Stream Reasoning in DatalogMTL via Finite Materialisation

Przemysław (Przemek) Wałęga, Michał Zawidzki, and Bernardo Cuenca Grau

Stream Reasoning Workshop, 2022

Motivations

- DatalogMTL is a powerful extension of Datalog with metric temporal operators.
- DatalogMTL programs allow for **recursive propagation** of information (towards future and past) which is **problematic for stream reasoning**.

Question:

- How to *guarantee* that a DatalogMTL program has *no infinite recursion via time*?

Contributions

- We *define data dependent and data independent notions* of ‘no infinite recursion via time’.
- We show *algorithms* for checking if a DatalogMTL program satisfies above properties.
- We show *tight complexity bounds* for these problems.
- We show *sufficient conditions* which are easy to check.
- We show *tight complexity bounds* for reasoning in programs without recursion via time.

DatalogMTL

Dataset

A *dataset* consists of facts over rational intervals, e.g.:

Fraud detection example:

$$RedList(adam)@[0, 20]$$
$$HighRisk(david)@[0, 500]$$
$$HighRisk(ernesto)@[0, 500]$$
$$Transaction(adam, betty)@2.87$$
$$Transaction(betty, charlie)@12.15$$
$$Transaction(charlie, david)@17.5$$
$$Transaction(charlie, ernesto)@129.43$$

Program

Programs use MTL operators, e.g.,

$$\diamondplus_{(0,100]} A \text{ at } t \quad \Leftrightarrow \quad A \text{ is true at } \textit{some moment in } (t+0, t+100]$$

$$\boxplus_{(0,100]} A \text{ at } t \quad \Leftrightarrow \quad A \text{ is true at } \textit{every moment in } (t+0, t+100]$$

A *program* is a set of rules $B \leftarrow A_1 \wedge \cdots \wedge A_n$ where:

$$A := \top \mid \bot \mid P(\boldsymbol{c}) \mid \diamondminus_\varrho A \mid \diamondplus_\varrho A \mid \boxminus_\varrho A \mid \boxplus_\varrho A \mid A \mathcal{S}_\varrho A \mid A \mathcal{U}_\varrho A$$

$$B := \top \mid \bot \mid P(\boldsymbol{c}) \mid \boxminus_\varrho B \mid \boxplus_\varrho B$$

Fraud detection example:

$$\mathit{TransactionChain}(x,y) \leftarrow \mathit{Transaction}(x,y) \wedge \mathit{RedList}(x)$$

$$\mathit{TransactionChain}(x,z) \leftarrow \diamondminus_{[0,24]} \mathit{TransactionChain}(x,y) \wedge \mathit{Transaction}(y,z)$$

$$\boxplus_{[0,100)} \mathit{Suspect}(y) \leftarrow \mathit{TransactionChain}(x,y) \wedge \mathit{HighRisk}(y)$$

Reasoning

Main reasoning tasks (we consider the rational timeline):

- *Fact entailment*: do a program Π and a dataset $\mathcal{D}$ entail a fact, e.g., $Suspect(adam)@100$?
- *Consistency checking*: do Π and $\mathcal{D}$ have a model?

Theorem. Reasoning in DatalogMTL is:

- ExpSpace-*complete* for combined complexity,
- PSpace-*complete* for data complexity.

RECURSION VIA TIME

Canonical Interpretation

Unlike in Datalog:

- materialisation in DatalogMTL **can require infinitely many steps of rule application**.

Canonical Interpretation

Unlike in Datalog:

- materialisation in DatalogMTL **can require infinitely many steps of rule application**.

Example:

$\mathcal{D} = \{\text{Even}@0\}$

$\Pi = \{\boxplus_{[1,1]}\text{Odd} \leftarrow \text{Even}, \quad \boxplus_{[1,1]}\text{Even} \leftarrow \text{Odd}\}$

Step 0:

Canonical Interpretation

Unlike in Datalog:

- materialisation in DatalogMTL **can require infinitely many steps of rule application**.

Example:

$\mathcal{D} = \{\mathsf{Even}@0\}$

$\Pi = \{\boxplus_{[1,1]}\mathsf{Odd} \leftarrow \mathsf{Even}, \quad \boxplus_{[1,1]}\mathsf{Even} \leftarrow \mathsf{Odd}\}$

Step 1:

Canonical Interpretation

Unlike in Datalog:

- materialisation in DatalogMTL **can require infinitely many steps of rule application**.

Example:

$\mathcal{D} = \{\mathsf{Even}@0\}$

$\Pi = \{\boxplus_{[1,1]}\mathsf{Odd} \leftarrow \mathsf{Even}, \quad \boxplus_{[1,1]}\mathsf{Even} \leftarrow \mathsf{Odd}\}$

Step 2:

Canonical Interpretation

Unlike in Datalog:

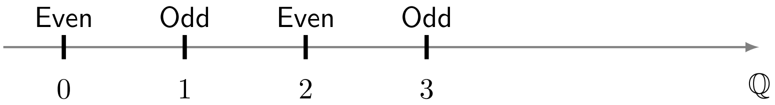
- materialisation in DatalogMTL **can require infinitely many steps of rule application**.

Example:

$\mathcal{D} = \{\text{Even}@0\}$

$\Pi = \{\boxplus_{[1,1]}\text{Odd} \leftarrow \text{Even}, \quad \boxplus_{[1,1]}\text{Even} \leftarrow \text{Odd}\}$

Step 3:



Finitely Materialisable Programs

- How can we *guarantee that no infinite materialisation* occurs?

We say that a program Π is *finitely materialisable* for a dataset $\mathcal{D}$ if materialisation of Π and $\mathcal{D}$ takes a finite number of steps.

We say that Π is *finitely materialisable* for all datasets if materialisation of Π and any dataset $\mathcal{D}$ takes a finite number of steps.

ALGORITHMS AND COMPLEXITY

Checking Finite-materialisability for a Single Dataset

Theorem. If Π is finitely materialisable for $\mathcal{D}$, then all the facts they entail are inside $[t_{\mathcal{D}}^{\min} - \mathsf{offset}(\Pi, \mathcal{D}), t_{\mathcal{D}}^{\max} + \mathsf{offset}(\Pi, \mathcal{D})]$.

Algorithm 1: Checking finite materialisability for a single dataset

Input: A program Π and a dataset $\mathcal{D}$

Output: A Boolean value

$\mathcal{D}_{\mathsf{new}} := \mathcal{D}$;
$\varrho := [t_{\mathcal{D}}^{\min} - \mathsf{offset}(\Pi, \mathcal{D}), t_{\mathcal{D}}^{\max} + \mathsf{offset}(\Pi, \mathcal{D})]$;
repeat
    $\mathcal{D}_{\mathsf{old}} := \mathcal{D}_{\mathsf{new}}$;
    $\mathcal{D}_{\mathsf{new}} := \mathsf{ApplyRules}(\Pi, \mathcal{D}_{\mathsf{old}})$;
    if *there is* $M@\varrho' \in \mathcal{D}_{\mathsf{new}}$ *with* $\varrho' \not\subseteq \varrho$ **then** Return false;
until $\mathfrak{I}_{\mathcal{D}_{\mathsf{old}}} = \mathfrak{I}_{\mathcal{D}_{\mathsf{new}}}$;
return true;

Checking Finite-materialisability for a Single Dataset

Theorem. Algorithm 1 returns 'true' iff Π is finitely materialisable for $\mathcal{D}$.

Theorem. Algorithm 1 works in doubly exponential time in the size of Π and $\mathcal{D}$.

Checking Finite-materialisability for a Single Dataset

Theorem. Algorithm 1 returns 'true' iff Π is finitely materialisable for $\mathcal{D}$.

Theorem. Algorithm 1 works in doubly exponential time in the size of Π and $\mathcal{D}$.

The algorithm is practically efficient, but **it is not worst-case optimal**:

Theorem. Checking finite materialisability for a given dataset is ExpSpace-complete for combined and PSpace-complete for data complexity.

Checking Finite-materialisability for All Datasets

- Checking data-independent finite-materialisability *reduces to checking the data-dependent variant* for a **critical dataset** defined as follows:

Definition. **Critical dataset** $\mathcal{D}_\Pi$ for Π contains all facts $P(\mathbf{s})@[0, \mathsf{depth}(\Pi)]$, where:

- P occurs in Π,
- $\mathbf{s}$ mentions constants from Π and a single fresh constant.

Checking Finite-materialisability for All Datasets

- Checking data-independent finite-materialisability *reduces to checking the data-dependent variant* for a **critical dataset** defined as follows:

Definition. **Critical dataset** $\mathcal{D}_\Pi$ for Π contains all facts $P(\mathbf{s})@[0, \mathsf{depth}(\Pi)]$, where:

- P occurs in Π,
- $\mathbf{s}$ mentions constants from Π and a single fresh constant.

Theorem. Π is finitely materialisable for all datasets iff it is for $\mathcal{D}_\Pi$.

- If materialisation of Π and some $\mathcal{D}$ takes infinitely many steps, then the same holds for Π and $\mathcal{D}_\Pi$. This is guaranteed by the definition of $\mathcal{D}_\Pi$, in particular, by choosing the sufficiently long interval $[0, \mathsf{depth}(\Pi)]$.

Checking Finite-materialisability for All Datasets

Theorem. If Π is finitely materialisable for all datasets, facts entailed by Π and any $\mathcal{D}$ are in $[t_{\mathcal{D}}^{\min} - \text{offset}(\Pi), t_{\mathcal{D}}^{\max} + \text{offset}(\Pi)]$.

Algorithm 2: Checking finite materialisability for all datasets

Input: A program Π

Output: A Boolean value

```
D_new := D_Π;
ϱ := [t^min_{D_Π} − offset(Π), t^max_{D_Π} + offset(Π)];
repeat
    D_old := D_new;
    D_new := ApplyRules(Π, D_old);
    if there is M@ϱ' ∈ D_new with ϱ' ⊈ ϱ then Return false;
until ℑ_{D_old} = ℑ_{D_new};
return true;
```

Checking Finite-materialisability for All Datasets

Theorem. Algorithm 2 returns 'true' iff Π is finitely materialisable for all datasets.

Theorem. Algorithm 2 runs in exponential time in the size of Π.

Checking Finite-materialisability for All Datasets

Theorem. Algorithm 2 returns 'true' iff Π is finitely materialisable for all datasets.

Theorem. Algorithm 2 runs in exponential time in the size of Π.

Moreover, our algorithm *is worst-case optimal*:

Theorem. Checking finite materialisability (for all datasets) is $\textsc{ExpTime}$-complete.

- If materialisation of Π and $\mathcal{D}$ takes finitely many steps, then the number of these steps is exponential in the size of Π. Thus, it suffices to check if the number of steps in materialisation of Π and $\mathcal{D}_\Pi$ exceeds our exponential bound.

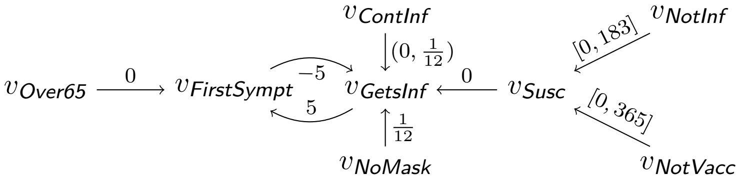
SUFFICIENT CONDITIONS

MTL-acyclicity

Consider the program:

$$\mathit{Susc}(x) \leftarrow \boxminus_{[0,365]} \mathit{NotVacc}(x) \land \boxminus_{[0,183]} \mathit{NotInf}(x),$$
$$\mathit{GetsInf}(x) \leftarrow \mathit{ContInf}(x,y) \mathcal{S}_{\frac{1}{12}} \mathit{NoMask}(x) \land \mathit{Susc}(x),$$
$$\mathit{FirstSympt}(x) \leftarrow \boxminus_{5} \mathit{GetsInf}(x) \land \mathit{Over65}(x),$$
$$\boxminus_{5} \mathit{GetsInf}(x) \leftarrow \mathit{FirstSympt}(x).$$

It's *metric dependency graph* is:



There is no cycles with weight $\neq [0,0]$, so the program is *MTL-acyclic*.

MTL-acyclicity

Theorem. MTL-acyclic programs are finitely materialisable (for all datasets).

Theorem. Checking if a program is MTL-acyclic is NL-complete.

- Hence, MTL-acyclicity is an *easy to check sufficient condition* for finite materialisability.

COMPLEXITY OF REASONING

Reasoning in Finitely Materialisable Programs

Theorem. Fact entailment in finitely materialisable programs is ExpTime-complete for combined and PSpace-complete for data complexity.

Observe that:

- DatalogMTL is ExpSpace-complete for combined and PSpace-complete for data complexity.
- Datalog is ExpTime-complete for combined and P-complete for data complexity.

Moreover:

- Finitely materialisable DatalogMTL programs *strictly contain Datalog* programs.
- However, reasoning in finitely materialisable DatalogMTL programs has the *same combined complexity as Datalog*.

Conclusions

We introduced a class of *finitely materialisable* DatalogMTL *programs*:

- that are naturally amenable to materialisation-based reasoning via scalable forward chaining techniques,
- for which we provided a membership check and an easy to verify sufficient condition,
- in which reasoning is no harder (for combined complexity) than in pure Datalog.

Thank you for your attention

Przemysław Wałęga | www.walega.pl | przemyslaw.walega@cs.ox.ac.uk

Our results on finitely materialisable DatalogMTL programs:

Theorem. Checking finite materialisability for a given dataset is ExpSpace-complete for combined and PSpace-complete for data complexity.

Theorem. Checking finite materialisability (for all datasets) is ExpTime-complete.

Theorem. MTL-acyclic programs are finitely materialisable (for all datasets).

Theorem. Checking if a program is MTL-acyclic is NL-complete.

Theorem. Fact entailment in finitely materialisable programs is ExpTime-complete for combined and PSpace-complete for data complexity.