zakharyaschev, M. (2016). Temporal and Spatial OBDA with Many-Dimensional Halpern-Shoham Logic. In *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 1160–1166).

Koymans, R. (1990). Specifying Real-time Properties with Metric Temporal Logic. *Real-time systems*, *2*(4), 255–299.

Lacroix, T., Obozinski, G., and Usunier, N. (2020). Tensor decompositions for temporal knowledge base completion. *arXiv preprint arXiv:2004.04926*.

Lanka, S., and Jena, S. (2014). A Study on Time Based Association Rule Mining on Spatial-temporal Data for Intelligent Transportation Applications. In *First International Conference on Networks & Soft Computing* (pp. 395–399).

Leblay, J., and Chekol, M. W. (2018). Deriving Validity Time in Knowledge Graph. In *Companion Proceedings of the Web Conference* (pp. 1771–1776).

Li, Y., Ning, P., Wang, X. S., and Jajodia, S. (2003). Discovering Calendar-based Temporal Association Rules. *Data & Knowledge Engineering*, *44*(2), 193–218.

Li, Z., Guan, S., Jin, X., Peng, W., Lyu, Y., Zhu, Y., Bai, L., Li, W., Guo, J., and Cheng, X. (2022). Complex evolutional pattern learning for temporal knowledge graph reasoning. In *Proceedings of the 60th annual meeting of the association for computational linguistics* (pp. 290–296).

Liang, Z., Xinming, T., Lin, L., and Wenliang, J. (2005). Temporal Association Rule Mining Based on T-Apriori Algorithm and Its Typical Application. In *Proceedings of International Symposium on Spatio-temporal Modeling, Spatial Reasoning, Analysis, Data Mining and Data Fusion.*

Lin, Q., Liu, J., Mao, R., Xu, F., and Cambria, E. (2023). TECHS: Temporal logical graph networks for explainable extrapolation reasoning. In *Proceedings of the 61st annual meeting of the association for computational linguistics* (pp. 1281–1293).

Liu, Y., Ma, Y., Hildebrandt, M., Joblin, M., and Tresp, V. (2022). Tlogic: Temporal Logical Rules for Explainable Link Forecasting on Temporal Knowledge Graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 4120–4127).

Margara, A., Urbani, J., van Harmelen, F., and Bal, H. E. (2014). Streaming the Web: Reasoning over Dynamic Data. *Journal of Web Semantics*, *25*, 24–44.

Maurer, E. P., Wood, A. W., Adam, J. C., Lettenmaier, D. P., and Nijssen, B. (2002). A Long-term Hydrologically Based Dataset of Land Surface Fluxes and States for the Conterminous United States. *Journal of Climate*, *15*(22), 3237–3251.

McDermott, D. (1982). A Temporal Logic for Reasoning About Processes and Plans. *Cognitive Science*, *6*(2), 101–155.

Meilicke, C., Chekol, M. W., Ruffinelli, D., and Stuckenschmidt, H. (2019). Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In *Proceedings of the International Joint Conferences on Artificial Intelligence* (pp. 3137–3143).

Mileo, A., Abdelrahman, A., Policarpio, S., and Hauswirth, M. (2013). Streamrule: a nonmonotonic stream reasoning system for the semantic web. In

Web reasoning and rule systems: 7th international conference, rr 2013, mannheim, germany, july 27-29, 2013. proceedings 7 (pp. 247–252).

Mori, M., Papotti, P., Bellomarini, L., and Giudice, O. (2022). Neural Machine Translation for Fact-checking Temporal Claims. In *Proceedings of the Fifth Fact Extraction and VERification Workshop* (pp. 78–82).

Motik, B., Nenov, Y., Piro, R., and Horrocks, I. (2019a). Maintenance of Datalog Materialisations Revisited. *Artificial Intelligence*, *269*, 76–136.

Motik, B., Nenov, Y., Piro, R., and Horrocks, I. (2019b). Maintenance of Datalog Materialisations Revisited. *Artificial Intelligence*, *269*, 76–136.

Motik, B., Nenov, Y., Piro, R., Horrocks, I., and Olteanu, D. (2014). Parallel Materialisation of Datalog Programs in Centralised, Main-memory RDF Systems. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 129–137).

Muggleton, S., and De Raedt, L. (1994). Inductive Logic Programming: Theory and Methods. *The Journal of Logic Programming*, *19*, 629–679.

Nair, B. B., Mohandas, V., Nayanar, N., Teja, E., Vigneshwari, S., and Teja, K. (2015). A Stock Trading Recommender System Based on Temporal Association Rule Mining. *SAGE Open*, *5*(2), 2158244015579941.

Nazerfard, E., Rashidi, P., and Cook, D. J. (2011). Using Association Rule Mining to Discover Temporal Relations of Daily Activities. In *International Conference on Smart Homes and Health Telematics* (pp. 49–56).

Nguyen, D., Luo, W., Phung, D., and Venkatesh, S. (2018). LTARM: A Novel Temporal Association Rule Mining Method to Understand Toxicities in a Routine Cancer Treatment. *Knowledge-Based Systems*, *161*, 313–328.

Ničković, D., and Piterman, N. (2010). From MTL to Deterministic Timed Automata. In *International Conference on Formal Modeling and Analysis of Timed Systems* (pp. 152–167).

Nissl, M., and Sallinger, E. (2022). Modelling Smart Contracts with DatalogMTL. In *Proceedings of the Workshops of the EDBT/ICDT*.

Nuti, G., Mirghaemi, M., Treleaven, P. C., and Yingsaeree, C. (2011). Algorithmic Trading. *IEEE Computer*, *44*(11), 61–69.

Omran, P. G., Wang, K., and Wang, Z. (2018). Scalable Rule Learning via Learning Representation. In *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 2149–2155).

Omran, P. G., Wang, K., and Wang, Z. (2019). Learning Temporal Rules from Knowledge Graph Streams. In *AAAI Spring Symposium: Combining Machine Learning with Knowledge Engineering*.

Özçep, Ö. L., Möller, R., and Neuenstadt, C. (2014). A Stream-Temporal Query Language for Ontology Based Data Access. In *Advances in Artificial Intelligence: 37th Annual German Conference on AI* (pp. 183–194).

Pirrò, G. (2020). Relatedness and Tbox-driven Rule Learning in Large Knowledge Bases. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 2975–2982).

Pnueli, A. (1977). The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science* (pp. 46–57).

Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., and Rosati, R. (2008). Linking Data to Ontologies. In *Journal on Data Semantics X* (pp. 133–173).

Raheb, K. E., Mailis, T., Ryzhikov, V., Papapetrou, N., and Ioannidis, Y. E. (2017). Balonse: Temporal aspects of dance movement and its ontological representation. In *European semantic web conference* (pp. 49–64).

Ramaswamy, S., Mahajan, S., and Silberschatz, A. (1998). On the Discovery of Interesting Patterns in Association Rules. In *VLDB* (Vol. 98, pp. 368–379).

Reynolds, M. (2016). A New Rule for LTL Tableaux. *arXiv preprint*

arXiv:1609.04102.

Ronca, A., Kaminski, M., Cuenca Grau, B., Motik, B., and Horrocks, I. (2018). Stream Reasoning in Temporal Datalog. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 1941–1948).

Ronca, A., Kaminski, M., Grau, B. C., and Horrocks, I. (2022). The Delay and Window Size Problems in Rule-based Stream Reasoning. *Artificial Intelligence*, *306*, 103668.

Ryzhikov, V., Wałęga, P. A., and Zakharyaschev, M. (2019). Data Complexity and Rewritability of Ontology-mediated Queries in Metric Temporal Logic Under the Event-based Semantics. In *International Joint Conferences on Artificial Intelligence* (p. 1851-1857).

Salman, A. G., Kanigoro, B., and Heryadi, Y. (2015). Weather Forecasting Using Deep Learning Techniques. In *International Conference on Advanced Computer Science and Information Systems* (pp. 281–285).

Schmiedel, A. (1990). *A Temporal Terminological Logic*. Techn. Univ., Projektgruppe KIT.

Schneider, P., Alvarez-Coello, D., Le-Tuan, A., Nguyen-Duc, M., and Le-Phuoc, D. (2022). Stream Reasoning Playground. In *European Semantic Web Conference* (pp. 406–424).

Segura-Delgado, A., Gacto, M. J., Alcalá, R., and Alcalá-Fdez, J. (2020). Temporal Association Rule Mining: An Overview Considering the Time Variable as an Integral or Implied Component. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *10*(4).

Soos, M., Nohl, K., and Castelluccia, C. (2009). Extending SAT Solvers to Cryptographic Problems. In *International Conference on Theory and Applications of Satisfiability Testing* (pp. 244–257).

Sun, H., Zhong, J., Ma, Y., Han, Z., and He, K. (2021). Timetraveler: Reinforce-

ment learning for temporal knowledge graph forecasting. In *Proceedings of the conference on empirical methods in natural language processing*.

Tena Cucala, D. J., Wałęga, P. A., Cuenca Grau, B., and Kostylev, E. V. (2021). Stratified Negation in Datalog with Metric Temporal Operators. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 6488–6495).

Thati, P., and Roşu, G. (2005). Monitoring Algorithms for Metric Temporal Logic Specifications. *Electronic Notes in Theoretical Computer Science*, *113*, 145–162.

Van Harmelen, F., Lifschitz, V., and Porter, B. (2008). *Handbook of knowledge representation*. Elsevier.

Vardi, M. Y. (2009). From Philosophical to Industrial Logics. In *Indian Conference on Logic and Its Applications* (pp. 89–115).

Wałęga, P. A., Cuenca Grau, B., Kaminski, M., and Kostylev, E. V. (2021). Tractable Fragments of Datalog with Metric Temporal Operators. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence* (pp. 1919–1925).

Wałęga, P. A., Grau, B. C., Kaminski, M., and Kostylev, E. V. (2020). DatalogMTL over the Integer Timeline. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning* (pp. 768–777).

Wałęga, P. A., Kaminski, M., and Cuenca Grau, B. (2019). Reasoning over Streaming Data in Metric Temporal Datalog. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 3092–3099).

Wałęga, P. A., Kaminski, M., Wang, D., and Grau, B. C. (2023). Stream reasoning with DatalogMTL. *Journal of Web Semantics*, *76*, 100776.

Wałęga, P. A., Tena Cucala, D. J., Kostylev, E. V., and Cuenca Grau, B. (2021). DatalogMTL with negation under stable models semantics. In *Proceedings*

of the international conference on principles of knowledge representation and reasoning (pp. 609–618).

Wałęga, P. A., Zawidzki, M., and Cuenca Grau, B. (2021). Finitely Materialisable Datalog Programs with Metric Temporal Operators. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning* (pp. 619–628).

Wałęga, P. A., Zawidzki, M., Wang, D., and Cuenca Grau, B. (2023). Materialisation-based Reasoning in DatalogMTL with Bounded Intervals. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 37, pp. 6566–6574).

Wałęga, P. A., Cuenca Grau, B., Kaminski, M., and Kostylev, E. V. (2019). DatalogMTL: Computational Complexity and Expressive Power. In *International Joint Conferences on Artificial Intelligence* (pp. 1886–1892).

Wang, C., and Zheng, X. (2020). Application of Improved Time Series Apriori Algorithm by Frequent Itemsets in Association Rule Data Mining Based on Temporal Constraint. *Evolutionary Intelligence*, *13*(1), 39–49.

Wang, D., Hu, P., Wałęga, P. A., and Cuenca Grau, B. (2022). MeTeoR: Practical Reasoning in Datalog with Metric Temporal Operators. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 5906–5913).

Wałęga, P. A., Zawidzki, M., and Cuenca Grau, B. (2023). Finite Materialisability of Datalog Programs with Metric Temporal Operators. *Journal of Artificial Intelligence Research*, *76*, 471–521.

Wei, Y. (2007). Stock price Prediction Based on Fuzzy Logic. In *International Conference on Machine Learning and Cybernetics* (pp. 1309–1314).

Wen, F., Zhang, G., Sun, L., Wang, X., and Xu, X. (2019). A Hybrid Temporal Association Rules Mining Method for Traffic Congestion Prediction. *Computers & Industrial Engineering*, *130*, 779–787.

Woltert, P., and Zakharyaschev, M. (1999). Modal Description Logics: Modalizing Roles. *Fundamenta Informaticae*, *39*(4), 411–438.

Xiong, S., Yang, Y., Fekri, F., and Kerce, J. C. (2022). TILP: Differentiable learning of temporal logical rules on knowledge graphs. In *The eleventh international conference on learning representations.*

Yang, F., Yang, Z., and Cohen, W. W. (2017). Differentiable Learning of Logical Rules for Knowledge Base Reasoning. *Advances in Neural Information Processing Systems*, 2319–2328.

Yang, J. (2022). *Translation of DatalogMTL into PLTL* (Unpublished master's thesis). University of Oxford, UK.

Zaniolo, C. (2012). Logical Foundations of Continuous Query Languages for Data Streams. In *Datalog 2.0* (pp. 177–189).

Zhou, X., Wang, F., and Zaniolo, C. (2006). Efficient Temporal Coalescing Query Support in Relational Database Systems. In *International Conference on Database and Expert Systems Applications* (pp. 676–686).

Zhu, C., Chen, M., Fan, C., Cheng, G., and Zhang, Y. (2021). Learning from History: Modeling Temporal Knowledge Graphs with Sequential Copy-generation Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 4732–4740).