Temporal Datalog with Existential Quantification

Matthias Lanzinger\textsuperscript{1}, Markus Nissl\textsuperscript{2}, Emanuel Sallinger,\textsuperscript{1,2} and Przemysław A. Wałęga\textsuperscript{1}
\textsuperscript{1} Department of Computer Science, University of Oxford
\textsuperscript{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\textsuperscript{+} 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\textsuperscript{3}—an extension of metric temporal Datalog (DatalogMTL) obtained by allowing for existential rules in programs. We show that DatalogMTL\textsuperscript{3} 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 [Kooi\-man, 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., \(\Diamond\) 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\textsuperscript{2} (also known as Datalog\textsuperscript{+} 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\textsuperscript{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].
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; decidability 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\(^3\), which is an extension of DatalogMTL with existential rules. Hence, DatalogMTL\(^3\) can be seen as an extension of both DatalogMTL and Datalog\(^9\). 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\(^3\) program \(\Pi_x\), where \(U\) is the ‘until’ operator, whereas \(\Diamond_{[0,\infty)}\) and \(\Box_{[0,8]}\) stand for ‘sometime in the past’, and ‘continuously in the last 8 hours’, respectively:

\[
\exists z \text{Drive}(z, x, y) \leftarrow (\Diamond_{[0,\infty)} \text{Depart}(x, y)) \cup_{[0,\infty)} \text{Arrive}(x, y),
\]

\[
\text{Working}(z) \leftarrow \text{Drive}(z, x, y),
\]

\[
\text{Dangerous}(x) \leftarrow \Box_{[0,8]} \text{Working}(z) \land \text{Drive}(z, x, y).
\]

Our main contributions in this paper are as follows.

− We introduce DatalogMTL\(^3\) 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\(^3\) 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\(^3\) 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\(^3\) 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 iTempo generator. Our results demonstrate that reasoning in DatalogMTL\(^3\) can be feasible even for large instances.

We provide preliminary definitions in Section 2. Then, in Section 3, we introduce DatalogMTL\(^3\) 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, \(\varnothing\), 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 \(\varnothing\) 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 \(\text{Dom}\) consisting of countably infinitely many constants. A relational atom is a first-order atom of the form \(P(s)\), with \(P\) a predicate and \(s\) 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 \(P(s)\) is a relational atom and \(\varnothing\) is a positive interval:

\[
M ::= \top \mid \bot \mid P(s) \mid \Phi\varnothing M \mid \Phi\varnothing M \mid \Box\varnothing M \mid MS\varnothing M \mid MU\varnothing M.
\]

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

\[
M' \leftarrow M_1 \land \cdots \land M_n, \quad \text{for } n \geq 1,
\]

where each \(M_i\) is a metric atom, whereas \(M'\) is a metric atom not mentioning \(\Diamond, \Box, S\), and \(U\), and hence generated by the following grammar:

\[
M' ::= \top \mid \bot \mid P(s) \mid \Box\varnothing M' \mid \Box\varnothing M'.
\]
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 
\textit{safe} if all its variables occur in the body; a DatalogMTL \textit{program} \ is a finite set of safe rules. An expression (metric atom, rule, program, etc.) is \textit{ground} if it mentions no variables. The \textit{grounding} ground($\Pi$) of a program $\Pi$ is the (usually infinite) set of all ground rules obtained by assigning constants from Dom to variables in $\Pi$. A \textit{dataset} is a finite set of relational facts. The grounding ground($\Pi$, $D$) of a program $\Pi$ with respect to a dataset $D$ \ is the (finite) set of all ground rules that can be obtained by assigning constants in $\Pi$ or $D$ to variables in $\Pi$. A metric/relation atom over an interval $\varrho$ is $M@\varrho$, with $M$ a metric/relation atom.

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

| $\mathcal{I},t \models \top$ | for each $t$ |
| $\mathcal{I},t \models \bot$ | for no $t$ |
| $\mathcal{I},t \models \Phi_\varrho M$ | iff $\mathcal{I},t' \models M$ \ for some $t'$ with $t-t' \in \varrho$ |
| $\mathcal{I},t \models \Phi_\varrho M$ | iff $\mathcal{I},t' \models M$ \ for some $t'$ with $t-t' \in \varrho$ |
| $\mathcal{I},t \models \exists_\varrho M$ | iff $\mathcal{I},t' \models M$ \ for all $t'$ with $t-t' \in \varrho$ |
| $\mathcal{I},t \models \exists_\varrho M$ | iff $\mathcal{I},t' \models M$ \ for all $t'$ with $t-t' \in \varrho$ |
| $\mathcal{I},t \models M_1 \wedge_\varrho M_2$ | iff $\mathcal{I},t' \models M_2$ \ for some $t'$ with $t-t' \in \varrho$ \ and $\mathcal{I},t' \models M_1$ \ for all $t'' \in (t',t)$ |
| $\mathcal{I},t \models M_1 \wedge_\varrho M_2$ | iff $\mathcal{I},t' \models M_2$ \ for some $t'$ with $t-t' \in \varrho$ \ and $\mathcal{I},t' \models M_1$ \ for all $t'' \in (t',t')$ |

Table 2: Semantics of ground metric atoms

\textbf{Reasoning.} The main reasoning tasks in DatalogMTL are \textit{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 [Wałega et al., 2019], and ExpSpace-complete for combined complexity, when complexity is measured also with respect to the program [Brandt et al., 2018].

\section{DatalogMTL\textsuperscript{3}}

In this section we introduce DatalogMTL\textsuperscript{3}. We provide it's syntax, two types of semantics, and results on decidability as well as computational complexity of reasoning.

\subsection{Syntax}

We obtain DatalogMTL\textsuperscript{3} by extending DatalogMTL with (temporal) existential rules, analogously to the way Datalog\textsuperscript{3} extends Datalog with existential rules [Cali et al., 2009].

Formally, we let an \textit{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 $x$ of (existential) variables which are mentioned in $M'$ but not in the rule body, so an existential rule is of the form

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

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, $\exists x (Q(x,y) \land R(x,y)) \leftarrow \mathcal{P}(y)$ can be written as three rules: $\exists x \mathcal{P}'(x,y) \leftarrow \mathcal{P}(y)$, $Q(x,y) \leftarrow \mathcal{P}'(x,y)$, and $R(x,y) \leftarrow \mathcal{P}(x,y)$.

An existential rule is \textit{safe} if all its non-existential variables are mentioned in the body. A (DatalogMTL\textsuperscript{3}) program \ is a finite set of existential and non-existential safe rules. An existential rule $r$ is \textit{ground} if all the variables it mentions are existential and its grounding, ground($r$), is the set of all ground rules obtained by assigning constants from Dom to non-existential variables in $r$.

As DatalogMTL\textsuperscript{3} inherits from Datalog\textsuperscript{3} undecidability of reasoning, we will study standard syntactical fragments that are known to be decidable for Datalog\textsuperscript{3} [Fagin et al., 2005; Cali et al., 2013]; in particular, we will consider guarded and weakly acyclic programs.

\textbf{Guarded Programs.} A rule is \textit{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, $\Pi_{\text{ex}}$ from Example 1 is guarded.

\textbf{Weakly Acyclic Programs.} We use a standard definition of a weakly acyclic program [Fagin et al., 2005] involving a \textit{dependency graph}. The dependency graph for a program $\Pi$ has a vertex $v_{\Pi,i}$ for each predicate $P$ in $\Pi$ and for each position $i \in \{1, \ldots, a\}$, where $a$ is the arity of $P$. There is a normal edge from $v_{\Pi,i}$ to $v_{\Pi,j}$ if there is a rule in $\Pi$ with a body atom mentioning $P(s)$ and the head mentioning $Q(s')$ such that the $i$th element of $s$ is a variable, which is the same as the $j$th element of $s'$; Moreover, there is a special edge from $v_{\Pi,i}$ to $v_{\Pi,j}$ if there is an existential rule in $\Pi$ with a body atom mentioning $P(s)$ and the head mentioning $Q(s')$ such that the $i$th element of $s$ is a variable which occurs also in $s'$, and the $j$th element of $s'$ is any existentially quantified variable in $\Pi$. A program is \textit{weakly acyclic} if its dependency graph has no cycle containing a special edge. For instance, $\Pi_{\text{ex}}$ 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.
3.2 Semantics

First, we describe the ‘natural’ semantics (N) of DatalogMTL3, which is based on the standard reading of existential quantification.

Definition 2. Under the natural semantics, an interpretation J satisfies an existential rule r, written as J |=N r, if for every ground rule r' ∈ ground({r}) of the form ∃x M' ← M1, ..., Mn and every time point t in which J satisfies all body atoms M1, ..., Mn, there exists an assignment υ of constants in Dom to variables of M' such that J, t |= υ(M'). Interpretation J satisfies a program Π under the natural semantics, written as J |=N Π, if and only if J |=N r, for each r ∈ Π.

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 [g], then the rule can lead to invention of distinct constants for each t ∈ [g], as illustrated in Example 3.

Example 3. Consider Πex from Example 1 and a dataset Dex with the following facts about vehicles Veh1, Veh2 and orders Ord1, Ord2, Ord3:

Depart(Veh1, Ord1) @0, 10, 
Depart(Veh2, Ord2) @0, 4, 
Drive(Amy, Veh1, Ord1) @0, 10, 
Drive(Amy, Veh2, Ord3) @4, 10.

We have that (Πex, Dex) |=N Dangerous(Veh2) @8, 10. Intuitively, we would also want to deduce that Veh1 is being operated under dangerous conditions, as its driver needs to work for 12 hours to deliver Ord1. This entailment, however, does not hold, since the natural semantics allows for unintended models where Veh1 has many drivers within the interval [0, 12], say a different driver in each of the densely distributed time points t ∈ [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 J satisfies an existential rule r, written as J |=U r, if for every atom M' there exists an assignment υr,M' of constants to variables of M' such that for each ground rule r' ∈ ground({r}) of the form ∃x M' ← M1, ..., Mn and every time point t in which J satisfies all body atoms M1, ..., Mn, we have J, t |= υr,M'(M'). Interpretation J satisfies a program Π under the uniform semantics, written as J |=U Π, if J |=U r for each r ∈ Π.

The uniform semantics provides an intuitive reading of our running example; we observe that now, the program Πex and the dataset Dex from Example 3 entail that Veh1 operated under dangerous conditions, as intended. Indeed, we have (Πex, Dex) |=U Dangerous(Veh1) @8, 12 as the existential rule of Πex is interpreted via an ‘uniform’ assignment that invents the same constant representing a driver of Veh1, within the whole interval [0, 12].

We will also differentiate between the open world assumption (OWA), where the assignments υ 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, DatalogMTL3 under both semantics behaves exactly like Datalog3, as we report formally next.

Proposition 5. Let Π be a DatalogMTL3 program, D a dataset, and M ⊦∅ a relational fact. If (Π, D) |=N M ⊦∅, then (Π, D) |=U M ⊦∅, but the opposite implication does not hold. This property holds under both OWA and CWA.

Proposition 6. Let M ⊦∅ be a fact. Π a DatalogMTL3 program without temporal operators, D a set of facts M' ⊦∅, and D' = {M' | M' ⊦∅ ∈ D}. The following are equivalent: (i) (Π, D) |=N M ⊦∅ under OWA, (ii) (Π, D) |=U M ⊦∅ under OWA, and (iii) Π and D' entail M in Datalog3.

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 DatalogMTL3 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 reducible, 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 $s \in \{U, N\}$, checking $s$-consistency and fact $s$-entailment reduce in logarithmic space to the complement of each other.

Undecidability of $N$- and $U$-consistency in full DatalogMTL follows immediately from the well-known undecidability of Datalog$^3$. Thus, we will consider the guarded and weakly-acyclic fragments, since in the case of Datalog$^3$ both of them are decidable and proved to be useful in numerous applications [Fagin et al., 2005; Cali 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$^3$ 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 $\exists z \text{ Next}(y, z) \leftarrow \text{Next}(x, y)$, which introduces an infinite sequence of constants, that we use to represent tape cells. With the rule $\forall z \text{ Next}(x, y) \leftarrow \text{Next}(x, y)$ 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 $\Phi(0,1)$ in heads [Brandt et al., 2018, Theorem 10]. We show that $\Phi(0,1)Q \leftarrow P$ can be simulated with constants true and false together with rules $\forall x P(x) \leftarrow P'$, $\perp \leftarrow P \land \Phi(0,1)P''(\text{false})$, and $Q \leftarrow P''(\text{true})$.

In contrast, reasoning under the uniform semantics becomes decidable for weakly acyclic programs under OWA as well as for full DatalogMTL programs under CWA. Note that the former result shows that reasoning with programs such as our $\Pi_2$ 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$^3$ programs under OWA and ExpSpace-complete for arbitrary DatalogMTL$^3$ 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$^3$ [Cali 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 $\nu$ over $X$, interpreting existential rules. This allows us to transform the input DatalogMTL$^3$ 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$^3$, 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$^3$ 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$_{1,S}$ [Chomicki and Imieliński, 1988].

### 4 Experimental Evaluation

In this section, we report first experiments with our prototypical implementation of DatalogMTL$^3$ 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\(^3\) 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\(^3\) reasoning.

LARS\(^+\). The most related language to DatalogMTL\(^3\) 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\(^3\), 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 Incld, stand for belt operator, broken gear, and incident Id, respectively

\[
\begin{align*}
\exists y & \text{ Bopr}(x, y) \leftarrow \text{Belt}(x), \\
\exists z & \text{ Brkg}(x, z) \leftarrow \Phi_{[0,5]} \text{Speed}(x, y) \land \text{Slow}(y), \\
\exists z & \text{ Incld}(x, z) \leftarrow \Phi_{[0,3]} \text{Temp}(x, y) \land \text{High}(y), \\
\text{Assign}(y, z) & \leftarrow \text{Incld}(x, y) \land \text{Bopr}(x, z), \\
\text{Block}(x) & \leftarrow \Phi_{[0,3]} \text{Incld}(x, z).
\end{align*}
\]

The structure of some programs in our benchmarks for DatalogMTL\(^3\) 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\(^3\), 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\(^3\)) is interpreted over the rational timeline, which is increases the complexity of reasoning [Walega 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 D\(_1\), D\(_2\), and 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\(^3\) program for detecting dangerous vehicles (where 1 unit on the timeline represents 1 second):

\[
\begin{align*}
\exists d & \text{ Drive}(d, v) \land \text{SpeedExt}(v, s) \leftarrow \Phi_{[0,3]} \text{Speed}(v, s), \\
& \text{Speeding}(v) \leftarrow \text{SpeedExt}(v, s) \land \text{HighSpeed}(s), \\
& \exists z \text{ Incld}(z, v) \leftarrow \Phi_{[0,5]} \text{Speeding}(v), \\
& \text{Inc}(z, d) \land \text{Danger}(v) \leftarrow \text{Incld}(z, v) \land \text{Drive}(d, v), \\
& \text{Danger}(v) \leftarrow \text{Speeding}(v) \land \Phi_{[0,\infty]} \text{Danger}(v).
\end{align*}
\]

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\(^3\) as we explained in Section 3.1.

iT Erdoğan. Our second benchmark exploits iT Erdoğan, 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\(^3\) we proceeded as follows. We started by generating with iT Erdoğan DatalogMTL programs \(\Pi_1, \ldots, \Pi_4\) such that \(\Pi_1\) (3 rules) is non-recursive and linear, \(\Pi_2\) (14 rules) is also non-recursive but non-linear, whereas \(\Pi_3\) (8 rules) and \(\Pi_4\) (25 rules) are both recursive. Next, we extended each \(\Pi_i\), into 5 different DatalogMTL\(^3\) 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\(^3\) programs which, as we confirmed by our experiments, involve non-trivial propagation of existential variables. Moreover, for each \(\Pi_i\), we invoke iT Erdoğan 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\(^3\) programs under the uniform semantics and OWA. Our implementation exploits the well-known Skolemisation technique used for weakly-acyclic Datalog\(^3\) programs [Marnette, 2009; Benedikt et al., 2017], where existential variables \(z\) in rules are replaced by Skolem terms \(f_\exists(x)\) that depend only on the frontier variables \(x\) (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 \(\exists z. B(x, z) \leftarrow \Phi_3 A(x, y)\) is replaced by \(r_1(x)\) resulting in \(B(x, r_1(x)) \leftarrow \Phi_3 A(x, y)\). 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; Walega et al., 2019].

It is worth observing that our practical procedure is not guaranteed to terminate for arbitrary DatalogMTL\(^3\) pro-
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 [Bel-lomarini et al., 2018; 2022a], which allows for reasoning over Datalog\(^4\), but not in DatalogMTL\(^3\).

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\(^3\) 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 \(D_1\), \(D_2\), and \(D_3\), respectively. The results for the \(i\)Temporal 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 \(i\)Temporal, those are the average times computed for all five DatalogMTL\(^3\) programs obtained by extending a particular program \(\Pi_i\). 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 \(\Pi_i\).

Although the theoretical complexity of DatalogMTL\(^3\) 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 \(i\)Temporal, 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 \(i\)Temporal benchmark are higher for the SUMO benchmark, as both programs and datasets in the \(i\)Temporal are significantly larger. Nevertheless, even for the hardest case, where the programs (obtained by extending \(\Pi_i\)) 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 \(\Pi_4\) we observe a large standard derivation compared to the other instances. A detailed investigation showed that one of the DatalogMTL\(^3\) programs obtained from \(\Pi_4\) 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\(^3\) 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 \(\Pi_i\) we have generated several different DatalogMTL\(^3\) programs.

<table>
<thead>
<tr>
<th>(\Pi_i)</th>
<th>small</th>
<th>medium</th>
<th>large</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Pi_1)</td>
<td>88 (6)</td>
<td>738 (41)</td>
<td>7196 (225)</td>
</tr>
<tr>
<td>(\Pi_2)</td>
<td>464 (50)</td>
<td>455 (38)</td>
<td>4154 (463)</td>
</tr>
<tr>
<td>(\Pi_3)</td>
<td>161 (21)</td>
<td>1385 (151)</td>
<td>13701 (1019)</td>
</tr>
<tr>
<td>(\Pi_4)</td>
<td>273 (133)</td>
<td>2211 (1181)</td>
<td>16857 (8727)</td>
</tr>
</tbody>
</table>

Table 3: Results for increasing datasets in \(i\)Temporal; reported numbers are average times for five runs in ms whereas standard deviations are located in parentheses

5 Conclusions

We have introduced DatalogMTL\(^3\), 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\(^3\) 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\(^3\) 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. RPR1201074). 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


