Temporal Datalog with Existential Quantification

Matthias Lanzinger[1] , **Markus Nissl**[2] , **Emanuel Sallinger,**[1,2] and **Przemysław A. Wałęga**[1]
[1] Department of Computer Science, University of Oxford
[2] TU Wien
matthias.lanzinger@cs.ox.ac.uk, markus.nissl@tuwien.ac.at, sallinger@dbai.tuwien.ac.at,
przemyslaw.walega@cs.ox.ac.uk

Abstract

Existential rules, also known as tuple-generating dependencies (TGDs) or Datalog$^{\pm}$ rules, are heavily studied in the communities of Knowledge Representation and Reasoning, Semantic Web, and Databases, due to their rich modelling capabilities. In this paper we consider TGDs in the temporal setting, by introducing and studying DatalogMTL$^{\exists}$—an extension of metric temporal Datalog (DatalogMTL) obtained by allowing for existential rules in programs. We show that DatalogMTL$^{\exists}$ is undecidable even in the restricted cases of guarded and weakly-acyclic programs. To address this issue we introduce uniform semantics which, on the one hand, is well-suited for modelling temporal knowledge as it prevents from unintended value invention and, on the other hand, provides decidability of reasoning; in particular, it becomes 2-ExpSpace-complete for weakly-acyclic programs but remains undecidable for guarded programs. We provide an implementation for the decidable case and demonstrate its practical feasibility. Thus we obtain an expressive, yet decidable, rule-language and a system which is suitable for complex temporal reasoning with existential rules.

1 Introduction

DatalogMTL [Brandt *et al.*, 2018] is an extension of Datalog with operators from metric temporal logic MTL [Koymans, 1990], which are interpreted over the rational timeline. As a result, DatalogMTL provides an expressive rule-language which is well suited for temporal representation and reasoning, with applications in stream reasoning [Wałęga *et al.*, 2019; Wałęga *et al.*, 2023b] and temporal ontology-based query answering [Artale *et al.*, 2017; Kikot *et al.*, 2018], among others.

Reasoning in DatalogMTL is decidable; it is ExpSpace-complete for combined complexity [Brandt *et al.*, 2018] and PSpace-complete for data complexity [Wałęga *et al.*, 2019]. Furthermore, a number of syntactical fragments of DatalogMTL [Brandt *et al.*, 2018; Wałęga *et al.*, 2019; Wałęga *et al.*, 2020b] as well as modifications of its semantics [Wałęga *et al.*, 2020a; Ryzhikov *et al.*, 2019] have been established to obtain favourable computational properties. DatalogMTL has also been extended with stratified negation [Tena Cucala *et al.*, 2021] and with unrestricted negation under the stable model semantics [Wałęga *et al.*, 2021], in the spirit of the recent work on metric temporal answer set programming [Cabalar *et al.*, 2020].

Decidability and reasoning techniques devised for DatalogMTL heavily rely on the 'deterministic' character of its rules, namely rule heads are not allowed to mention existential quantification over the object domain, 'non-deterministic' temporal operators (e.g., $\diamondplus$ operator which stands for 'somewhere in the future'), or disjunctions [Brandt *et al.*, 2018; Wałęga *et al.*, 2023a; 2023c; Wang *et al.*, 2022]. On the other hand, it is well known that allowing for such constructs significantly increases modelling capabilities and application areas of logical languages.

In particular, logical languages with *existential rules*—allowing for existential quantification in heads—constitute a prominent research topic studied by the communities of Knowledge Representation and Reasoning (KRR) [Calì *et al.*, 2010; Leone *et al.*, 2012], Semantic Web [Arenas *et al.*, 2018; Calì *et al.*, 2012a], and Databases [Gottlob *et al.*, 2014; Fagin *et al.*, 2005]. Existential rules are used in KRR ontology languages to allow for *value invention*—reasoning about constants that do not explicitly occur in a problem specification—to enrich incomplete data with domain knowledge. Such rules play a crucial role in rule-based reasoning systems like Vadalog [Bellomarini *et al.*, 2018; Berger *et al.*, 2019], which are successfully applied in industry. The extension of Datalog with existential rules, Datalog$^{\exists}$ (also known as Datalog$^{\pm}$ rules or tuple-generating dependencies TGDs), is also widely studied in Semantic Web since it covers description logics from the DL-Lite [Calvanese *et al.*, 2007] and $\mathcal{EL}$ [Baader *et al.*, 2005] families, which underpin the standard OWL 2 profiles. Moreover, existential rules have been deeply studied in Databases in the context of constraint languages and their applications in data exchange as well as in data integration.

Existential rules are also of high importance for temporal reasoning; they have been studied in the context of temporal description logics [Artale and Franconi, 2005] and atemporal languages with linear-order operators, which can simulate some forms of temporal reasoning [Amarilli *et al.*, 2018]. In

		Full DatalogMTL$^\exists$	Guarded programs	Weakly acyclic programs
Natural semantics	OWA	undecidable		
	CWA			
Uniform semantics	OWA	undecidable		**2-ExpSpace-co.**
	CWA	**ExpSpace-co.**		

Table 1: Our results on the computational complexity of reasoning in DatalogMTL$^\exists$

the case of highly-expressive temporal extensions of Datalog, however, there is a lack of their extensive study. This is due to a bad computational behaviour of such languages; decidablity was obtained only in very restrictive cases, for example when the temporal domain is bounded [Urbani *et al.*, 2022].

In this paper, we address the above challenge. We introduce and study DatalogMTL$^\exists$, which is an extension of DatalogMTL with existential rules. Hence, DatalogMTL$^\exists$ can be seen as an extension of both DatalogMTL and Datalog$^\exists$. The combination of temporal and existential rules, as presented in the example below, yields a very expressive and natural language for modelling problems with a temporal dimensions and incomplete data.

Example 1. *Consider a system monitoring shipping services to detect vehicles operating in dangerous conditions, as described next. After a vehicle x departs with an order y, until the order arrives to the destination, there exists a driver z of this vehicle (1st rule). Time spent on driving a vehicle is considered as working time (2nd rule). A vehicle is operating in dangerous conditions if it is driven by a driver who has been working continuously for at least 8 hours (3rd rule). These rules are expressed by the following* DatalogMTL$^\exists$ *program* Π_{ex}*, where* $\mathcal{U}$ *is the 'until' operator, whereas* $\diamondminus_{[0,\infty)}$ *and* $\boxminus_{[0,8]}$ *stand for 'sometime in the past', and 'continuously in the last 8 hours', respectively:*

$$\exists z \mathsf{Drive}(z,x,y) \leftarrow (\diamondminus_{[0,\infty]}\mathsf{Depart}(x,y))\,\mathcal{U}_{[0,\infty)}\mathsf{Arrive}(x,y),$$
$$\mathsf{Working}(z) \leftarrow \mathsf{Drive}(z,x,y),$$
$$\mathsf{Dangerous}(x) \leftarrow \boxminus_{[0,8]}\mathsf{Working}(z) \wedge \mathsf{Drive}(z,x,y).$$

Our main contributions in this paper are as follows.

- We introduce DatalogMTL$^\exists$ and propose its two semantics: *natural*, where existential rules are evaluated individually at each time point, and *uniform*, where the evaluation is 'uniform' along the whole timeline.
- We give decidability and computational complexity results for the main reasoning tasks in full DatalogMTL$^\exists$ as well as in its *guarded* and *weakly-acyclic* fragments. In particular, we show that under the uniform semantics reasoning is decidable for weakly acyclic programs and for arbitrary DatalogMTL$^\exists$ programs if we consider the closed world assumption (CWA). In all other cases reasoning is undecidable, as depicted in Table 1.
- We implement and describe a reasoning system for DatalogMTL$^\exists$ programs under the uniform semantics. The system is obtained by combining Skolemisation with a (temporal) chase procedure.
- We evaluate our implementation using two benchmarks; the first one is based on a simulator of urban mobility and the second on the recently introduced iTemporal generator. Our results demonstrate that reasoning in DatalogMTL$^\exists$ can be feasible even for large instances.

We provide preliminary definitions in Section 2. Then, in Section 3, we introduce DatalogMTL$^\exists$ and show our main theoretical results together with proof sketches. In Section 4 we describe our prototypical implementation and its experimental evaluation. Finally, we conclude the paper in Section 5.

2 Preliminaries

In this section we show the standard definitions for DatalogMTL (without existential rules), interpreted over the rational timeline and under the continuous semantics [Brandt *et al.*, 2018; Wałęga *et al.*, 2019], as opposed to the alternative approaches with the integer timeline [Wałęga *et al.*, 2020a] or the pointwise semantics [Ryzhikov *et al.*, 2019].

Time and Intervals. The *(rational) timeline* is the set $\mathbb{Q}$ of rational numbers, called also *time points*. A time point is a fraction with an integer numerator and a positive integer denominator, both encoded in binary (a standard assumption in DatalogMTL). We consider intervals, ϱ, over $\mathbb{Q}$ with the standard notation (i.e., involving round and square brackets to denote if the interval is open or closed). An interval ϱ is *punctual* if it contains exactly one number and it is *positive* if it does not contain negative numbers. We will often abbreviate a punctual interval $[t,t]$ as t.

Syntax. Assume a function-free first-order signature with a *domain* Dom consisting of countably infinitely many constants. A *relational atom* is a first-order atom of the form $P(\mathbf{s})$, with P a predicate and $\mathbf{s}$ a tuple of terms (constants and variables) of the length matching the arity of P. A *metric atom* is an expression given by the following grammar, where $P(\mathbf{s})$ is a relational atom and ϱ is a positive interval:

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

A (non-existential) *rule* is an expression of the form

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

where each M_i is a metric atom, whereas M' is a metric atom not mentioning $\diamondminus$, $\diamondplus$, $\mathcal{S}$, and $\mathcal{U}$, and hence generated by the following grammar:

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

The conjunction $M_1 \wedge \cdots \wedge M_n$ in Expression (1) is the rule's *body*, each M_i is a *body atom*, and M' is the *head*. A rule is *safe* if all its variables occur in the body; a DatalogMTL *program* is a finite set of safe rules. An expression (metric atom, rule, program, etc.) is *ground* if it mentions no variables. The *grounding* $\mathsf{ground}(\Pi)$ of a program Π is the (usually infinite) set of all ground rules obtained by assigning constants from Dom to variables in Π. A *dataset* is a finite set of relational facts. The *grounding* $\mathsf{ground}(\Pi, \mathcal{D})$ of a program Π with respect to a dataset $\mathcal{D}$ is the (finite) set of all ground rules that can be obtained by assigning constants in Π or $\mathcal{D}$ to variables in Π. A metric/relational *fact* over an interval ϱ is $M@\varrho$, with M a ground metric/relational atom.

Semantics. An *interpretation* $\mathfrak{I}$ is a function assigning to each time point t a set of ground relational atoms; if $P(\mathbf{s})$ belongs to this set, we write $\mathfrak{I}, t \models P(\mathbf{s})$ and say that $P(\mathbf{s})$ is *satisfied* at t in $\mathfrak{I}$. This extends to complex metric atoms as given in Table 2. Interpretation $\mathfrak{I}$ satisfies a metric fact $M@\varrho$, written $\mathfrak{I} \models M@\varrho$, if $\mathfrak{I}, t \models M$ for all $t \in \varrho$. Interpretation $\mathfrak{I}$ satisfies a ground rule r whenever, if $\mathfrak{I}$ 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 rule r if it satisfies each rule in $\mathsf{ground}(\{r\})$. 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 all facts in $\mathcal{D}$. A dataset $\mathcal{D}$ *entails* a metric fact $M@\varrho$ if each model of $\mathcal{D}$ is also a model of $M@\varrho$. A program Π and a dataset $\mathcal{D}$ *entail* a metric fact $M@\varrho$, written $(\Pi, \mathcal{D}) \models M@\varrho$, if each model of Π and $\mathcal{D}$ satisfies $M@\varrho$.

$$
\begin{array}{lll}
\mathfrak{I}, t \models \top & & \text{for each } t \\
\mathfrak{I}, t \models \bot & & \text{for no } t \\
\mathfrak{I}, t \models \diamondminus_\varrho M & \text{iff} & \mathfrak{I}, t' \models M \text{ for some } t' \text{ with } t - t' \in \varrho \\
\mathfrak{I}, t \models \diamondplus_\varrho M & \text{iff} & \mathfrak{I}, t' \models M \text{ for some } t' \text{ with } t' - t \in \varrho \\
\mathfrak{I}, t \models \boxminus_\varrho M & \text{iff} & \mathfrak{I}, t' \models M \text{ for all } t' \text{ with } t - t' \in \varrho \\
\mathfrak{I}, t \models \boxplus_\varrho M & \text{iff} & \mathfrak{I}, t' \models M \text{ for all } t' \text{ with } t' - t \in \varrho \\
\mathfrak{I}, t \models M_1 \mathcal{S}_\varrho M_2 & \text{iff} & \mathfrak{I}, t' \models M_2 \text{ for some } t' \text{ with } t - t' \in \varrho \\
 & & \text{and } \mathfrak{I}, t'' \models M_1 \text{ for all } t'' \in (t', t) \\
\mathfrak{I}, t \models M_1 \mathcal{U}_\varrho M_2 & \text{iff} & \mathfrak{I}, t' \models M_2 \text{ for some } t' \text{ with } t' - t \in \varrho \\
 & & \text{and } \mathfrak{I}, t'' \models M_1 \text{ for all } t'' \in (t, t')
\end{array}
$$

Table 2: Semantics of ground metric atoms

Reasoning. The main reasoning tasks in DatalogMTL are consistency checking and fact entailment. The former is to determine if a given program and a dataset have a model and the latter is to check if a program and a dataset entail a given relational fact. These problems reduce to the complements of each other; both of them are PSpace-complete for data complexity, that is, when the size of a program is considered as fixed [Wałęga *et al.*, 2019], and ExpSpace-complete for combined complexity, when complexity is measured also with respect to the program [Brandt *et al.*, 2018].

3 DatalogMTL$^\exists$

In this section we introduce DatalogMTL$^\exists$. We provide it's syntax, two types of semantics, and results on decidability as well as computational complexity of reasoning.

3.1 Syntax

We obtain DatalogMTL$^\exists$ by extending DatalogMTL with (temporal) existential rules, analogously to the way Datalog$^\exists$ extends Datalog with existential rules [Calì *et al.*, 2009].

Formally, we let an *existential rule* be an expression of a similar form to a (non-existential) rule from Expression (1) except that now, in front of the head atom M', we allow for the existential quantifier $\exists$ with a tuple $\mathbf{x}$ of (*existential*) variables which are mentioned in M' but not in the rule body, so an existential rule is of the form

$$\exists \mathbf{x}\, M' \leftarrow M_1 \wedge \cdots \wedge M_n, \qquad \text{for } n \geq 1,$$

where M', $M_1, \ldots, M_n$ are metric atoms (i.e., they allow for nesting of temporal operators) generated by the same grammars as in Expression (1). Note that with such rules we can easily express multi-atom heads; for example $\exists x\, (Q(x, y) \wedge R(x, y)) \leftarrow P(y)$ can be written as three rules: $\exists x\, P'(x, y) \leftarrow P(y)$, $Q(x, y) \leftarrow P'(x, y)$, and $R(x, y) \leftarrow P(x, y)$.

An existential rule is *safe* if all its non-existential variables are mentioned in the body. A (DatalogMTL$^\exists$) program is a finite set of existential and non-existential safe rules. An existential rule r is *ground* if all the variables it mentions are existential and its *grounding*, $\mathsf{ground}(r)$, is the set of all ground rules obtained by assigning constants from Dom to non-existential variables in r.

As DatalogMTL$^\exists$ inherits from Datalog$^\exists$ undecidability of reasoning, we will study standard syntactical fragments that are known to be decidable for Datalog$^\exists$ [Fagin *et al.*, 2005; Calì *et al.*, 2013]; in particular, we will consider guarded and weakly acyclic programs.

Guarded Programs. A rule is *guarded* if one of its body atoms mentions all the variables occurring in the body of this rule. A program is guarded, if all its rules are so; for instance, Π_{ex} from Example 1 is guarded.

Weakly Acyclic Programs. We use a standard definition of a weakly acyclic program [Fagin *et al.*, 2005] involving a *dependency graph*. The dependency graph for a program Π has a vertex $v_{(P,i)}$ for each predicate P in Π and for each position $i \in \{1, \ldots, a\}$, where a is the arity of P. There is a normal edge from $v_{(P,i)}$ to $v_{(Q,j)}$ if there is a rule in Π with a body atom mentioning $P(\mathbf{s})$ and the head mentioning $Q(\mathbf{s}')$ such that the ith element of $\mathbf{s}$ is a variable, which is the same as the jth element of $\mathbf{s}'$. Moreover, there is a *special edge* from $v_{(P,i)}$ to $v_{(Q,j)}$ if there is an existential rule in Π with a body atom mentioning $P(\mathbf{s})$ and the head mentioning $Q(\mathbf{s}')$ such that the ith element of $\mathbf{s}$ is a variable which occurs also in $\mathbf{s}'$, and the jth element of $\mathbf{s}'$ is any existentially quantified variable in Π. A program is *weakly acyclic* if its dependency graph has no cycle containing a special edge. For instance, Π_{ex} from Example 1 is weakly acyclic, as its dependency graph is as in Figure 1, where normal edges are presented as solid arrows and special edges as dashed arrows.

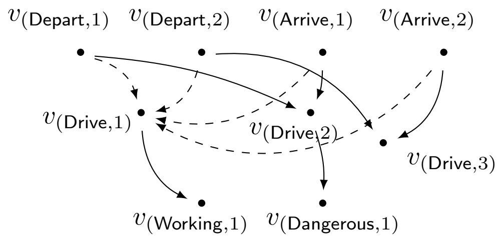

Figure 1: The dependency graph for Π_{ex} from Example 1

3.2 Semantics

First, we describe the 'natural' semantics (N) of DatalogMTL$^\exists$, which is based on the standard reading of existential quantification.

Definition 2. Under the *natural semantics*, an interpretation $\mathfrak{I}$ satisfies an existential rule r, written as $\mathfrak{I} \models_N r$, if for each ground rule $r' \in \mathsf{ground}(\{r\})$ of the form $\exists \mathbf{x}\, M' \leftarrow M_1 \wedge \cdots \wedge M_n$ and every time point t in which $\mathfrak{I}$ satisfies all body atoms $M_1, \ldots, M_n$, there exists an assignment ν of constants in Dom to variables of M' such that $\mathfrak{I}, t \models \nu(M')$. Interpretation $\mathfrak{I}$ satisfies a program Π under the natural semantics, written as $\mathfrak{I} \models_N \Pi$, if and only if $\mathfrak{I} \models_N r$, for each $r \in \Pi$.

We observe, however, that the natural semantics can introduce unintuitive behaviour in some settings. In particular, if the body of an existential rule holds in an interval ϱ, then the rule can lead to invention of distinct constants for each $t \in \varrho$, as illustrated in Example 3.

Example 3. *Consider Π_{ex} from Example 1 and a dataset $\mathcal{D}_{\mathsf{ex}}$ with the following facts about vehicles Veh_1, Veh_2 and orders Ord_1, Ord_2, Ord_3:*

$$\mathsf{Depart}(Veh_1, Ord_1)@0, \quad \mathsf{Arrive}(Veh_1, Ord_1)@12,$$
$$\mathsf{Depart}(Veh_2, Ord_2)@0, \quad \mathsf{Arrive}(Veh_2, Ord_2)@4,$$
$$\mathsf{Depart}(Veh_2, Ord_3)@4, \quad \mathsf{Arrive}(Veh_2, Ord_3)@10,$$
$$\mathsf{Drive}(Amy,\ Veh_2, Ord_2)@[0,4],$$
$$\mathsf{Drive}(Amy,\ Veh_2, Ord_3)@[4,10].$$

We have that $(\Pi_{\mathsf{ex}}, \mathcal{D}_{\mathsf{ex}}) \models_N \mathsf{Dangerous}(Veh_2)@[8,10]$. Intuitively, we would also want to deduce that Veh_1 is being operated under dangerous conditions, as its driver needs to work for 12 hours to deliver Ord_1. This entailment, however, does not hold, since the natural semantics allows for unintended models where Veh_1 has many drivers within the interval $[0,12]$, say a different driver in each of the densely distributed time points $t \in [0,12]$.

This example illustrates that while natural semantics is the straightforward way of extending DatalogMTL semantics, the resulting interactions with time can lead to unintended meaning of existential rules. Furthermore, as we will show in the next subsection, natural semantics lead to significant issues from a computational perspective.

To address these problems, we introduce 'uniform' semantics (U) which behaves better in both respects. Intuitively, it can be seen as replacing existential variables with Skolem terms, where the same Skolem terms are used no matter in which time point a rule is applied.

Definition 4. Under the *uniform semantics*, an interpretation $\mathfrak{I}$ satisfies an existential rule r, written as $\mathfrak{I} \models_U r$, if for every atom M' there exists an assignment $\nu_{r,M'}$ of constants to variables of M' such that for each ground rule $r' \in \mathsf{ground}(\{r\})$ of the form $\exists \mathbf{x}\, M' \leftarrow M_1 \wedge \cdots \wedge M_n$ and every time point t in which $\mathfrak{I}$ satisfies all body atoms $M_1, \ldots, M_n$, we have $\mathfrak{I}, t \models \nu_{r,M'}(M')$. Interpretation $\mathfrak{I}$ satisfies a program Π under the uniform semantics, written as $\mathfrak{I} \models_U \Pi$, if $\mathfrak{I} \models_U r$ for each $r \in \Pi$.

The uniform semantics provides an intuitive reading of our running example; we observe that now, the program Π_{ex} and the dataset $\mathcal{D}_{\mathsf{ex}}$ from Example 3 entail that Veh_1 operated under dangerous conditions, as intended. Indeed, we have $(\Pi_{\mathsf{ex}}, \mathcal{D}_{\mathsf{ex}}) \models_U \mathsf{Dangerous}(Veh_1)@[8,12]$ as the existential rule of Π_{ex} is interpreted via an 'uniform' assignment that invents *the same* constant representing a driver of Veh_1, within the whole interval $[0,12]$.

We will also differentiate between the *open world assumption* (OWA), where the assignments ν can use arbitrary constants from the domain Dom and the *closed world assumption* (CWA), where only constants from the active domain (i.e., occurring explicitly in a program or in a dataset) can be used. In particular, existential rules under CWA provide us with a compact and more human-friendly representation of long disjunctions in rule heads, which have gained attention for reasoning about 'complete data' or when privacy reasons impose restrictions on the form of allowed reasoning [Lutz *et al.*, 2019; Benedikt *et al.*, 2016; Wolter *et al.*, 2019].

Now, we can use our definitions to show that entailment in the uniform semantics generalises entailment in natural semantics, namely if a fact is entailed in natural semantics by some program and dataset, then it is also entailed in the uniform semantics. Furthermore, in the absence of temporal operators, DatalogMTL$^\exists$ under both semantics behaves exactly like Datalog$^\exists$, as we report formally next.

Proposition 5. *Let Π be a* DatalogMTL$^\exists$ *program, $\mathcal{D}$ a dataset, and $M@\varrho$ a relational fact. If $(\Pi, \mathcal{D}) \models_N M@\varrho$, then $(\Pi, \mathcal{D}) \models_U M@\varrho$, but the opposite implication does not hold. This property holds under both OWA and CWA.*

Proposition 6. *Let $M@0$ be a fact, Π a* DatalogMTL$^\exists$ *program without temporal operators, $\mathcal{D}$ a set of facts $M'@0$, and $\mathcal{D}' = \{M' \mid M'@0 \in \mathcal{D}\}$. The following are equivalent: (i) $(\Pi, \mathcal{D}) \models_N M@0$ under OWA, (ii) $(\Pi, \mathcal{D}) \models_U M@0$ under OWA, and (iii) Π and $\mathcal{D}'$ entail M in* Datalog$^\exists$.

3.3 Decidability and Computational Complexity

We recall that the central reasoning problems of DatalogMTL are consistency checking and fact entailment; we will study them in DatalogMTL$^\exists$ under both of the proposed semantics. We refer to the consistency checking problem under natural semantics and uniform semantics as N- and U-consistency, respectively. The same applies to N- and U-entailment. We start by observing that both reasoning problems are interreducible, as we can use a similar reduction as in the case of DatalogMTL [Brandt *et al.*, 2018]. Hence, in the further analysis of computational properties, we will focus on consistency checking only.

Proposition 7. *For both OWA and CWA and $\mathfrak{S} \in \{U, N\}$, checking $\mathfrak{S}$-consistency and fact $\mathfrak{S}$-entailment reduce in logarithmic space to the complement of each other.*

Undecidability of N- and U-consistency in full DatalogMTL$^{\exists}$ follows immediately from the well-known undecidability of Datalog$^{\exists}$. Thus, we will consider the guarded and weakly-acyclic fragments, since in the case of Datalog$^{\exists}$ both of them are decidable and proved to be useful in numerous applications [Fagin *et al.*, 2005; Calì *et al.*, 2009]. As the first main result, we observe that N-consistency is undecidable in each of these fragments. Surprisingly, this holds true even under CWA, when existentially quantified variables can be bound only to constants in the active domain.

Theorem 8. *Checking N-consistency for guarded as well as for weakly acyclic* DatalogMTL$^{\exists}$ *programs is undecidable under both OWA and CWA.*

Proof sketch. Under OWA we show undecidability by simulating a Turing machine computation. In the case of guarded programs we obtain it by constructing a program with a single existential rule $\exists z\, \mathsf{Next}(y, z) \leftarrow \mathsf{Next}(x, y)$, which introduces an infinite sequence of constants, that we use to represent tape cells. With the rule $\boxplus_1 \mathsf{Next}(x, y) \leftarrow \mathsf{Next}(x, y)$ we propagate facts about Next to all positive integer time points, which allows us to simulate configurations of the machine with facts holding in subsequent integer time points.

The existential rule we use in the reduction for the guarded programs makes the program not weakly acyclic. However, we can obtain a similar behaviour with a weakly acyclic program. To this end, we use an existential rule which introduces one constant per each negative integer time point. We propagate all these constants to the future which, again, gives us an access to infinitely many constants simulating tape cells and allows for simulating a Turing machine.

In the case of CWA we show undecidability for programs which are both guarded and weakly-acyclic. For this we use the fact that propositional DatalogMTL (with predicates of 0-arity) is undecidable if we extend it with rules mentioning $\diamondplus_{(0,1)}$ in heads [Brandt *et al.*, 2018, Theorem 10]. We show that $\diamondplus_{(0,1)} Q \leftarrow P$ can be simulated with constants true and false together with rules $\boxplus_{(0,1)} P' \leftarrow P$, $\exists x\, P''(x) \leftarrow P'$, $\bot \leftarrow P \wedge \boxplus_{(0,1)} P''(\mathsf{false})$, and $Q \leftarrow P''(\mathsf{true})$. ☐

In contrast, reasoning under the uniform semantics becomes decidable for weakly acyclic programs under OWA as well as for full DatalogMTL$^{\exists}$ programs under CWA. Note that the former result shows that reasoning with programs such as our Π_{ex} from Example 1 (whose intended meaning is provided by the uniform semantics) is decidable. Observe also that the result is not straightforward, as it requires providing a reasoning procedure which extends reasoning in DatalogMTL (for which, in general, the chase does not terminate) and, at the same time, handles invention of new values by existential rules. We provide tight complexity bounds for both reasoning tasks.

Theorem 9. *Checking U-consistency is 2-*ExpSpace*-complete for weakly acyclic* DatalogMTL$^{\exists}$ *programs under OWA and* ExpSpace*-complete for arbitrary* DatalogMTL$^{\exists}$ *programs under CWA.*

Proof sketch. ExpSpace-hardness is inherited from consistency checking in DatalogMTL [Brandt *et al.*, 2018]. For 2-ExpSpace-hardness we simulate computation of a Turing machine with doubly-exponentially many tape cells. We obtain it by combining the ideas from our undecidability proof for weakly acyclic programs in Theorem 8 with the ideas from the 2-ExpTime-hardness proof for weakly acyclic Datalog$^{\exists}$ [Calì *et al.*, 2012b].

For the upper bounds we reduce the problem to reasoning in DatalogMTL. In particular, we construct a set X of constants (with nulls) and non-deterministically guess assignments ν over X, interpreting existential rules. This allows us to transform the input DatalogMTL$^{\exists}$ program into a ground DatalogMTL program. Under CWA X is of linear size and under OWA with weakly acyclic programs, we can show that it suffices to consider doubly exponentially large X. The obtained ground DatalogMTL programs are exponentially and doubly exponentially large, respectively. Reasoning with such programs is performed by a translation to linear temporal logic [Brandt *et al.*, 2018], which yields the required bounds. The main technical challenge of the proof lies in the careful construction of the appropriate X's to avoid a blowup. ☐

On the other hand, we observe that reasoning for guarded programs remains undecidable even under the uniform semantics. This also implies undecidability of many other fragments that are decidable in Datalog$^{\exists}$, such as weakly-guarded or frontier-guarded programs, suggesting that the standard notions of guardedness are not applicable to the setting of temporal programs.

Theorem 10. *Checking U-consistency under OWA is undecidable for guarded* DatalogMTL$^{\exists}$ *programs.*

Proof sketch. The proof is obtained by modifying the reduction for guarded programs from Theorem 8. In a sense, the proof becomes even easier, as we do not need to propagate invented constants to the future/past time points; indeed, the uniform semantics guarantees that existential rules invent constants uniformly along the time line. ☐

We observe that all our undecidability proofs apply also to the case when the timeline consists of integer time points only. This, in turn, corresponds to extensions with existential rules of such formalisms as Temporal Datalog [Ronca *et al.*, 2018] or Datalog$_{1S}$ [Chomicki and Imieliński, 1988].

4 Experimental Evaluation

In this section, we report first experiments with our prototypical implementation of DatalogMTL$^{\exists}$ with uniform semantics and OWA, to demonstrate that our formalism has a potential for practical feasibility. We focus on the uniform semantics and OWA, as we consider it the choice of practical interest: it allows us to naturally express scenarios like the one from Example 1 and reasoning in this setting can be decidable.

4.1 Benchmarks and Execution Environment

As DatalogMTL$^\exists$ is a new formalism, there are no benchmarks or real-world instances to use in our experiments, as well as there are no other implementations for comparison. Thus, we exploit available resources to introduce two benchmarks with instances applicable to DatalogMTL$^\exists$ reasoning.

LARS$^+$. The most related language to DatalogMTL$^\exists$ seems to be LARS$^+$ which was recently introduced via extending the LARS framework with existential rules [Urbani *et al.*, 2022]. Rules of LARS$^+$ can be expressed in DatalogMTL$^\exists$, for example, the LARS$^+$ program used in experiments to reason about conveyor belts [Urbani *et al.*, 2022] can be written in our language in a straightforward manner as follows, where Bopr, Brkg, and IncId, stand for belt operator, broken gear, and incident Id, respectively

$$\begin{aligned}
\exists y\ \mathsf{Bopr}(x,y) &\leftarrow \mathsf{Belt}(x),\\
\exists z\ \mathsf{Brkg}(x,z) &\leftarrow \diamondminus_{[0,5]}\mathsf{Speed}(x,y) \land \mathsf{Slow}(y),\\
\exists z\ \mathsf{IncId}(z,x) &\leftarrow \boxminus_{[0,3]}\mathsf{Temp}(x,y) \land \mathsf{High}(y),\\
\mathsf{Assign}(y,z) &\leftarrow \mathsf{IncId}(y,x) \land \mathsf{Bopr}(x,z),\\
\mathsf{Block}(x) &\leftarrow \boxplus_{[0,3]}\mathsf{IncId}(z,x).
\end{aligned}$$

The structure of some programs in our benchmarks for DatalogMTL$^\exists$ are inspired by the above rules; however, the datasets used in LARS$^+$ experiments are not available [Urbani *et al.*, 2022]. It is also worth observing that although there are similarities in the syntax of LARS$^+$ and DatalogMTL$^\exists$, unlike in DatalogMTL$^\exists$, decidability of LARS$^+$ was shown only for cases when the temporal dimension can be finitely grounded (which is not the case already in DatalogMTL) and LARS$^+$ was interpreted over the integer time line (DatalogMTL$^\exists$ is interpreted over the rational timeline, which is increases the complexity of reasoning [Wałęga *et al.*, 2020a]).

SUMO. Our first benchmark is based on data from Eclipse Simulation of Urban MObility (SUMO)[1] describing road vehicles in a traffic jam, which was used during the Hackathon Challenge at the Stream Reasoning Workshop 2021 [Schneider *et al.*, 2022]. As in the Hackathon, we considered a simple map of roads and three dataset $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$ corresponding to *small*, *medium*, and *large* levels of traffic, respectively. The datasets have roughly 800, 5,000, and 9,000 temporal facts describing vehicles' position, speed, acceleration, and direction, among others. In the experiments we used the following DatalogMTL$^\exists$ program for detecting dangerous vehicles (where 1 unit on the timeline represents 1 second):

$$\begin{aligned}
&\exists d\,\mathsf{Drive}(d,v) \land\\
&\qquad \mathsf{SpeedExt}(v,s) \leftarrow \diamondminus_{[0,3)}\mathsf{Speed}(v,s),\\
&\qquad\quad \mathsf{Speeding}(v) \leftarrow \mathsf{SpeedExt}(v,s) \land \mathsf{HighSpeed}(s),\\
&\qquad\quad \exists z\ \mathsf{IncId}(z,v) \leftarrow \boxminus_{[0,5]}\mathsf{Speeding}(v),\\
&\mathsf{Inc}(z,d) \land \mathsf{Danger}(v) \leftarrow \mathsf{IncId}(z,v) \land \mathsf{Drive}(d,v),\\
&\qquad\qquad \mathsf{Danger}(v) \leftarrow \mathsf{Speeding}(v) \land \diamondminus_{[0,\infty)}\mathsf{Danger}(v).
\end{aligned}$$

As SUMO generates information about the traffic every 3 seconds, our first rule extends the information about speed to intervals of length 3 (using SpeedExt). The same rule also assigns a driver d to each vehicle v. The second rule marks with Speeding times when a vehicle exceeded allowed speed, whereas the third rule invents new ID numbers for incidents in which a vehicle was continuously Speeding for the last 5 seconds. The forth rule assigns incident IDs to involved drivers and marks corresponding vehicles as dangerous. The final rule recursively propagates via time the information about dangerous vehicles, whenever their Speeding is detected again. Note that the program is recursive and that it mentions heads with multi-atoms, which can be easily expressed in DatalogMTL$^\exists$ as we explained in Section 3.1.

iTemporal. Our second benchmark exploits *iTemporal*, which allows us to generate DatalogMTL programs and matching datasets of varying size [Bellomarini *et al.*, 2022b]. To obtain a benchmark for reasoning in DatalogMTL$^\exists$ we proceeded as follows. We started by generating with iTemporal DatalogMTL programs $\Pi_1, \dots, \Pi_4$ such that Π_1 (3 rules) is non-recursive and linear, Π_2 (14 rules) is also non-recursive but non-linear, whereas Π_3 (8 rules) and Π_4 (23 rules) are both recursive. Next, we extended each Π_i into 5 different DatalogMTL$^\exists$ programs in the following manner. First, we introduce existential rules by randomly selecting rule heads and extending their predicates with existentially quantified variables (this involves increasing the arity of predicates, which we perform consistently in the whole program). Second, we extend rules so that existential variables are 'propagated' to rule heads; namely for each rule whose body mentions a newly invented variable, we randomly decide if and where this variable will occur in the rule head. This process allows us to generate DatalogMTL$^\exists$ programs which, as we confirmed by our experiments, involve non-trivial propagation of existential variables. Moreover, for each Π_i, we invoke iTemporal to generate three matching datasets that contain approximately 5000, 50,000, and 500,000 temporal facts, respectively.

4.2 Implementation

We have provided a prototype implementation for weakly-acyclic DatalogMTL$^\exists$ programs under the uniform semantics and OWA. Our implementation exploits the well-known *Skolemisation* technique used for weakly-acyclic Datalog$^\exists$ programs [Marnette, 2009; Benedikt *et al.*, 2017], where existential variables z in rules are replaced by Skolem terms $f_z(\mathbf{x})$ that depend only on the frontier variables $\mathbf{x}$ (i.e., those variables that are shared between the head and the body of a rule) and where f is a unique function symbol per existential variable in a rule head. For example, the existential variable z in a rule r_1 of the form $\exists z\ B(x,z) \leftarrow \diamondminus_3 A(x,y)$ is replaced by $r_{1z}(x)$ resulting in $B(x, r_{1z}(x)) \leftarrow \diamondminus_3 A(x,y)$. Given an input dataset our implementation performs a chase (with a Skolemised program) which mimics consecutive applications of the immediate consequence operator in DatalogMTL [Brandt *et al.*, 2018; Wałęga *et al.*, 2019].

It is worth observing that our practical procedure is not guaranteed to terminate for arbitrary DatalogMTL$^\exists$ pro-

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

grams. However, it terminates in all instances of our benchmarks. Note also that after the chase terminates we obtain a model which allows us to determine all the facts that are entailed by the input program and the dataset, as well as to detect if the program and the dataset are consistent.We provided this implementation by extending the Vadalog system [Bellomarini *et al.*, 2018; 2022a], which allows for reasoning over Datalog$^{\pm}$, but not in DatalogMTL$^{\exists}$.

4.3 Experiments

All our experiments were run on an Intel Core i7-8700 CPU with 64GB memory. Since Vadalog is a commercial system we are not able to provide a full access to our implementation. Instead, we provide an online tool (https://kg.dbai.tuwien.ac.at/vadalog-scheduler), which allows to pass DatalogMTL$^{\exists}$ instances to our implementation, as well as to view the obtained outputs. This allows a user to pass programs from our benchmarks together with datasets (which can be freely modified by users) on the same hardware we used in the experiments reported in this paper.

The run times we obtain for the SUMO benchmark are 66 ms, 267 ms, and 397 ms, for increasing size datasets $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$, respectively. The results for the iTemporal benchmark with increasing size datasets (small, medium, and large, respectively) are presented in Table 3. The reported numbers are the runtimes of our implementation in ms until the chase terminates; in the case of iTemporal, those are the average times computed for all five DatalogMTL$^{\exists}$ programs obtained by extending a particular program Π_i. As we have already observed, this allows for performing fact entailment of arbitrary facts and checking consistency. It is worth to notice, however, that entailment of a fact can be verified as soon as the fact is derived, which may happen much quicker than the chase terminates. Thus, fact entailment may require in practice significantly less time. In Table 3 we have additionally reported in parentheses the standard deviations (also in ms), computed for each set of five programs generated from a particular Π_i.

Although the theoretical complexity of DatalogMTL$^{\exists}$ is high, we observed that both in the real-world inspired SUMO benchmark and in the artificially generated instances of various degree of complexity with iTemporal, the running times were usually around few seconds (17 seconds for the hardest instance). This supports usefulness of the language in practical scenarios. Regarding the SUMO benchmark, given the quite small number of facts in each of the three datasets, it is not surprising that the results show only slight increase in running time. Yet, it demonstrates that for a real-world scenario, reasoning can be performed fast enough (with the running time around 1 second), which is enough even for the stream reasoning setting, where reasoning needs to be performed in almost real time. Runtimes for the iTemporal benchmark are higher for the SUMO benchmark, as both programs and datasets in the iTemporal are significantly larger. Nevertheless, even for the hardest case, where the programs (obtained by extending Π_4) consist of 23 rules and the dataset consists of 500,000 temporal facts, the average runtime is less than 17 seconds. In the case of Π_4 we observe a large standard derivation compared to the other instances. A detailed investigation showed that one of the DatalogMTL$^{\exists}$ programs obtained from Π_4 has a runtime of around $2,800$ ms while the other four instances have run times over $10,000$ ms. Due to the random generation of existential rules, it may happen that some of the DatalogMTL$^{\exists}$ programs we generate do not allow for long derivations from a particular input dataset. It is hard to predict such a situation, and this is why for each Π_i we have generated several different DatalogMTL$^{\exists}$ programs.

	small		medium		large	
Π_1	88	(6)	738	(41)	7196	(225)
Π_2	464	(50)	455	(38)	4154	(463)
Π_3	161	(21)	1385	(151)	13701	(1019)
Π_4	273	(133)	2211	(1181)	16857	(8727)

Table 3: Results for increasing datasets in iTemporal; reported numbers are average times for five runs in ms whereas standard deviations are located in parentheses

5 Conclusions

We have introduced DatalogMTL$^{\exists}$, an extension of DatalogMTL that allows for existential rules. We have studied the computational complexity of the obtained language under the natural and uniform semantics. We showed that reasoning in DatalogMTL$^{\exists}$ under the natural semantics is undecidable even under severe restrictions. In contrast, reasoning becomes decidable under the uniform semantics in the case of CWA or weakly acyclic programs where we showed tight ExpSpace and 2-ExpSpace bounds. This provides us with a strict extension of both DatalogMTL and weakly acyclic Datalog$^{\exists}$, in which reasoning is decidable, even though it allows us to express complex temporal properties over a dense timeline and in the presence of existential rules. Furthermore, we provided a prototypical implementation and performed a series of experiments illustrating practical feasibility of the formalism. Our current implementation supports only the uniform semantics and OWA; in future we plan to consider natural semantics and CWA. This, however, requires fundamentally different approaches: natural semantics requires novel ideas for dealing with the possibility of infinitely many distinct assignments to existential variables in any interval, whereas CWA requires reasoning over large disjunctions in rule heads; both of them need techniques different from those used in our prototype implementation. In future we plan to introduce alternative restrictions to the language, and collaborate with our industrial partners on practical applications.

Acknowledgements

This work has been funded by the Vienna Science and Technology Fund (WWTF) [10.47379/VRG18013, 10.47379/NXT22018, 10.47379/ICT2201]; and the Christian Doppler Research Association (CDG) JRC LIVE. This work was supported by the SIRIUS Centre for Scalable Data Access (Research Council of Norway, project 237889), Samsung Research UK, and the EPSRC projects OASIS (EP/S032347/1), UK FIRES (EP/S019111/1) and ConCur (EP/V050869/1). Matthias Lanzinger acknowledges support by the Royal Society "RAISON DATA" project (Reference No. RP\R1\201074). For the purpose of Open Access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.

References

[Amarilli *et al.*, 2018] Antoine Amarilli, Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. Query answering with transitive and linear-ordered data. *J. Artif. Intell. Res.*, pages 191–264, 2018.

[Arenas *et al.*, 2018] Marcelo Arenas, Georg Gottlob, and Andreas Pieris. Expressive languages for querying the semantic web. *ACM Trans. Database Syst.*, 43(3):13:1–13:45, 2018.

[Artale and Franconi, 2005] Alessandro Artale and Enrico Franconi. Temporal description logics. In *Handbook of Temporal Reasoning in Artificial Intelligence*, pages 375–388, 2005.

[Artale *et al.*, 2017] Alessandro Artale, Roman Kontchakov, Alisa Kovtunova, Vladislav Ryzhikov, Frank Wolter, and Michael Zakharyaschev. Ontology-mediated query answering over temporal data: A survey. In *Proc. of TIME*, pages 1–37, 2017.

[Baader *et al.*, 2005] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope. In *Proc. of IJCAI*, pages 364–369, 2005.

[Bellomarini *et al.*, 2018] Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. The vadalog system: Datalog-based reasoning for knowledge graphs. *Proc. VLDB Endow.*, pages 975–987, 2018.

[Bellomarini *et al.*, 2022a] Luigi Bellomarini, Livia Blasi, Markus Nissl, and Emanuel Sallinger. The temporal vadalog system. In *Proc. of RuleML+RR*, pages 130–145, 2022.

[Bellomarini *et al.*, 2022b] Luigi Bellomarini, Markus Nissl, and Emanuel Sallinger. itemporal: An extensible generator of temporal benchmarks. In *Proc. of ICDE*, pages 2021–2033, 2022.

[Benedikt *et al.*, 2016] Michael Benedikt, Pierre Bourhis, Balder Ten Cate, and Gabriele Puppis. Querying visible and invisible information. In *Proc. LICS*, pages 297–306, 2016.

[Benedikt *et al.*, 2017] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. Benchmarking the chase. In *Proc. of PODS*, pages 37–52, 2017.

[Berger *et al.*, 2019] Gerald Berger, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. The space-efficient core of vadalog. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *In Proc. of PODS*, pages 270–284, 2019.

[Brandt *et al.*, 2018] Sebastian Brandt, Elem Güzel Kalaycı, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyaschev. Querying log data with metric temporal logic. *J. Artif. Intell. Res.*, pages 829–877, 2018.

[Cabalar *et al.*, 2020] Pedro Cabalar, Martin Dieguez, Torsten Schaub, and Anna Schuhmann. Towards metric temporal answer set programming. *Theory Pract. Log. Program.*, 20(5):783–798, 2020.

[Calì *et al.*, 2009] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *Proc. of ICDT*, pages 14–30, 2009.

[Calì *et al.*, 2010] Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *Proc. of LICS*, pages 228–242, 2010.

[Calì *et al.*, 2012a] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Semant.*, 14:57–83, 2012.

[Calì *et al.*, 2012b] Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012.

[Calì *et al.*, 2013] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.

[Calvanese *et al.*, 2007] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reason.*, 39(3):385–429, 2007.

[Chomicki and Imieliński, 1988] Jan Chomicki and Tomasz Imieliński. Temporal deductive databases and infinite objects. In *Proc. of PODS*, pages 61–73, 1988.

[Fagin *et al.*, 2005] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[Gottlob *et al.*, 2014] Georg Gottlob, Thomas Lukasiewicz, and Andreas Pieris. Datalog+/-: Questions and answers. In *Proc. of KR*, 2014.

[Kikot *et al.*, 2018] Stanislav Kikot, Vladislav Ryzhikov, Przemysław Andrzej Wałęga, and Michael Zakharyaschev. On the data complexity of ontology-mediated queries with MTL operators over timed words. In *Proc. of DL*, 2018.

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

[Leone *et al.*, 2012] Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. Efficiently computable Datalog$^{\exists}$ programs. In *Proc. of KR*, 2012.

[Lutz *et al.*, 2019] Carsten Lutz, Inanç Seylan, and Frank Wolter. The data complexity of ontology-mediated queries with closed predicates. *Log. Methods Comput. Sci.*, 15(3), 2019.

[Marnette, 2009] Bruno Marnette. Generalized schema-mappings: from termination to tractability. In *Proc. of PODS*, pages 13–22, 2009.

[Ronca *et al.*, 2018] Alessandro Ronca, Mark Kaminski, Bernardo Cuenca Grau, Boris Motik, and Ian Horrocks. Stream reasoning in temporal Datalog. In *Proc. of AAAI*, pages 1941–1948, 2018.

[Ryzhikov *et al.*, 2019] Vladislav Ryzhikov, Przemysław Andrzej Wałęga, and Michael Zakharyaschev. Data complexity and rewritability of ontology-mediated queries in metric temporal logic under the event-based semantics. In *Proc. of IJCAI*, pages 1851–1857, 2019.

[Schneider *et al.*, 2022] Patrik Schneider, Daniel Alvarez-Coello, Anh Le-Tuan, Manh Nguyen-Duc, and Danh Le-Phuoc. Stream reasoning playground. In *Proc. of ESWC*, pages 406–424, 2022.

[Tena Cucala *et al.*, 2021] David J. Tena Cucala, Przemysław Andrzej Wałęga, Bernardo Cuenca Grau, and Egor V. Kostylev. Stratified negation in Datalog with metric temporal operators. In *Proc. of AAAI*, pages 6488–6495, 2021.

[Urbani *et al.*, 2022] Jacopo Urbani, Markus Krötzsch, and Thomas Eiter. Chasing streams with existential rules. In *Proc. of KR*, 2022.

[Wałęga *et al.*, 2019] Przemysław Andrzej Wałęga, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. DatalogMTL: Computational complexity and expressive power. In *Proc. of IJCAI*, pages 1886–1892, 2019.

[Wałęga *et al.*, 2023a] Przemysław Wałęga, Michał Zawidzki, and Bernardo Cuenca Grau. Finite materialisability of Datalog programs with metric temporal operators. *J Artif Intell Res*, 76:471–521, 2023.

[Wałęga *et al.*, 2023b] Przemysław A Wałęga, Mark Kaminski, Dingmin Wang, and Bernardo Cuenca Grau. Stream reasoning with DatalogMTL. *Web Semant*, 76:100776, 2023.

[Wałęga *et al.*, 2023c] Przemysław Andrzej Wałęga, Michał Zawidzki, Dingmin Wang, and Bernardo Cuenca Grau. Materialisation-based reasoning in DatalogMTL with bounded intervals. In *Proc. of AAAI*, 2023.

[Wang *et al.*, 2022] Dingmin Wang, Pan Hu, Przemysław Andrzej Wałęga, and Bernardo Cuenca Grau. MeTeoR: Practical reasoning in Datalog with metric temporal operators. In *Proc. of AAAI*, pages 5906–5913, 2022.

[Wałęga *et al.*, 2019] Przemysław Andrzej Wałęga, Bernardo Cuenca Grau, and Mark Kaminski. Reasoning over streaming data in metric temporal Datalog. In *Proc. of AAAI*, pages 1941–1948, 2019.

[Wałęga *et al.*, 2020a] Przemysław Andrzej Wałęga, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. DatalogMTL over the integer timeline. In *Proc. of KR*, pages 768–777, 2020.

[Wałęga *et al.*, 2020b] Przemysław Andrzej Wałęga, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. Tractable fragments of Datalog with metric temporal operators. In *Proc. of IJCAI*, pages 1919–1925, 2020.

[Wałęga *et al.*, 2021] Przemysław Andrzej Wałęga, David J. Tena Cucala, Egor V. Kostylev, and Bernardo Cuenca Grau. DatalogMTL with negation under stable models semantics. In *Proc. of KR*, pages 609–618, 2021.

[Wolter *et al.*, 2019] Frank Wolter, Inanc Seylan, and Carsten Lutz. The data complexity of ontology-mediated queries with closed predicates. *Log. Methods Comput. Sci.*, 15, 2019.