Proceedings of the Sixteenth International Conference on
Principles of Knowledge Representation and Reasoning (KR 2018)

The Window Validity Problem in Rule-Based Stream Reasoning

Alessandro Ronca, Mark Kaminski, Bernardo Cuenca Grau, Ian Horrocks
Department of Computer Science, University of Oxford, UK
{alessandro.ronca, mark.kaminski, bernardo.cuenca.grau, ian.horrocks}@cs.ox.ac.uk

Abstract

Rule-based temporal query languages provide the expressive power and flexibility required to capture in a natural way complex analysis tasks over streaming data. Stream processing applications, however, typically require near real-time response using limited resources. In particular, it becomes essential that the underpinning query language has favourable computational properties and that stream processing algorithms are able to keep only a small number of previously received facts in memory at any point in time without sacrificing correctness. In this paper, we propose a recursive fragment of temporal Datalog with tractable data complexity and study the properties of a generic stream reasoning algorithm for this fragment. We focus on the *window validity problem* as a way to minimise the number of time points for which the stream reasoning algorithm needs to keep data in memory at any point in time.

1 Introduction

Query processing over streams is becoming increasingly important for data analysis in domains as diverse as financial trading (Nuti et al. 2011), equipment maintenance (Cosad et al. 2009), or network security (Münz and Carle 2007).

A growing body of research has recently focused on extending traditional stream management systems with reasoning capabilities (Barbieri et al. 2010; Calbimonte, Corcho, and Gray 2010; Anicic et al. 2011; Le-Phuoc et al. 2011; Zaniolo 2012; Özçep, Möller, and Neuenstadt 2014; Beck et al. 2015; Dao-Tran, Beck, and Eiter 2015; Ronca et al. 2018). Languages well-suited for stream reasoning applications are typically rule-based, where prominent examples include *temporal Datalog* (Chomicki and Imieliński 1988) and *DatalogMTL* (Brandt et al. 2017). These core languages are powerful enough to capture many other temporal formalisms (Abadi and Manna 1989; Baudinet, Chomicki, and Wolper 1993) and provide the logical underpinning for other expressive languages proposed in the stream reasoning literature (Zaniolo 2012; Beck et al. 2015).

Rules provide the expressive power and flexibility required to naturally capture in a declarative way complex analysis tasks over streaming data. This is illustrated by the following example in network security, where intrusion detection policies (IDPs) are represented in temporal Datalog.

Example 1. *Consider a computer network which is being monitored for external threats. Bursts (unusually high amounts of data) between any pair of nodes in the network are detected by specialised monitoring devices and streamed to the network's management centre as timestamped facts. A monitoring task in the centre is to identify nodes that may have been hacked according to a specific IDP, and add them to a blacklist of nodes. In this setting, one may want to know the contents of the blacklist at any given point in time in order to decide on further action. This task is captured by a temporal Datalog query consisting of the rules given next and where* Black *is the designated output predicate:*

$$\mathsf{Brst}(x,y,t) \wedge \mathsf{Brst}(z,y,t+1) \rightarrow \mathsf{Attk}(x,y,t+1) \quad (1)$$

$$\mathsf{Attk}(x,y,t) \wedge \mathsf{Attk}(x,y,t+1) \wedge \mathsf{Attk}(x,y,t+2) \rightarrow \mathsf{Black}(x,t+2) \quad (2)$$

$$\mathsf{Black}(x,t) \rightarrow \mathsf{Black}(x,t+1) \quad (3)$$

$$\mathsf{Attk}(x,y,t) \rightarrow \mathsf{Grey}(x,max,t) \quad (4)$$

$$\mathsf{Grey}(x,i,t) \wedge \mathsf{Succ}(j,i) \rightarrow \mathsf{Grey}(x,j,t+1) \quad (5)$$

$$\mathsf{Grey}(x,i,t) \wedge \mathsf{Brst}(x,y,t) \rightarrow \mathsf{Black}(x,t) \quad (6)$$

Rule (1) *identifies two consecutive bursts from nodes v and v'' to a node v' in the network as an attack on v' originated by v. Rule* (2) *implements an IDP where three consecutive attacks from v on v' result in v being added to the blacklist, where it remains indefinitely (Rule* (3)*). Rules* (4)–(6) *implement a second IDP where an attack from v on any node leads to v being identified as suspicious and added to a "greylist". Such list comes with a succession of decreasing warning levels, where the maximum is represented by the constant max and where the relationship from each level to the next is captured by a binary, non-temporal,* Succ *predicate. As time goes by, the warning level decreases; however, if at any point during this process node v generates another burst to any other node in the network, then it gets blacklisted.*

Stream processing applications typically require near real-time response using limited resources; this becomes especially challenging in the context of rule-based stream reasoning due to the following reasons:

1. Fact entailment over temporal rule languages is typically intractable in data complexity—ExpSpace-complete in

the case of DatalogMTL and PSPACE-complete in the case of temporal Datalog. Furthermore, known tractable fragments are non-recursive (Ronca et al. 2018; Brandt et al. 2017), which limits their applicability to certain data analysis applications.

2. In order to adhere to memory limitations and scalability requirements, systems can only keep a limited history of previously received input facts in memory to perform further computations. Rules, however, can propagate derived information both towards past and future time points and hence query answers can depend on data that has not yet been received as well as on data that arrived far in the past. This may force the system to keep in memory a very large (or even unbounded) input history to ensure correctness.

We address the first challenge by introducing in Section 3 the language of *forward-propagating queries*—a fragment of temporal Datalog that extends plain (non-temporal) Datalog by allowing unrestricted recursive propagation of information into future time points, while at the same time precluding propagation of derived facts towards past time points. Our language is sufficiently expressive to capture interesting analysis tasks over streaming data, such as the one illustrated in our previous example. Moreover, we show that forward-propagating queries can be answered in polynomial time in the size of the input data and hence they are well-suited for data-intensive applications.

To address the second challenge, we take as a starting point a generic algorithm which accepts as input a set of non-temporal background facts and a stream of timestamped facts and outputs as a stream the answers to a forward-propagating query Q. The algorithm is parametrised by a window size w and a signature Σ, which determine the set of facts stored in memory by the algorithm at any point in time. As the algorithm receives the input stream at time point τ, it computes all implicit Σ-facts and answers to Q holding at τ using only the facts held in memory, and subsequently discards all stored facts holding at $\tau - w$. For the algorithm to be correct, the computed answers for each τ over the restricted set of facts in memory must coincide with the answers over the entire stream. Such an assurance, however, can only be given for certain values of Σ and if the window parameter w is large enough so that facts that may influence answers at later time points are not discarded too early. This motivates the *window validity problem*, which is to decide whether a given window w is valid for a given query Q and signature Σ in the sense that the aforementioned correctness guarantee holds for any input data.

In our prior work (Ronca et al. 2018), we considered an instantiation of the generic stream reasoning algorithm where only explicit facts from the input stream (and hence no entailed facts) are kept in memory. This setting can, however, be problematic in the presence of recursion, in that a recursive query may not admit a valid window. Stream reasoning clearly becomes impractical for such queries since the entire stream received so far must be kept in memory by the algorithm in order to ensure correctness.

To address this limitation, we consider in Section 4 a *full materialisation* variant of the algorithm in which all facts (explicit or implicit) over the entire signature are kept in memory; as a result, when a fact is discarded by the algorithm, its consequences at later time points are not lost. In this setting, we can show that a valid window is guaranteed to exist for any forward-propagating query, and a (possibly larger than needed) window can be obtained syntactically by inspection of the query. From a practical perspective, however, it is important to have a valid window that is as small as possible since the number of facts entailed by the query's rules at any given time point can be very large. Thus, we investigate in Sections 5 and 6 the computational properties of window validity in this revised setting.

In Section 5, we show that window validity and query containment are interreducible problems, and hence known complexity bounds on temporal query containment transfer directly. In particular, undecidability of window validity for forward-propagating queries follows from the undecidability of query containment for non-temporal Datalog.

To regain decidability, we consider in Section 6 the situation where the set of relevant domain objects can be fixed in advance, in the sense that input facts can refer only to those objects. In Example 1, this assumption amounts to fixing both the nodes in the network and the grey list's warning levels, and requiring that all input facts mention only these objects. This assumption allows us to ground the non-temporal variables of the query to a set of known objects; such grounding is exponential and results in an *object-ground* query where all variables are temporal. We show that the window problem is PSPACE-complete for object-ground forward-propagating queries and coNP-complete if the query is also non-recursive. This immediately gives us an EXPSPACE upper bound (coNEXP if queries are additionally assumed to be non-recursive) for the fixed-domain window validity problem; we then prove that these bounds are tight. Our results show that, although window validity is undecidable, we can obtain decidability under reasonable assumptions on the input data. Even under such assumptions, the problem is computationally intractable; however, queries can be assumed to be relatively small in practice and windows can be computed offline, prior to receiving any data.

Finally, for applications where one cannot assume the object domain to be fixed in advance, we propose in Section 7 a sufficient condition for the validity of a window that can be checked in exponential time without additional assumptions.

Due to space limitations, our technical results are accompanied with a sketch outlining the main ideas behind each proof. Further details are deferred to an extended version of this paper available online at `arXiv:1808.02291`.

2 Preliminaries

We recapitulate temporal Datalog (Chomicki and Imieliński 1988) as a basic language for stream reasoning.

Syntax A signature consists of predicates, constants and variables, where constants are partitioned into objects and non-negative integer time points and variables are partitioned into object variables and time variables. An object term is an object or an object variable. A time term is a time

point, a time variable, or an expression of the form $t + k$ with t a time variable, k an integer, and $+$ the integer addition function.

Predicates are partitioned into *extensional* (EDB) and *intensional* (IDB) and they come with a non-negative integer *arity* n, where each position $1 \leq i \leq n$ is of either *object* or *time sort*. A predicate is *rigid* if all its positions are of object sort and it is *temporal* if the last position is of time sort and all other positions are of object sort. An *atom* is an expression $P(s_1, \ldots, s_n)$ where P is a n-ary predicate and each s_i is a term of the required sort; we sometimes use the term P-atom to refer to an atom with predicate P. A *rigid* atom (respectively, temporal, IDB, EDB) is an atom over a rigid predicate (respectively, temporal, IDB, EDB).

A *rule* r is of the form $\bigwedge_i \alpha_i \to \alpha$, where α and each α_i are rigid or temporal atoms, and α is IDB whenever $\bigwedge_i \alpha_i$ is non-empty. Atom $\mathrm{head}(r) = \alpha$ is the *head* of r, and $\mathrm{body}(r) = \bigwedge_i \alpha_i$ is the *body* of r. Rules are *safe*—that is, all variables occur in the body. A *program* Π is a finite set of rules. A term, atom, rule, or program is *ground* if it has no variables. A predicate P is Π-*dependent* on predicate P' if Π has a rule with P in the head and P' in the body. A *fact* is a ground, function-free rigid or temporal atom, and a *dataset* is a (possibly infinite) set of EDB facts. Each fact α corresponds to a rule having empty body and α in the head, so we use α and its corresponding rule interchangeably.

A *query* is a pair $Q = \langle P_Q, \Pi_Q \rangle$ with Π_Q a program and P_Q an IDB *output predicate* in Π_Q not occurring in the body of any rule in Π_Q. We also denote with Σ_Q the set of all IDB predicates in Π_Q. Query Q is

- *temporal* if P_Q is a temporal predicate;
- *Datalog* if no temporal predicate occurs in Π_Q;
- *object-ground* if Π_Q has no object variables; and
- *non-recursive* if the directed graph induced by the Π_Q-dependencies is acyclic.

Semantics Rules are interpreted as universally quantified first-order sentences. A Herbrand interpretation $\mathcal{H}$ is a (possibly infinite) set of facts. It *satisfies* a rigid atom α if $\alpha \in \mathcal{H}$, and it satisfies a temporal atom β if evaluating the addition function in β yields a fact in $\mathcal{H}$. Satisfaction is extended to conjunctions of ground atoms, rules and programs in the standard way. If $\mathcal{H} \models \Pi$, then $\mathcal{H}$ is a *model* of Π. Program Π *entails* a fact α, written $\Pi \models \alpha$, if $\mathcal{H} \models \Pi$ implies $\mathcal{H} \models \alpha$. The set of *answers* to a query Q over a dataset D, written $Q(D)$, consists of each P_Q-fact α such that $\Pi_Q \cup D \models \alpha$.

Reasoning We next define two basic reasoning problems, which we parametrise to specific classes of input queries $\mathcal{Q}$ and datasets $\mathcal{D}$. Similarly to (Chomicki and Imieliński 1988), we assume from now onwards in all reasoning problems that numbers in input queries and datasets are coded in unary; our complexity results may (and almost certainly will) change if binary encoding is assumed, and we leave this investigation for future work. Furthermore, we make the following general assumptions for each $\mathcal{D}$: *(1)* for each $D \in \mathcal{D}$ and each finite subset S of D there is a finite $D' \in \mathcal{D}$ such that $S \subseteq D' \subseteq D$; and *(2)* for each $D \in \mathcal{D}$ and unary temporal fact α, we have $D \cup \{\alpha\} \in \mathcal{D}$. The former property is a form of compactness closure, whereas the latter is a closure property under addition of unary temporal facts.

The *query evaluation problem* $\textsc{Eval}^{\mathcal{Q}}_{\mathcal{D}}$, for $\mathcal{Q}$ a class of queries and $\mathcal{D}$ a class of finite datasets, is to check whether $\alpha \in Q(D)$ for α an input fact, $Q \in \mathcal{Q}$ and $D \in \mathcal{D}$; the *data complexity* of $\textsc{Eval}^{\mathcal{Q}}_{\mathcal{D}}$ is the complexity for fixed Q. Query evaluation for arbitrary datasets is PSPACE-complete in data complexity under unary encoding of numbers (Chomicki and Imieliński 1988), and in AC^0 for non-recursive queries.

Let Q_1 and Q_2 be queries having the same output predicate. Then, Q_1 is *contained* in Q_2 with respect to $\mathcal{D}$, written $Q_1 \sqsubseteq_{\mathcal{D}} Q_2$, if $Q_1(D) \subseteq Q_2(D)$ for each $D \in \mathcal{D}$. The *containment problem* $\textsc{Cont}^{\mathcal{Q}}_{\mathcal{D}}$ is to check $Q_1 \sqsubseteq_{\mathcal{D}} Q_2$ for given $Q_1, Q_2 \in \mathcal{Q}$. For simplicity, we drop $\mathcal{D}$ from $Q_1 \sqsubseteq_{\mathcal{D}} Q_2$ and $\textsc{Cont}^{\mathcal{Q}}_{\mathcal{D}}$ (respectively, from $\textsc{Eval}^{\mathcal{Q}}_{\mathcal{D}}$) whenever $\mathcal{D}$ is the class of all datasets (respectively, of all finite datasets).

Our definition of containment considers infinite datasets, which is required to capture streams. This does not change the nature of the problem due to the properties of first-order logic and our assumptions on $\mathcal{D}$; in particular, $Q_1 \sqsubseteq_{\mathcal{D}} Q_2$ if and only if $Q_1 \sqsubseteq_{\mathcal{D}'} Q_2$ with $\mathcal{D}'$ the class consisting of all finite datasets in $\mathcal{D}$. By standard results in nontemporal Datalog, it follows that unrestricted containment is undecidable (Shmueli 1993), and it is coNEXP-hard for non-recursive queries (Benedikt and Gottlob 2010).

3 Forward-Propagating Queries

Stream processing applications are data-intensive, requiring fast response using limited resources. Tractability of query evaluation in data complexity is thus a key requirement for logics underpinning stream reasoning systems. Query evaluation in temporal Datalog is, however, PSPACE-complete in data complexity, which limits its applicability.

In this section we introduce the language of forward-propagating queries—a fragment of temporal Datalog which allows unrestricted recursive propagation of derived facts into the present and future time points, while at the same time precluding propagation towards past time points.

Definition 2. *The* offset *of a time term s equals zero if s is a time variable, and it equals k if s is the time point k or a time term of the form $t + k$. The* radius *of a rule is zero if its head is rigid, and it is the maximum difference between the offset of its head time argument and the offset of a body time argument otherwise. A rule r is* forward-propagating *if it is Datalog, or it satisfies all of the following properties:*

- *it contains no time points;*
- *it has a single time variable, which occurs in the head;*
- *its radius is non-negative.*

A query Q is forward-propagating, or an fp-query *for short, if so is each rule in Π_Q. The radius of Q is the maximum radius amongst the rules in Π_Q. For $k \geq 0$, we denote as Q^k the query $\langle P_Q, \Pi^k_Q \rangle$ with Π^k_Q the subset of rules in Π_Q with radius at most k.*

We denote the class of fp-queries as FP*, and let* OG*,* NR*, and* OGNR *be the subclasses of* FP *where queries are required to be object-ground, non-recursive, and both object-ground and non-recursive, respectively.*

Example 3. *The query in our running Example 1 is forward-propagating. Its radius is two, which is justified by Rule* (2)*, where the offset of the head is two and the offset of the first body atom is zero.*

The conditions in Definition 2 ensure that the derivation via rule application of a fact α holding at a time point τ can be justified by facts holding at time points no greater than τ; as a result, one can safely disregard all facts holding after τ for the purpose of deriving α.

The restrictions imposed by Definition 2 are sufficient to ensure tractability of query evaluation, while at the same time allowing for temporal recursion. The following theorem shows a stronger result, namely that query evaluation over fp-queries can be reduced to query evaluation over standard non-temporal Datalog.

Theorem 4. *Let $\mathcal{D}$ be a class of finite datasets, let $\mathcal{Q} \in \{\text{FP}, \text{NR}, \text{OG}, \text{OGNR}\}$, and let $\mathcal{Q}'$ be the Datalog subset of $\mathcal{Q}$. Then,* $\text{EVAL}^{\mathcal{Q}}_{\mathcal{D}}$ *is* LOGSPACE*-reducible to* $\text{EVAL}^{\mathcal{Q}'}_{\mathcal{D}}$.

Proof sketch. To check whether $\Pi_Q \cup D$ entails fact α holding at a time point τ, it suffices to consider facts (explicitly given or derived) holding at time points in the interval between the minimum time point mentioned in D and τ; such interval contains linearly-many time points due to τ being encoded in unary. We can then transform Π_Q in LOGSPACE into a plain Datalog program Π' by first introducing an object for each time point in the interval, and then grounding the temporal arguments of all rules in Π_Q over these objects. Clearly, it holds that $\Pi_Q \cup D$ entails α if so does $\Pi' \cup D$. □

Theorem 4 allows us to immediately transfer known complexity bounds for query evaluation over different classes of Datalog queries to the corresponding class of fp-queries—see, e.g., (Dantsin et al. 2001; Vorobyov and Voronkov 1998). In particular, it follows that evaluation of fp-queries is tractable in data complexity.

Corollary 5. *The following complexity bounds hold for the query evaluation problem over classes of fp-queries:*
– EVAL^{FP} *is* EXP-complete *and* P-complete *in data;*
– EVAL^{NR} *is* PSPACE-complete *and in* AC^0 *in data; and*
– EVAL^{OG} *is* P-complete*.*

4 A Generic Stream Reasoning Algorithm

A stream reasoning algorithm receives as input an unbounded *stream* S of timestamped facts and a set B of rigid *background facts*, and outputs (also as a stream) the answers to a standing temporal query Q, which is considered fixed. Algorithm 1, which we describe next, is a generic such algorithm that is applicable to any fp-query. In the algorithm (as well as in the rest of the paper), we denote with $F\restriction_{[\tau,\tau']}$ the subset of temporal facts in a dataset F holding in the interval $[\tau, \tau']$, and write $F\restriction_{\tau}$ for $F\restriction_{[\tau,\tau]}$. Furthermore, from now on we will silently assume all queries to be temporal.

Algorithm 1 is parametrised by an fp-query Q, a non-negative integer *window size* w and a signature Σ, where the latter two parameters determine the set of facts M kept in memory by the algorithm at any point in time. The algorithm is initialised in Line 1, where the input set B of rigid background facts is loaded into memory and the current time τ is set to zero. The core of the algorithm is an infinite loop, where each iteration consists of the following four steps and the current time τ is incremented at the end of each iteration.

Algorithm 1: A generic stream reasoning algorithm

Parameters: Temporal fp-query Q, window size w, and a subset Σ of the IDBs in Q with $P_Q \in \Sigma$.
Input: Background dataset B, stream S.

```
1 Assign M := B and τ := 0.
2 loop
3     Receive S⌈τ and assign M := M ∪ S⌈τ.
4     Add to M all Σ-facts α holding at τ s.t.
        Π_Q ∪ M ⊨ α.
5     Stream out all P_Q-facts in M⌈τ.
6     If τ ≥ w, remove from M all facts in M⌈τ−w.
7     τ := τ + 1.
8 end
```

1. The batch of input stream facts holding at τ is received and loaded into memory (Line 3).
2. All implicit facts over the relevant signature Σ holding at τ are computed and materialised in memory (Line 4).
3. Query answers holding at τ are read from memory and streamed out (Line 5);
4. All facts (explicit in S or implicitly derived) holding at $\tau - w$ are removed from memory (Line 6).

In order to favour scalability, Algorithm 1 restricts at any point in time the set of facts kept in memory and therefore considered for query evaluation. This, however, carries the obvious risk that valid answers holding over the entire stream may be missed by the algorithm if the facts they depend on are removed from memory too early. Therefore, the window size of the algorithm should be chosen so that the following correctness property is satisfied.

Definition 6. *A window size w is* valid *for an fp-query Q, a signature Σ, and a class $\mathcal{D}$ of datasets if, when parametrised with Q, w and Σ, and for each input $\langle B, S\rangle$ with $B \cup S \in \mathcal{D}$ and each $n > 0$, the set of facts streamed out by Algorithm 1 in the first n iterations coincides with $Q(B \cup S)\restriction_{[0,n-1]}$.*

In prior work (Ronca et al. 2018) we considered an algorithm that does not keep derived facts (other than possibly query answers) in memory and thus only stores EDB facts from the input stream. When applied to an fp-query Q, the algorithm in our previous work can be seen as a variant of Algorithm 1 where $\Sigma = \{P_Q\}$. This variant of Algorithm 1 is, however, problematic for recursive queries since no valid window size may exist, in which case the entire stream received so far must be kept in memory to ensure correctness.

Proposition 7. *There exists no valid window size for the object-ground fp-query Q where, for A an EDB predicate, $\Pi_Q = \{A(t) \to B(t); B(t) \to B(t+1); B(t) \to P_Q(t)\}$, $\Sigma = \{P_Q\}$, and the class of all datasets.*

To address this limitation, we focus from now onwards on a *full materialisation* variant of Algorithm 1, in which the signature parameter is fixed to the set Σ_Q of all IDB predicates in Q—that is, where the algorithm keeps in memory a

complete materialisation of the query's program for the relevant time points. Computing and incrementally maintaining a full materialisation is a common reasoning approach adopted by many rule-based systems (Motik et al. 2015; 2014; Leone et al. 2006; Baget et al. 2015). In this setting, we will be able to ensure existence of a valid window size for any fp-query, and to show that a (maybe larger than needed) valid window size can be obtained syntactically by inspecting the rules in the query one at a time.

Towards this goal, we first analyse the aforementioned stream reasoning algorithm parametrised with query Q, window size w, and signature Σ_Q, and show that only the rules in Q with radius at most w can contribute to the output.

Theorem 8. *Consider Algorithm 1 parametrised with Q, w and Σ_Q. On input $\langle B, S\rangle$, the set of P_Q-facts streamed out in the first n iterations coincides with $Q^w(B \cup S)\restriction_{[0,n-1]}$.*

Proof sketch. We show by induction on τ that the set of temporal facts stored in M right after executing Line 4 of the algorithm's main loop coincides with the temporal facts entailed by $\Pi_Q^w \cup B \cup S$ and holding at any $\tau' \in [\tau - w, \tau]$, which directly implies the statement of the theorem. On the one hand, we show that any derivation from $\Pi_Q \cup M$ of a fact α holding at τ can involve only rules from Π_Q^w; in particular, any derivation involving a rule in Π_Q with radius exceeding w would require some fact holding at a time point prior to $\tau - w$, where all such facts were removed from M in previous iterations of the algorithm. On the other hand, we show that all facts holding at τ entailed by $\Pi_Q^w \cup B \cup S$ admit a derivation involving only facts holding in $[\tau - w, \tau]$; by the induction hypothesis, all such facts are in M when Line 4 of the algorithm is executed in the loop's iteration for τ. □

Theorem 8 immediately yields a characterisation of window size validity in terms of query containment.

Corollary 9. *A window size w is valid for an fp-query Q, the signature Σ_Q, and a class $\mathcal{D}$ of datasets iff $Q \sqsubseteq_{\mathcal{D}} Q^w$.*

Since Q and Q^w coincide unless the radius of Q exceeds w, we can conclude that the radius of Q is always a valid window size.

Corollary 10. *Let Q be an fp-query. Then, the radius of Q is a valid window size for Q, Σ_Q, and any class of datasets $\mathcal{D}$.*

5 The Window Validity Problem

The full materialisation of a query for any given time point may be rather large. Having a valid window size that is as small as possible is thus important for Algorithm 1 to be practically feasible, where even a small improvement on the window size can lead to a significant reduction in the number of facts stored in memory and used for query evaluation.

In particular, the radius of the query yields a valid window size that may be larger than strictly necessary. For instance, our running example query has a radius of two, which would require Algorithm 1 to keep a full materialisation for three consecutive time points; however, the query admits a valid window size of just one since the policy implemented by Rule (2) is subsumed by the other IDP in the example.

We next introduce the *window validity problem*, which is to check whether a given window size is valid for a given query. Due to Corollary 10, computing a valid window of minimal size is clearly feasible using a logarithmic number of calls in the radius of the query to an oracle for this problem. Furthermore, such minimal window can be computed "offline" before Algorithm 1 is applied to any input data.

Definition 11. *Let $\mathcal{Q}$ and $\mathcal{D}$ be classes of fp-queries and datasets, respectively. Then, $\textsc{Window}_{\mathcal{D}}^{\mathcal{Q}}$ is the problem of deciding, given $Q \in \mathcal{Q}$ and $w \geq 0$ as input, whether w is a valid window size for Q, Σ_Q, and $\mathcal{D}$.*

Corollary 9 provides a straightforward reduction from our problem to query containment. We next show that a reduction in the other direction also exists, which implies that our problem has exactly the same complexity as query containment for all classes of queries we consider.

Theorem 12. $\textsc{Window}_{\mathcal{D}}^{\mathcal{Q}}$ *and* $\textsc{Cont}_{\mathcal{D}}^{\mathcal{Q}}$ *are interreducible in* LogSpace *for each $\mathcal{Q} \in \{\textsc{fp}, \textsc{og}, \textsc{nr}, \textsc{ognr}\}$ and each class $\mathcal{D}$ of datasets.*

Proof sketch. Consider queries Q_1 and Q_2 in $\mathcal{Q}$, and assume w.l.o.g. that they do not share any IDBs other than the output predicate. In the case $\mathcal{Q} \in \{\textsc{og}, \textsc{ognr}\}$ we also assume w.l.o.g. that Q_1 and Q_2 are object-free. The key idea in reducing containment to window validity is to merge Q_1 and Q_2 into a single query Q such that

1. both Q_1 and Q_2 may contribute to the answers of Q, and
2. only Q_2 may contribute to the answers of Q^w if w is chosen as the maximum radius amongst Q_1 and Q_2.

It follows that such w is a valid window for Q, Σ_Q and $\mathcal{D}$ iff $Q_1 \sqsubseteq_{\mathcal{D}} Q_2$. To construct Π_Q, we first rename the output predicate in Π_{Q_1} and Π_{Q_2} to fresh P_{Q_1} and P_{Q_2}, then union the resulting programs, and finally include the following extra rules (7) and (8), where A and B are fresh unary temporal EDB predicates, w is as before, and $\mathbf{s} = \langle \mathbf{x}, t\rangle$ if Q_1 and Q_2 are temporal and $\mathbf{s} = \mathbf{x}$ otherwise.

$$A(t - w - 1) \wedge B(t) \wedge P_{Q_1}(\mathbf{s}) \to P_Q(\mathbf{x}, t) \quad (7)$$

$$B(t) \wedge P_{Q_2}(\mathbf{s}) \to P_Q(\mathbf{x}, t) \quad (8)$$

Note that both Q_1 and Q_2 contribute to the answers to Q if the input stream contains facts for A and B in all time points. Furthermore, Rule (7) has radius $w+1$; thus, it is not contained in Q^w and cannot contribute to its answers. □

Since the language of fp-queries is an extension of Datalog, it follows from Theorem 12 and standard results on Datalog query containment that window validity is undecidable (Shmueli 1993) in general and coNExp-hard for non-recursive queries (Benedikt and Gottlob 2010). Furthermore, the results on containment for non-recursive temporal queries in our prior work (Ronca et al. 2018) show that the aforementioned coNExp lower bound is tight.

Corollary 13. *Let $\mathcal{D}$ contain all finite datasets. Then,*

- $\textsc{Window}_{\mathcal{D}}^{\mathcal{Q}}$ *is undecidable for any $\mathcal{Q}$ containing all Datalog queries, and*
- $\textsc{Window}_{\mathcal{D}}^{\textsc{nr}}$ *is* coNExp-complete.

In the following section we show how to circumvent the undecidability result in Corollary 13 while preserving the full power of forward-propagating queries and, in particular, their ability to express temporal recursion.

6 Window Validity for Fixed Object Domain

We consider the situation where the set of objects relevant to the application domain can be fixed in advance, in the sense that any input set of background facts and any input stream refer only to those objects. This is a reasonable assumption in many applications of stream reasoning. For instance, when analysing temperature readings of wind turbines, one may assume that the set of turbines generating the data remains unchanged; furthermore, for the purpose of analysis we can often also assume that temperature readings themselves can be discretised into relevant levels according to suitable thresholds. In our running example, the set of nodes (pieces of data-generating computer equipment) present in the network is likely to change only rather rarely.

For the remainder of this section, let us fix a finite set O of objects and let us denote with $\mathcal{O}$ the class of datasets mentioning objects from O only. Note that $\mathcal{O}$ is a valid class of datasets since it trivially satisfies the relevant assumptions in Section 2; thus, problems $\textsc{Window}^{\mathcal{Q}}_{\mathcal{O}}$ and $\textsc{Cont}^{\mathcal{Q}}_{\mathcal{O}}$ are well-defined and, by Theorem 12, they are also interreducible for any class of queries $\mathcal{Q}$ mentioned in this paper.

In what follows, we show that $\textsc{Window}^{\mathcal{Q}}_{\mathcal{O}}$ is decidable and establish tight complexity bounds.

6.1 Decidability and Upper Bounds

Fixing O allows us to transform any input Q to $\textsc{Window}^{\mathcal{Q}}_{\mathcal{O}}$ for $\mathcal{Q} \subseteq \textsc{fp}$ into an object-ground query by grounding the object variables in Q to constants in O; this yields an exponential reduction from $\textsc{Window}^{\mathcal{Q}}_{\mathcal{O}}$ to $\textsc{Window}^{\textsc{og}}$. Thus, our first step will be to decide window validity for object-ground queries, and for this we provide a decision procedure for the corresponding query containment problem.

Let us consider fixed, but arbitrary, object-ground (temporal) queries Q_1 and Q_2 sharing an output predicate G. For simplicity, and without loss of generality, we assume that Q_1 and Q_2 contain no object terms and hence all predicates in the queries are either nullary or unary and temporal.

We first show that there exists a number b of exponential size in $|Q_1| + |Q_2|$ such that $Q_1 \not\sqsubseteq Q_2$ holds if and only if $G(\tau) \in Q_1(D)$ and $G(\tau) \notin Q_2(D)$ for some $\tau \in [0, b]$ and some dataset D over time points in $[0, b]$. We do so by constructing deterministic automata $\mathcal{A}_1$ and $\mathcal{A}_2$ for Q_1 and Q_2, respectively, and deriving b from well-known bounds for the size of counter-examples to automata containment.

Lemma 14. *For each $i \in \{1, 2\}$, let ρ_i and p_i be the radius and the size of the signature of Q_i, respectively. Let $b_i = 1 + 2^{p_i \cdot (\rho_i + 2)}$, and let $b = b_1 \cdot b_2$.*

If $Q_1 \not\sqsubseteq Q_2$, then there exists a time point $\tau \in [0, b]$ and a dataset D over time points in $[0, b]$ such that $G(\tau) \in Q_1(D)$ and $G(\tau) \notin Q_2(D)$.

Proof sketch. We start with the observation that, given Q_i and a dataset D, we can check whether the output predicate is derived at any time point from $\Pi_{Q_i} \cup D$ using our generic stream reasoning algorithm. That is, we can start by loading the rigid facts in D and subsequently reading the temporal facts one time point at a time while maintaining entailments over a window of size ρ_i until the output predicate is derived or D does not contain any further time points.

The correctness of this algorithm relies on the fact that Q_i is forward-propagating and hence ρ_i is a valid window. Based on this, we can construct a deterministic finite automaton $\mathcal{A}_i$ that captures Q_i in the following sense: on the one hand, each dataset D corresponds to a word over the alphabet of the automaton, where the first symbol is the set of rigid facts in D and the remaining symbols encode the temporal facts in D one time point at a time on the other hand, each state corresponds to a snapshot of the facts stored in memory by the algorithm, and a state is final if it corresponds to a snapshot in which the output predicate has just been derived. Automaton $\mathcal{A}_i$ is defined as follows:

- A state is either the initial state s^i_{init}, or a $(\rho_i + 2)$-tuple where the first component is a subset of the rigid EDB predicates in Q_i, and the other components are subsets of the temporal (EDB and IDB) predicates in Q_i. A state is final if its last component contains the output predicate G.
- Each alphabet symbol is a set Σ of EDB predicates occurring in Q_i such that Σ does not contain temporal and rigid predicates simultaneously.
- The transition function δ_i consists of
  - transitions $s^i_{init}, \Sigma \mapsto \langle \Sigma, \emptyset, \dots, \emptyset \rangle$ such that Σ consists of rigid predicates;
  - transitions $\langle B, M_0, \dots, M_{\rho_i} \rangle, \Sigma \mapsto \langle B, M'_0, \dots, M'_{\rho_i} \rangle$ such that: Σ consists of temporal predicates; $M'_j = M_{j+1}$ for each $0 \leq j < \rho_i$; and M'_{ρ_i} consists of each predicate P satisfying $\Pi_{Q_i} \cup B \cup H \cup U \models P(\rho_i)$ for H the set of all facts $R(j)$ with $R \in M_j$ and $0 \leq j < \rho_i$, and U the set of all facts $R(\rho_i)$ with $R \in \Sigma$.

The fact that each automaton $\mathcal{A}_i$ captures Q_i in the sense described before ensures that the following properties immediately hold:

1. If $Q_1 \not\sqsubseteq Q_2$, then there exists a word that is accepted by $\mathcal{A}_1$ and not by $\mathcal{A}_2$.
2. For each word of length n accepted by $\mathcal{A}_1$ and not by $\mathcal{A}_2$, there exists a dataset D over time points in $[0, n-2]$ such that $G(n-2) \in Q_1(D)$ and $G(n-2) \notin Q_2(D)$.

We finally argue that these properties imply the statement of the lemma. If $Q_1 \not\sqsubseteq Q_2$ then, by Property 1, there is a word accepted by $\mathcal{A}_1$ and not by $\mathcal{A}_2$. By standard automata results, it follows that there is also a word accepted by $\mathcal{A}_1$ and not by $\mathcal{A}_2$ having length n bounded by the product of the number of states in $\mathcal{A}_1$ and $\mathcal{A}_2$, where the number of states in $\mathcal{A}_i$ is bounded by b_i. By Property 2, there exists a dataset D over time points in $[0, n-2]$ such that $G(n-2) \in Q_1(D)$ and $G(n-2) \notin Q_2(D)$, where n is bounded by b. □

Lemma 14 immediately suggests a non-deterministic algorithm for deciding $Q_1 \not\sqsubseteq Q_2$, in which a witness dataset is constructed and checked in each branch. In order to ensure that the space used in each branch stays polynomial, we exploit our observation in the beginning of the proof of Lemma 14. A witness D is guessed one time point at a time

until reaching the bound b, and $Q_1(D) \not\subseteq Q_2(D)$ is verified incrementally after each guess while keeping in memory just a window of size bounded by the radiuses of Q_1 and Q_2.

Lemma 15. CONT$^{\text{OG}}$ *is in* PSPACE.

Proof. We decide $Q_1 \not\sqsubseteq Q_2$ using the following algorithm, where ρ is the maximum radius of Q_1 and Q_2.

1. Guess a set D_r of rigid facts and set M_1 and M_2 to D_r.
2. For each value of τ from 0 to b as in Lemma 14.
   a. Guess $D\restriction_\tau$.
   b. Set each M_i to $M_i \cup D\restriction_\tau$.
   c. Add to each M_i facts α at τ s.t. $\Pi_{Q_i} \cup M_i \models \alpha$.
   d. If there is a G-fact in $M_1\restriction_\tau$ and not in $M_2\restriction_\tau$, accept.
   e. Remove from each M_i all facts in $M_i\restriction_{\tau-\rho}$.
3. Reject.

The algorithm correctly computes the answers over the guessed facts, since it mimics Algorithm 1 and ρ is a valid window for both queries. By Lemma 14, the algorithm finds a witness dataset for non-containment whenever one exists. Furthermore, the algorithm runs in polynomial space since the size of each M_i is polynomial, and a polynomially-sized counter suffices for checking the halting condition. □

Lemma 15 yields a PSPACE upper bound to WINDOW$^{\text{OG}}$. In turn, it also provides an EXPSPACE upper bound to WINDOW$^{\text{FP}}_{\mathcal{O}}$, which is obtained by first applying to the input query Q a grounding step where object variables from Π_Q are replaced with constants from the object domain. Furthermore, this grounding process is polynomial in the number of domain objects and exponential in the maximum number of object variables in a rule from Π_Q; thus, the PSPACE upper bound in Lemma 15 extends to any class of queries where the maximum number of object variables in a rule can be bounded by a constant (which equals zero for OG).

Theorem 16. *The following upper bounds hold:*

- WINDOW$^{\text{FP}}_{\mathcal{O}}$ *is in* EXPSPACE*; and*
- WINDOW$^{\mathcal{Q}}_{\mathcal{O}}$ *it is in* PSPACE *for any class $\mathcal{Q}$ of fp-queries where the maximum number of object variables in any rule of any $Q \in \mathcal{Q}$ is bounded by a constant.*

By exploiting results from our prior work (Ronca et al. 2018), we can show that WINDOW$^{\text{OGNR}}$ reduces to query containment over non-recursive plain propositional Datalog. The latter can be decided in coNP by universally guessing a set of propositional symbols D and then checking (in polynomial time) that $Q_2(D)$ holds whenever $Q_1(D)$ does, which yields a coNP bound for WINDOW$^{\text{OGNR}}$. In turn, this bound yields a coNEXP upper bound for WINDOW$^{\text{NR}}_{\mathcal{O}}$ by means of an exponential grounding step of the object variables. Furthermore, such grounding is polynomial for any class $\mathcal{Q} \subseteq$ NR where the maximum number of object variables in any rule is bounded by a constant; hence, the coNP upper bound for OGNR seamlessly extends to any such class.

Theorem 17. *The following upper bounds hold:*

- WINDOW$^{\text{NR}}_{\mathcal{O}}$ *is in* coNEXP*; and*
- WINDOW$^{\mathcal{Q}}_{\mathcal{O}}$ *is in* coNP *for any class $\mathcal{Q} \subseteq$ NR where the maximum number of object variables in any rule of any $Q \in \mathcal{Q}$ is bounded by a constant.*

6.2 Lower Bounds

We next show that all the upper bounds established in Section 6.1 are tight. We start by providing a matching PSPACE lower bound to WINDOW$^{\text{OG}}$.

Theorem 18. WINDOW$^{\text{OG}}$ *is* PSPACE-hard.

Proof sketch. We show hardness for CONT$^{\text{OG}}$, which implies the theorem's statement by Theorem 12. The proof is by reduction from the containment problem for regular expressions. Let R_1 and R_2 be regular expressions over a common finite alphabet Σ. We construct object-free queries Q_1 and Q_2 with unary output temporal predicate G such that $R_1 \sqsubseteq R_2$ if and only if $Q_1 \sqsubseteq Q_2$.

Each Q_i is defined such that it captures R_i as described next. We encode words in Σ^* using facts over unary temporal EDB predicates F and A_σ for each alphabet symbol $\sigma \in \Sigma$. Intuitively, a fact $F(\tau)$ indicates that τ is the first position of the word, whereas a fact $A_\sigma(\tau')$ with $\tau' \geq \tau$ means that σ is the symbol in position $\tau' - \tau$. Queries Q_i are constructed from R_i such that the following property $(\star)$ holds for each dataset D over the aforementioned EDB predicates and each time point τ:

> $(\star)$: $G(\tau) \in Q_i(D)$ if and only if there exists a word $\sigma_1 \dots \sigma_n$ in the language of R_i such that D contains facts $F(\tau - n), A_{\sigma_1}(\tau - n), A_{\sigma_2}(\tau - n + 1), \dots, A_{\sigma_n}(\tau - 1)$.

Property $(\star)$ implies the statement of the theorem. On the one hand, if $Q_1 \not\sqsubseteq Q_2$, then $G(\tau) \in Q_1(D)$ and $G(\tau) \notin Q_2(D)$ for some τ and D; by $(\star)$, the former implies existence of a word s in $\mathcal{L}(R_1)$ such that D contains the relevant facts, whereas the latter together with the aforementioned property of D implies that $s \notin \mathcal{L}(R_2)$. On the other hand, $R_1 \not\sqsubseteq R_2$ implies that there exists $s = \sigma_1 \dots \sigma_n$ with $s \in \mathcal{L}(R_1)$ and $s \notin \mathcal{L}(R_2)$; let D_s be the dataset consisting of facts

$$F(0), A_{\sigma_1}(0), A_{\sigma_2}(1), \dots, A_{\sigma_n}(n-1)$$

By $(\star)$, we then have $G(n) \in Q_1(D_s)$ and $G(n) \notin Q_2(D_s)$, and hence $Q_1 \not\sqsubseteq Q_2$.

We now define $Q_i = \langle G, \Pi_{R_i} \rangle$, where Π_{R_i} is defined inductively from R_i as described next; note that, for Π a program, we denote with Π' (resp., Π'') the program obtained from Π by renaming each predicate P not in $\{A_\sigma \mid \sigma \in \Sigma\}$ to a globally fresh predicate P' (P'') of the same arity.

1. $R_i = \emptyset$. Then, Π_{R_i} is the empty program.
2. $R_i = \sigma$ for $\sigma \in \Sigma$. Then, Π_{R_i} consists of rule
$$F(t) \wedge A_\sigma(t) \to G(t+1).$$
3. $R_i = \varepsilon$. Then, Π_{R_i} consists of rule
$$F(t) \to G(t).$$
4. $R_i = S \cup T$. Then, Π_{R_i} extends $\Pi'_S \cup \Pi''_T$ with rules
$$F(t) \to F'(t) \qquad F(t) \to F''(t)$$
$$G'(t) \to G(t) \qquad G''(t) \to G(t).$$
5. $R_i = S \circ T$. Then, Π_{R_i} extends $\Pi'_S \cup \Pi''_T$ with rules
$$F(t) \to F'(t), \quad G'(t) \to F''(t), \quad G''(t) \to G(t).$$

6. $R_i = S^+$. Then, Π_{R_i} extends Π'_S with rules
$$F(t) \to F'(t), \quad G'(t) \to F'(t), \quad G'(t) \to G(t).$$

It can be checked using a simple induction that the construction ensures that $(\star)$ holds. □

Theorem 18 implies PSPACE-hardness of $\text{WINDOW}^{\mathcal{Q}}_{\mathcal{O}}$ for any class $\mathcal{Q}$ of fp-queries where the maximum number of object variables is bounded by a constant.

We next show a matching EXPSPACE lower bound to the complexity of $\text{WINDOW}^{\text{FP}}_{\mathcal{O}}$. To this end, we upgrade the reduction in Theorem 18 to a reduction from the containment problem of succinct regular expressions—regular expression extended with an exponentiation operation R^k where k is coded in binary (Sipser 2006).

Theorem 19. $\text{WINDOW}^{\text{FP}}_{\mathcal{O}}$ *is* EXPSPACE-hard.

Proof sketch. We show hardness of the corresponding query containment problem, which implies the statement by Theorem 12. Let R_1 and R_2 be succinct regular expressions over the same vocabulary Σ. We construct fp-queries Q_1 and Q_2 over the same unary temporal output predicate G such that $R_1 \sqsubseteq R_2$ if and only if $Q_1 \sqsubseteq Q_2$.

As in the proof of Theorem 18, we construct Q_i such that it captures R_i. We encode words as before using unary temporal EDB predicates F and A_σ for each $\sigma \in \Sigma$. Also as before, we construct Q_i from R_i such that property $(\star)$ holds where D in the formulation of $(\star)$ is over objects in O.

We now define $Q_i = \langle G, \Pi_{\text{succ}} \cup \Pi_{R_i} \rangle$, where Π_{R_i} will be defined inductively over the structure of R_i, and Π_{succ} is a Datalog program that defines in the standard way (Dantsin et al. 2001) rigid IDB successor predicates $succ^m$ of arity $2m$ relating m-strings over objects $\bar{0}$ and $\bar{1}$ for each exponent k occurring in R_i with $m = \lceil \log_2 k \rceil$. Now we proceed with the inductive definition of Π_{R_i}, which is analogous to that in the proof of Theorem 18 with the following additional case, and the minor modification that successor predicates are never renamed apart:

7. $R_i = S^k$ for some succinct regular expression S and $k \geq 2$. Then, Π_{R_i} is constructed from Π_S as follows. First, we replace each n-ary atom $P(\mathbf{p}, s)$, for $\mathbf{p}$ a vector of object terms and s a temporal term, with $P'(\mathbf{p}, \mathbf{x}, s)$ for P' a fresh predicate (unique to P) of arity $n+m$ with $m = \lceil \log_2 k \rceil$, and $\mathbf{x}$ a fixed m-vector of fresh object variables. Second, we extend the resulting program with the following rules, where $\mathbf{a}$ is the encoding of $k-1$ as a binary string over $\bar{0}$ and $\bar{1}$:
$$F(t) \to F'(\bar{\mathbf{0}}, t)$$
$$G'(\mathbf{a}, t) \to G(t)$$
$$G'(\mathbf{x}, t) \wedge succ^m(\mathbf{x}, \mathbf{y}) \to F'(\mathbf{y}, t)$$

We can show inductively that $(\star)$ holds. □

To conclude, we turn our attention to the case of non-recursive queries. A matching coNP lower bound to the complexity of $\text{WINDOW}^{\text{OGNR}}$ is obtained by a simple reduction from 3-SAT to the complement of our problem. A matching coNEXP lower bound for $\text{WINDOW}^{\text{NR}}_{\mathcal{O}}$ follows by a simple adaptation of the hardness proofs in (Benedikt and Gottlob 2010) for containment in non-recursive Datalog.

Theorem 20. $\text{WINDOW}^{\text{OGNR}}$ *is* coNP-hard. *Furthermore,* $\text{WINDOW}^{\text{NR}}_{\mathcal{O}}$ *is* coNEXP-hard *if O has at least two objects.*

7 A Sufficient Condition for Window Validity

The assumption that the object domain can be fixed in advance may not be reasonable in some applications. For instance, it may be the case that sensor values cannot be naturally discretised into suitable levels according to a threshold, or that new sensors are continuously activated on-the-fly.

As already established, dropping the fixed domain assumption leads to undecidability of window validity for (recursive) fp-queries. In this section, we propose a sufficient condition for the validity of a window that can be checked in exponential time without additional assumptions, and which leads to smaller window sizes compared to the radius of the query. Our condition relies on the notion of *uniform containment* of two programs Π_1 and Π_2 (Sagiv 1988), which is sufficient to ensure containment of any queries Q_1 and Q_2 based on Π_1 and Π_2, respectively.

Definition 21. *An extended dataset E is a (possibly infinite) set of (not necessarily EDB) facts. Program Π_1 is* uniformly contained *in program Π_2, written $\Pi_1 \sqsubseteq^{\text{u}} \Pi_2$, if and only if, for each extended dataset E and each fact α, it holds that $\Pi_1 \cup E \models \alpha$ implies $\Pi_2 \cup E \models \alpha$.*

A window size w is uniformly valid *for an fp-query Q if and only if $\Pi_Q \sqsubseteq^{\text{u}} \Pi_{Q^w}$.*

It is straightforward to check that, given any queries Q_1 and Q_2, it holds that $\Pi_{Q_1} \sqsubseteq^{\text{u}} \Pi_{Q_2}$ implies $Q_1 \sqsubseteq Q_2$. Hence, we can establish that uniform validity is a sufficient condition for window validity, which is more precise than the syntactic condition given by the radius.

Proposition 22. *Let Q be an fp-query with radius ρ, and let w be a non-negative integer. If w is a uniformly valid window size for Q, then w is also a valid window size for Σ_Q and any class $\mathcal{D}$ of datasets. Furthermore, if w is the smallest uniformly valid window size for Q, then $w \leq \rho$.*

Example 23. *Consider the query Q where Π_Q consists of the following rules and A is the only EDB predicate:*
$$A(t) \to P_Q(t) \qquad A(t-1) \wedge A(t) \to P_Q(t)$$
Query Q has radius one. We can see that $w = 0$ is a (uniform) window. Intuitively, this is because the first rule entails the second; thus, Π_Q and Π_{Q^w} are logically (and hence also uniformly) equivalent.

It is well-known that uniform program containment amounts to checking fact entailment (Sagiv 1988). On the one hand, to check $\Pi_1 \sqsubseteq^{\text{u}} \Pi_2$, it suffices to show that Π_2 entails each rule r in Π_1, which can in turn be checked by first "freezing" r into an extended dataset E for the body and a fact α for the head and then verifying whether $\Pi_2 \cup E \models \alpha$. On the other hand, to check whether $\Pi \cup E \models \alpha$, it suffices to check uniform containment of a single rule r in Π, where r is obtained from E and α by replacing each constant with a fresh variable in the obvious way.

Theorem 24. *Let $\mathcal{Q} \in \{\text{FP}, \text{NR}, \text{OG}, \text{OGNR}\}$ and let $\mathcal{P}$ be the class of programs that occur in queries from $\mathcal{Q}$. Then, uniform window validity over queries in $\mathcal{Q}$ and fact entailment over programs in $\mathcal{P}$ are inter-reducible in* LOGSPACE.

The following complexity bounds for uniform window validity immediately follow from complexity results for fact entailment.

Corollary 25. *Uniform window validity over a class $\mathcal{Q}$ of queries is*
- EXP-complete *if* $\mathcal{Q} = \text{FP}$*;*
- PSPACE-complete *if* $\mathcal{Q} = \text{NR}$*;*
- *in* P *if $\mathcal{Q}$ is any subclass of* FP *where the maximum number of object variables in any rule of any $Q \in \mathcal{Q}$ is bounded by a constant; and*
- *in* AC^0 *if $\mathcal{Q}$ is any subclass of* NR *where the maximum number of object variables in any rule of any $Q \in \mathcal{Q}$ is bounded by a constant.*

We see uniform validity as a reasonable compromise in practice. On the one hand, it may yield smaller window sizes than the radius of the query, thus reducing the amount of information that a stream reasoning algorithm needs to retain in memory; on the other hand, it can be checked while relying solely on query processing infrastructure, and hence without the need for specialised algorithms.

8 Related Work

The formal underpinnings of stream query processing in databases were established in (Babcock et al. 2002; Arasu, Babu, and Widom 2006). Arasu, Babu, and Widom (2006) proposed CQL as an extension of SQL with a window construct, which specifies the input data relevant for query processing at any point in time. CQL has become since then the core of many other stream query languages, including languages for the Semantic Web (Barbieri et al. 2009; 2010; Le-Phuoc et al. 2011; 2013; Dell'Aglio et al. 2015).

In the context of stream reasoning, Zaniolo (2012) proposed Streamlog: a language which extends temporal Datalog with non-monotonic negation while at the same time restricting the syntax so that only facts over time points mentioned in the data can be derived. LARS (Beck et al. 2015; Beck, Dao-Tran, and Eiter 2015; 2016) is a temporal rule-based stream reasoning language featuring built-in window constructs and negation interpreted according to the stable model semantics. In contrast to temporal Datalog, the semantics of LARS assumes that the number of time points in a model is a part of the input to query evaluation, and hence is restricted to be finite. Stream reasoning has also been considered in ontology-based data access (Calbimonte, Corcho, and Gray 2010; Özçep, Möller, and Neuenstadt 2014) as well as in the context of complex event processing (Anicic et al. 2011; Dao-Tran and Le-Phuoc 2015).

There have been several proposals of Datalog extensions for reasoning over static temporal data. The language we consider is a notational variant of $\text{Datalog}_{1\text{S}}$ (Chomicki and Imieliński 1988; 1989; Chomicki 1990). Templog is an extension of Datalog with modal temporal operators (Abadi and Manna 1989); DatalogMTL is an extension with metric temporal logic (Brandt et al. 2017); and the language proposed by Toman and Chomicki (1998) extends Datalog with integer periodicity constraints.

Our language of fp-queries is related to past temporal logic, where formulae are restricted to refer to past time points only (Manna and Pnueli 1992; Chomicki 1995). Chomicki (1995) presents an incremental update algorithm for checking dynamic integrity constraints expressed in past temporal logic; similarly to our stream reasoning algorithm, Chomicki's update algorithm exploits the idea that the length of the stored history throughout a sequence of updates can be bounded to a value depending only on the query.

The window validity problem was introduced in our prior work (Ronca et al. 2018) based on a generic stream reasoning algorithm that only keeps EDB facts in memory. We established undecidability for unrestricted queries, and provided tight complexity bounds for the non-recursive case. Our current paper extends (Ronca et al. 2018) by generalising window validity to the case where the underpinning stream reasoning algorithm can also keep IDB facts in memory; furthermore, we show decidability and tight complexity bounds for recursive queries under the (rather mild) assumption that the object domain can be fixed in advance. The window validity problem is related to the *forgetting* problem in logic programming (Wang, Sattar, and Su 2005; Eiter and Wang 2008), where the goal is to eliminate predicates while preserving certain logical consequences.

9 Conclusion and Future Work

We have studied the window validity problem in stream reasoning and its computational properties for temporal Datalog. We showed that window validity is undecidable; however, decidability can be regained by making mild assumptions on the input data.

We see many avenues for future work. First, it would be interesting to consider window validity for extensions of temporal Datalog (e.g., with comparison atoms or stratified negation) as well as for DatalogMTL. Second, we have assumed throughout the paper that all numbers in input queries and data are coded in unary; it would be interesting to revisit our technical results for the case where binary encoding is assumed instead. Finally, our decidability results do not immediately yield implementable algorithms; we are planning to develop and implement practical window validity checking algorithms under the fixed object domain assumption.

Acknowledgments

This research was supported by the SIRIUS Centre for Scalable Data Access in the Oil and Gas Domain and the EPSRC projects DBOnto, MaSI^3, and ED^3.

References

Abadi, M., and Manna, Z. 1989. Temporal logic programming. *J. Symb. Comput.* 8(3).

Anicic, D.; Fodor, P.; Rudolph, S.; and Stojanovic, N. 2011. EP-SPARQL: a unified language for event processing and stream reasoning. In *WWW*.

Arasu, A.; Babu, S.; and Widom, J. 2006. The CQL continuous query language: Semantic foundations and query execution. *VLDB J.* 15(2).

Babcock, B.; Babu, S.; Datar, M.; Motwani, R.; and Widom, J. 2002. Models and issues in data stream systems. In *PODS*.

Baget, J.; Leclère, M.; Mugnier, M.; Rocher, S.; and Sipieter, C. 2015. Graal: A toolkit for query answering with existential rules. In *RuleML*.

Barbieri, D. F.; Braga, D.; Ceri, S.; Della Valle, E.; and Grossniklaus, M. 2009. C-SPARQL: SPARQL for continuous querying. In *WWW*.

Barbieri, D. F.; Braga, D.; Ceri, S.; Della Valle, E.; and Grossniklaus, M. 2010. Incremental reasoning on streams and rich background knowledge. In *ESWC*.

Baudinet, M.; Chomicki, J.; and Wolper, P. 1993. Temporal deductive databases. In Tansel, A. U.; Clifford, J.; Gadia, S.; Jajodia, S.; Segev, A.; and Snodgrass, R., eds., *Temporal Databases*. Benjamin Cummings.

Beck, H.; Dao-Tran, M.; Eiter, T.; and Fink, M. 2015. LARS: A logic-based framework for analyzing reasoning over streams. In *AAAI*.

Beck, H.; Dao-Tran, M.; and Eiter, T. 2015. Answer update for rule-based stream reasoning. In *IJCAI*.

Beck, H.; Dao-Tran, M.; and Eiter, T. 2016. Equivalent stream reasoning programs. In *IJCAI*.

Benedikt, M., and Gottlob, G. 2010. The impact of virtual views on containment. *PVLDB* 3(1-2).

Brandt, S.; Kalayci, E. G.; Kontchakov, R.; Ryzhikov, V.; Xiao, G.; and Zakharyaschev, M. 2017. Ontology-based data access with a Horn fragment of metric temporal logic. In *AAAI*.

Calbimonte, J.; Corcho, O.; and Gray, A. J. 2010. Enabling ontology-based access to streaming data sources. In *ISWC*.

Chomicki, J., and Imieliński, T. 1988. Temporal deductive databases and infinite objects. In *PODS*.

Chomicki, J., and Imieliński, T. 1989. Relational specifications of infinite query answers. In *SIGMOD*.

Chomicki, J. 1990. Polynomial time query processing in temporal deductive databases. In *PODS*.

Chomicki, J. 1995. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. Database Syst.* 20(2).

Cosad, C.; Dufrene, K.; Heidenreich, K.; McMillon, M.; Jermieson, A.; O'Keefe, M.; and Simpson, L. 2009. Wellsite support from afar. *Oilfield Review* 21(2).

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Comput. Surv.* 33(3).

Dao-Tran, M., and Le-Phuoc, D. 2015. Towards enriching CQELS with complex event processing and path navigation. In *HiDeSt@KI*.

Dao-Tran, M.; Beck, H.; and Eiter, T. 2015. Towards comparing RDF stream processing semantics. In *HiDeSt@KI*.

Dell'Aglio, D.; Calbimonte, J.; Della Valle, E.; and Corcho, O. 2015. Towards a unified language for RDF stream query processing. In *ESWC (Satellite Events)*.

Eiter, T., and Wang, K. 2008. Semantic forgetting in answer set programming. *Artif. Intell.* 172(14).

Le-Phuoc, D.; Dao-Tran, M.; Parreira, J. X.; and Hauswirth, M. 2011. A native and adaptive approach for unified processing of linked streams and linked data. In *ISWC*.

Le-Phuoc, D.; Quoc, H. N. M.; Le Van, C.; and Hauswirth, M. 2013. Elastic and scalable processing of linked stream data in the cloud. In *ISWC*.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.* 7(3).

Manna, Z., and Pnueli, A. 1992. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer.

Motik, B.; Nenov, Y.; Piro, R.; Horrocks, I.; and Olteanu, D. 2014. Parallel materialisation of Datalog programs in centralised, main-memory RDF systems. In *AAAI*.

Motik, B.; Nenov, Y.; Piro, R.; and Horrocks, I. 2015. Combining rewriting and incremental materialisation maintenance for Datalog programs with equality. In *IJCAI*.

Münz, G., and Carle, G. 2007. Real-time analysis of flow data for network attack detection. In *IM*.

Nuti, G.; Mirghaemi, M.; Treleaven, P.; and Yingsaeree, C. 2011. Algorithmic trading. *IEEE Computer* 44(11).

Özçep, Ö. L.; Möller, R.; and Neuenstadt, C. 2014. A stream-temporal query language for ontology based data access. In *KI*.

Ronca, A.; Kaminski, M.; Cuenca Grau, B.; Motik, B.; and Horrocks, I. 2018. Stream reasoning in temporal Datalog. In *AAAI*.

Sagiv, Y. 1988. Optimizing datalog programs. In Minker, J., ed., *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann.

Shmueli, O. 1993. Equivalence of datalog queries is undecidable. *J. Log. Program.* 15(3).

Sipser, M. 2006. *Introduction to the Theory of Computation*. Thomson Course Technology, 2nd edition.

Toman, D., and Chomicki, J. 1998. Datalog with integer periodicity constraints. *J. Log. Program.* 35(3).

Vorobyov, S. G., and Voronkov, A. 1998. Complexity of nonrecursive logic programs with complex values. In *PODS*.

Wang, K.; Sattar, A.; and Su, K. 2005. A theory of forgetting in logic programming. In *AAAI*.

Zaniolo, C. 2012. Logical foundations of continuous query languages for data streams. In *Datalog 2.0*.