February 2017

Ten years of Stream Reasoning. Now what?

Daniele DELL'AGLIO [a,1], Emanuele DELLA VALLE [b] and Frank VAN HARMELEN [c] and Abraham BERNSTEIN [a]

[a] *Department of Informatics - University of Zurich*
[b] *DEIB - Politecnico di Milano.*
[c] *Vrije Universiteit Amsterdam.*

Abstract. Stream reasoning studies the application of inference techniques to data characterised by being highly dynamic. It can find application in several settings, from Smart Cities to Industry 4.0, from Internet of Things to Social Media analytics. This year stream reasoning turns ten, and in this article we analyse its growth. In the first part, we trace the main results obtained so far, by presenting the most prominent studies. Looking at the past is useful to prepare for the future: in the second part, we present a set of open challenges and issues that stream reasoning will face in the next future.

Keywords. Semantic Web; Stream Processing; Stream Reasoning.

1. Introduction

Increasingly, applications require real-time processing of heterogeneous data streams together with large background knowledge. Consider the following examples. Which electricity-producing turbine has sensor readings similar (i.e., Pearson correlated by at least 0.75) to any turbine that subsequently had a critical failure in the past year [54]? When a sensor on a drill in a oil-rig indicates that it is about to get stuck, how long—according to historical records— can I keep drilling [53]? Where am I likely going to run into a traffic jam during my commute tonight and how long will it take, given current weather and traffic conditions [64,83,3]? Who are the current top influencers that are driving the discussion about the top emerging topics across all the social networks [50,8]?

Who should be asked to go exercising, given people's past, possibly sedentary behaviour and allergies (accessed in a privacy-preserving manner) as well as current weather conditions and pollution/allergen levels [31]?

To answer these queries a system must be able to:

[1]Corresponding Author: Daniele Dell'Aglio; E-mail: dellaglio@ifi.uzh.ch.

R1. handle *volume*: a typical oil production platform is equipped with about 400.000 sensors; Facebook, as of February, 2017, has 1.86 billion of monthly active users[2], etc.

R2. handle *velocity*: sensors on a power generation turbine can easily generate thousands of observations per minute; Instagram's users, as of February, 2017, like on average 2.92 million post per minute[3]; etc.

R3. handle *variety*: a large variety of static and streaming data sources and data management solutions exists in any domain. For instance, Milano has deployed some 600 traffic light systems equipped with inductive loops in the last 10 years: they use five different data formats, have different operational conditions, etc. Similarly, in social media, each network has its own data model and APIs.

R4. cope with *incompleteness*: sensors can run out of battery or networking links can break; in social media part of the conversation may occur outside the social network [7], or the APIs that are used to access the social stream may have got a limited sampling rate.

R5. cope with *noise*: sensors can be imperfect, faulty, or out of its ideal operational range; Text can be worded in an ironic way and a sentiment mining solution may be unable to detect it with 100% correctness.

R6. provide answers in *timely fashion*: answers should be generated within well-specified latency bounds, which depend on the application scenario: the detection of dangerous situations must occur within minutes (in near real-time).

R7. support *fine-grained information access*: the issued query may require to locate exactly a turbine, a means of public transportation, an agent in a contact centre among thousands of similar ones.

R8. integrate *complex domain models*: social media analytics may require topic models to make sense of a conversation; oil production control systems may require to model operational and control processes; traffic monitors may require rich background knowledge about topology, planned and unplanned events to improve the accuracy of the analyses.

R9. understand *what users want*: the query should let users define analytics-aware tasks such as Pearson correlation as a mean of similarity, or let specify complex concepts such as traffic jam and top influencer user.

Ten years ago, no system was able to address all those requirements simultaneously. Data Stream Management Systems (DSMSs) and Complex Event Processors (CEPs) [21] were two types of systems able to provide reactive fine-grained information access in the presence of noisy data. DSMSs transform data streams in timestamped relations (usually through a so-called window operator) and process them with well known techniques such as relational algebras [6]. CEPs, instead, look for patterns in the streams to identify when complex events occur [66]. These system where, however, unable to to access heterogenous data with complex domain models and rich background knowledge. Other approaches, based

[2]See `https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/` Accessed on February, 2017.

[3]See `https://blog.hootsuite.com/instagram-statistics/` Accessed on February, 2017.

on the Semantic Web and in particular on scalable Ontology Based Data Access (OBDA) [17], were starting to show that complex domain models can be used to offer fine-grained information access to heterogeneous and incomplete datasets. But these systems lacked the ability to provide reactive answers on data streams and to handle noise.

Table 1. The requirements Stream Reasoning aims at covering and how DSMS, CEP and OBDA cover them.

Requirement	**DSMS/CEP**	**SemWeb**	**Envisioned Stream Reasoning**
R1: Volume	✓	✓	✓
R2: Velocity	✓	✗	✓
R3: Variety	✗	✓	✓
R4: Incompleteness	✗	✓	✓
R5: Noise	✓	✗	✓
R6: Timely fashion	✓	✗	✓
R7: Fine-grained access	✓	✓	✓
R8: Complex domains	✗	✓	✓
R9: What user want	✓	✗	✓

Then, about ten years ago, a new research trend—Stream Reasoning [25]—combining the above developments emerged around the idea [90] of:

> Logical reasoning in real time on gigantic and inevitably noisy data streams in order to support the decision process of extremely large numbers of concurrent users.

Stream reasoners were envisioned as systems capable to address all the requirements above simultaneously (see Table 1).

Even if the goal can be easily stated, satisfying all above-stated requirements is challenging in multiple ways. Theoretically, it is difficult to create comprehensive data and processing models. Practically, it is non-trivial to guarantee reactiveness given the required functional requirements. Nonetheless, we have seen the emergence of various results [67,26]. Different research groups proposed (i) data models and vocabularies to capture data streams through RDF and ontologies (e.g. [47,12]), (ii) continuous query models, languages and prototypes (e.g. [10,16,60,5]), inspired by SPARQL [82] and collected under the *RDF Stream Processing* (RSP) label, (iii) extensions of the reasoning tasks over streams, as consistency check and closure (e.g. [84,61]), and (iv) applications built on top of the aforementioned results (e.g. [8,91,51]).

However, a complete answer to the stream reasoning research question is still missing and recent developments in scalable stream processing engines [100,19] and scalable OBDA [43] offer opportunities to engineer a new generation of stream reasoners.

In this article, we introduce a reference model for stream reasoning, which we use to summarize and organize the results the research has obtained so far. We then analyse the current trends and identify core challenges for the next years.

2. A reference model for stream reasoning

In the last ten years, several techniques were proposed under the stream reasoning label. Such techniques are heterogeneous in terms of input, output and use cases. To present them, we present Figure 1 as a reference model for stream reasoning.

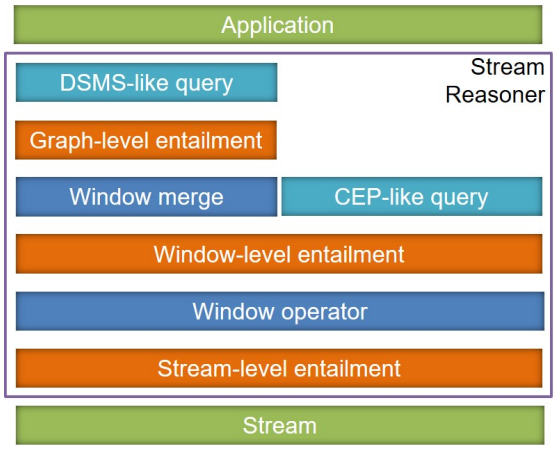

Figure 1. A model to describe stream reasoners

In the remaining of this paper, we denote a data stream as a sequence of time-annotated items ordered according to a temporal criteria. Each item brings an information unit, e.g. a sensor observation, a social network action or a stock exchange. It is possible to represent the stream items in different format. When items are represented through RDF (e.g. an RDF graph), we refer to the stream as RDF stream.

We now present a reference architecture for a generic stream processor, identified in Figure 1 by the blue blocks. *Window operators* manage the access to the stream: they create time-dependent finite views over the streams, namely windows, over which processors perform the task. Window contains a portion of the input streams, i.e. a set of timestamped data items, that represent the data needed to solve the task at the current time instant. Several types of window operators exist, initially defined in CEP and DSMS research [21].

CEP aims at verifying if given sequences of events happen in the stream. In this sense, time annotations are key, since engines use them to determine if temporal constraints defined in the event patterns are satisfied. When not specified, event patterns are evaluated over a large portion of the stream, i.e. from the moment on which the engine start to observe the stream up to the current moment. We can model this behaviour through *landmark windows*. Fixed an initial time instant, the window expands over time to capture portion of the stream that grows over time.

DSMS performs operations like aggregations and filters do not require to process time annotations after that windows have been computed. That means, while the content of the stream items is important to solve the task, time anno-

tations are not. In this sense, the *window merge* is an operation that moves from temporal data (i.e. time-annotated data items) to atemporal one (i.e. a collection of data items). *Sliding windows* are the typical window operators considered in DSMS. This operator creates a window with a fixed width in terms of time units or data items. The operator shifts (slides) the window over time, capturing the most recent part of the stream.

It is worth noting that time in CEP and DSMS is treated in a different way: in the former it has a predominant role while evaluating the event patterns, while in the latter it is used to identify the relevant portion of data over which an atemporal operation is performed.

We now move a step forward, by adding the reasoning task in the stack described above. The key question we need to answer to do it, is: *while processing a stream, when is the correct moment to take into account the inference process?* In SPARQL, the inference process is considered during the basic graph pattern evaluations over an RDF graph. The reference model described above allows different options, depicted through the orange boxes in Figure 1. It can be before or after the window (resp. stream- and window-level entailments), or still after that the window content has been merged (graph-level entailment). This decision may affect the performance, the result and the behaviour of the engine. Taking into account this framework, we can group the main works developed in the state of the art of stream reasoning, in four different groups, that we present in the rest of this section.

2.1. Graph-level entailment

The first possible possible moment to take into account the inference processes is after the window merge. That means a window captures a portion of the stream and a merge operation creates a collection from the stream item contents.

We name *graph-level entailment* the application of reasoning at this moment. As explained above, this enables the execution of typical DSMS-like queries, e.g. aggregations and filters, but make not possible to evaluate CEP-like queries, since temporal annotations are lost and it is not possible to verify if temporal constraints are satisfied. This is represented in Figure 1, where the graph-level entailment stacks on Stream Processing bu not on Event Processing.

The graph-level entailment can be viewed as a direct application of SPARQL entailment regimes, since the inference process is taken into account in the context of the evaluation of graph patterns over graphs.

A large amount of research can be captured by this level. In the following, we group it in two main groups: systems that work under a simple entailment regime, i.e. RDF entailment, and the ones that considers more complex reasoning formalisms, e.g. Description Logics and ASP. They are presented respectively in Sections 2.1.1 and 2.1.3. To enable the comprehension of the latter, we briefly review some prominent work in incremental reasoning in Section 2.1.2.

2.1.1. Stream reasoning at graph level under simple entailment regime

The first case we consider is the one where RDF entailment process is involved, that in SPARQL is named *simple entailment regime*. When a SPARQL query is

evaluated under this regime, the engine verifies if a graph pattern matches over the input graph, without considering ontological language or inference processes.

The first generation of stream processors for RDF data stream is represented in this category: engines where queries are registered and answers are continuously produced when new data is made available.

Several languages and prototypes were built by following the DSMS vision: sliding windows to capture finite portion of the streams to be queried through SPARQL as collections of RDF statements. Between 2010 and 2012, different groups proposed languages as the Continuous SPARQL (C-SPARQL) language [10], CQELS-QL [60] and SPARQL$_{stream}$ [16]. Such languages are similar and relies on the idea of extending SPARQL with sliding windows. For example, C-SPARQL extends the the FROM clause in order to support sliding windows, while CQELS-QL pushes sliding windows in the GRAPH clause. There are difference on the prototyping sides as well. The C-SPARQL engine and CQELS, implementing respectively C-SPARQL and CQELS-QL, take as input RDF streams and process them as extended SPARQL query processors (e.g. ARQ). morphstream, that implements SPARQL$_{stream}$, adopts a OBDA-like approach: it processes relational streams by transforming the query from SPARQL$_{stream}$ to one to be registered to a DSMS engine such as Esper and GSN.

A different approach is adopted by the Incremental eNgine for STANding Sparql – INSTANS [85]. As query language, it adopts SPARQL 1.1, with an extension on its query evaluation model to continuously query data streams. The implementation of INSTANS relies on the RETE algorithm: tasks are expressed as networks of queries and compiled in RETE-like structures to evaluate the results.

2.1.2. Incremental reasoning

Among the techniques to do reasoning, the closest to the stream reasoning idea are the incremental reasoning ones. They have been developed to cope with changes in knowledge bases. The intuition is to modify part of the materialisation of a knowledge base when updates happen, without re-materialising everything from scratch. As update operations occur these techniques identify the facts that have to be removed – since they were derived by deleted facts – and added – since they can be inferred by new facts. In this way, it is possible to avoid the recomputation of the whole materialisation upon changes.

DReD [97] has been proposed in 2005 by Volz et al. and it is inspired by techniques of view maintenance in databases. The idea is to compute two sets of axioms to be added and removed through a three-step process. In the first step, deletions are computed starting from the facts that should be deleted. This step produces an overestimation, due to the fact that some derived facts may still be inferred from other non-deleted facts. In the second step, the algorithm looks for these facts and remove them from the facts to be deleted. In the last step, new derivations are computed starting from the axioms that have been added to the knowledge base.

The first two steps of the DReD algorithm are critical for the performance of the technique: a considerable amount of computational effort may be required to identify the facts to be labelled as to be deleted in the first step and rectracted in the second. To overcome this limit, Motik and al. propose the Backward/For-

ward algorithm [72] in 2015. The idea is to use a combination of backward and forward inference to limit the number of overestimations in the first step. An implementation of this algorithm is available in RDFox [76], an in-memory datalog engine.

A different approach is the one considered by Ren and Pan in Truth Maintenance Systems in 2011 [84]. Differently from DReD, this technique builds and maintains a dependency graph. When deletions occur, this graph is used to decide if derived facts should be removed; when additions happen, the graph is updated by adding new edges and vertices. This approach is optimised for $\mathcal{EL}^{++}$.

DynamiTE [96] is a framework proposed in 2013 to maintain the materialisation incrementally. One of the novelties introduced by Dynamite is the parallelisation of the inference process. The framework supports DReD and a counting algorithm proposed by the authors for the ρDF fragment [75] of RDFS. On additions, DynamiTE recomputes the materialisation to add the new entailments through a parallel Datalog evaluation. On removals, it deletes the explicit and entailed axioms no longer valid. Several algorithms can perform this action: authors considered DReD and a counting algorithm they defined, that exploits the idea of counting the number of justifications that entailed it.

2.1.3. Stream reasoning at graph level under other entailment regimes

Moving to non-simple entailment regimes opens a set of challenges given by the introduction of the ontological languages and the associated inference processes.

One of the first attempts of building a stream reasoner is Streaming Knowledge Bases [98] in 2008. The idea was to pipe a DSMS engine, TelegraphCQ, with a reasoning engine, the Jena Rule engine. The system is able to performs RDFS inference.

The Incremental Materialization for RDF Streams algorithm (IMaRS) [28], proposed in 2011 for the RDFS+ ontological language makes a step further. This technique focuses the content of a window and the incrementally maintain it as the window slides. The intuition behind this algorithm is that in this setting, deletions can be foreseen and are not random as in the incremental reasoning setting described above. An expiration time annotation is associated to all the axioms involved in the materialisation, and such information is exploited to identify only the facts to be deleted, avoiding the overestimation typical of DReD. An implementation of IMaRS is available in Sparkwave [57]. It implements the algorithm on the top of the RETE algorithm and targets the RDF Schema entailment. RDF schema axioms are encoded as RETE rules and organised in a network. When new facts are added to the system, they are matched against the rules.

Another stream reasoner is StreamRule [70], proposed in 2013. As Streaming Knowledge Bases, the system is designed through a two-layer approach. The first layer is an RSP engine acting as a filter, that reduces the amount of data to be considered in the inference process. The second layer is a reasoning engine. What makes StreamRule unique w.r.t. the solution described above is the adoption of Answer Set Programming (ASP) [34], rather than a DL reasoner. ASP is a declarative problem-solving paradigm, characterized by rich modelling language maintaining very good performance, obtained by exploiting techniques from constraint

solving. ASP works with static knowledge; Incremental ASP [42] overcomes such a limit and extends ASP to compute the solutions incrementally.

The principle of working on a graph captured with a time window was also applied to inductive reasoning. In [11], authors used graph-level inductive reasoning in developing a recommendation engine that suggest topics to users.

2.2. Window-level entailment

We learned that graph-level entailment is a direct application of atemporal reasoning techniques over streams. The main drawback is that the process uses only a subset of the available information bought in the stream – it considers the data item contents but not the relative temporal annotations. For example CEP-like queries cannot be evaluated when stream reasoners such entailment.

Window-level entailment overcomes these limits, by applying the inference process context of the fixed windows produced by the window operators defined by the users while defining the task. Differently from the graph-level entailment, which works on graphs (ontologies), the window-level entailment applies the inference process to a window, i.e. a finite sequence of timestamped data items.

The Spatial and Temporal ontology Access with a Reasoning-based Query Language (STARQL) [77] proposes a framework to access and query heterogeneous sensor data through ontologies. STARQL is structured as a two-layer framework, composed by an Ontology Language, to model the data and its schema, and an Embedded Constraint Language, to compose the queries. STARQL offers window operators, clauses to express event matching and a layer to integrate static and streaming data. STARQL uses a sequence of time-annotated ontologies to make inference taking into account the temporal annotationsof the streaming data. Recently, STARQL has been implemented in Exastream [52], a system that adopts an OBDA approach, similar to morph-stream.

2.3. Stream-level entailment

One of the main limits of the window-level entailment is that it does consider only a recent portion of the ontology substream: when a sliding window computes a new window is computed, what happened in the past is forgotten and the entailment regime is applied from scratch restarting by the data contained in the new fixed window.

The stream-level entailment overcomes such a limit, considering a larger portion of the stream than the one defined by the user through window operator. Even if the name suggests that this entailment regime considers the whole stream, in this case the reasoning is made on the top of a landmark window That is a window that captures the stream from an initial time instant (e.g. when the source starts to supply the data when the engine starts to monitor the stream) up to now.

ETALIS (Event TrAnsaction Logic Inference System) [5] is a CEP-based stream reasoning engine. This query model processes streams where data items are annotated with two timestamps (i.e., time intervals). Users can specify event processing tasks in ETALIS using two declarative rule-based languages, ETALIS

Language for Events (ELE) and Event Processing SPARQL [4] (EP-SPARQL). The former language is more expressive than the latter, even if it is less usable. A common point is that complex events are derived from simpler events using deductive prolog rules. EP-SPARQL supports backward temporal reasoning over RDFS, continuously evaluating the query over the whole stream received by the engine.

The Logic-based framework for Analyzing Reasoning over Streams [13] (LARS) defines a logic for modelling stream-related axioms. LARS models the notion of stream as a sequence of time-annotated formulas. In addition to the usual boolean operators (and, or, implies, not), the language introduces operators, such as $\Diamond$ and $\Box$ to express the fact that a formula holds respectively at some time in the past and every time in the past; @ to state that a formula holds at a specific time instant. LARS formulas can be evaluated against the whole stream, or the scope can be limited through the usage of the $\boxplus$ operator, that models a window operator. Through this operator, LARS is also able to capture reasoning at window-level entailment.

The principle of reasoning at stream-level was also applied to inductive reasoning. In [63], authors used stream-level inductive reasoning in predicting Knowledge in an Ontology Stream.

3. Open problems and research questions

Stream reasoning research progressed and expanded its initial community to a growing number of practitioners. In this section, we can review the requirements presented in Section 1 in the light of the current state of the art and outline the open questions.

Table 2. A review of the stream reasoning requirements w.r.t. the current state of the art ($\star$=not specifically treated so far, $\star\star$= treated bot not resolved, $\star\star\star$=universally addressed by all studies)

Requirement	**Current Stream Reasoning**
R1: Volume	$\star$
R2: Velocity	$\star\star\star$
R3: Variety	$\star\star$
R4: Incompleteness	$\star$
R5: Noise	$\star$
R6: Timely fashion	$\star\star$
R7: Fine-grained access	$\star\star\star$
R8: Complex domains	$\star\star$
R9: What users need	$\star\star$

Table 2 summarizes the current state and serves as a indication towards possible directions for future stream reasoning research. Both stream processing and semantic web let users express and use complex notions in queries, such as

trends, skylines and aggregations. However, users are demanding for even more sophisticated features (R9), as discussed in Section 3.1.

Stream reasoning has always considered being in timely fashion (R6) important. It is possible to observe it by observing that time performance is usually the most relevant axis on which perform evaluations. Similarly, the focus has been on data streams with rich data models. In this sense, the requirement related to complex domains (R8) has been partially treated. However, we should consider this requirements in a broader sense. As we explain in Section 3.2 not only stream does reasoning needs to consider more expressive ontological languages, but it also needs to look at non-deductive reasoning.

Velocity and fine-grained access (R2 and R7) have been the two requirements at the centre of the research so far. Indeed, all studies on stream reasoning so far addressed the velocity dimension of the problem, considering streaming data the main object of investigation. In future, as we present in Section 3.3, we expect to assist to the rise of *smart* streams that leverage semantics for processing.

Volume and variety (R1 and R3) have not been properly addressed, but they are key requirements to enable stream reasoning in real-world big data environments. We discuss this in Section 3.4.

Finally, massive data is common, but real-world data is often characterized also by being imperfect, i.e. incomplete and noisy (R4 and R5). In this direction, stream reasoning research is still at the starting point. In Section 3.5 we analyze these research opportunities.

3.1. Towards queries that capture user needs

Queries connect users to stream reasoners: they encode the needs of the former in a format readable and processable by the latter. Research on query languages has been one of the leading topics of the stream reasoning trend up to now. There has been the first wave of languages, e.g. [10,60,16,4], refined and improved by new proposals to increase the expressiveness and the supported operations, e.g. [77,41]. In parallel, research has worked on comparing and contrasting such languages, e.g. [30,23,49,29].

Despite the results obtained so far, the research on query languages for stream reasoning is far from being complete. In this section, we present three directions that, in our opinion, are needed to reduce of the distance between stream reasoners and users (and consequently, real applications).

3.1.1. Capturing a wider set of tasks in queries

So far, language development focused primarily on SPARQL-like queries, where the main goal is matching of graph patterns over the streaming data under some entailment regimes. This kind of query sets the basis to filter, aggregate, as well as detect trends and complex event patterns in the data. However, there are other important tasks to be performed over streams. One example are spatial operations, that allow to define geographical and trajectory queries. They have been largely studied in semantic web and stream processing (e.g. [59,79,80]), but have not been tackled in stream reasoning so far.

In several domains, there is the need to perform complex analyses of streaming data that make heavy use of aggregation, correlation functions and arithmetic operations as well as of inductive methods. Examples of this trend are big data processors, where solutions such as Apache Spark [101], Apache Flink [19] and Twitter Heron [58] offer an extensive set of operations, from counting algorithms up to machine learning methods [69]. Solutions in this direction, which we may call *analytics-aware stream reasoning*, are starting to appear [54] and we expect a large interest in investigating this area in the near future.

Analytics provides insights, but users often also want fine grain access to the information that support those insights. This calls for optimizing a special type of continuous queries that include preferences in the information need, e.g. continuous top-k queries [74]. Some initial works exist on top-k query answering in SPARQL, but to the best of our knowledge there is only a vision paper [27] that cast some light in this direction. No one has attempted, yet, to investigate this area.

Moreover, the graph structure of the RDF data model opens the doors to the application of the classical algorithms related to the graph theory, e.g. path finding, connectivity, bipartiteness and clique search. How to apply such techniques to graph streams has been studied in the database literature [68].

Whilst there are examples for supporting these kinds of operations on static data (e.g., [55]), to the best of our knowledge, none of the stream reasoning-related query languages designed up to now support the above operations. Given the importance these non-deductive techniques have gained over the past few years (just think about the rise of machine learning), we strongly believe that the next years see the addition of new operations and the design of languages that support them. It may still be remain important to offer declarative languages (in addition to programmable APIs) to describe task, since they separate the expected output (the goal to the computation) from the way on which it should be computed and may—in some cases—be easier to handle by domain experts.

3.1.2. Tightening CEP and stream reasoning

It is worth considering the relation between CEP and stream reasoning. CEP queries are *de-facto* production rules: if a set of temporal constraints over events are satisfied (condition), then a complex event is triggered (action). That means a CEP-based stream reasoner has two inference components: one based on CEP rules and another one based on ontological axioms.

This consideration opens the door to several questions, such as: which part of the knowledge better fits the CEP rules and which one better fits ontology axioms? We can imagine that while some parts of the knowledge can be modelled as CEP rules or ontological axioms only, others may be captured by both formalisms. Such modelling decisions have an impact on the evaluation engine, its performance and the results it computes.

Another question is: how can the two components be combined? Following the framework presented in Section 2, the CEP evaluation is on the top of the ontology inference process. A well-marked separation between the two components is a safeguard with regards to complexity, but may not fit some applications. For example, a use case may require that the output of CEP reasoning will result

in changes to the ontological entailment, which in turn may trigger CEP rules. In such cases, the interaction of the two deduction sub-systems may affect the performance, with the risk to end up in endless processes. One possible solution to the problem is to identify which are the conditions under which CEP rules maintain the decidability of the stream reasoning process, similarly to the research on DL-safe rules in [73].

3.1.3. Forgetting knowledge

While processing streams, the identification of data that is useful for the current computation can be problematic. Several solutions have been proposed up to now, from sliding windows mechanisms DSMSs, to consumption and selection policies in CEPs. Such methods are key since they allow to discard data and keep under control the resource usage, and are currently adapted by stream reasoning solutions, e.g. sliding windows in CQELS and consumption policies in Etalis. However, they may lead to unexpected situations, e.g. a fact may fall out of the window while it is still relevant in the inference process.

In the context of stream reasoning, there is the opportunity to develop more sophisticated forgetting mechanisms. The notion of consumption is slightly different from the one of expiration [15]: consumed facts are not useful for processing, whilst expired facts are not true anymore. Incremental reasoning (Section 2.1.2) relies on the idea that data is removed because it expires. Similarly, initial work on stream reasoning like IMaRS assumes that consumption and expiration overlaps. However, this is not always the case: while some input data can be consumed (e.g. by a sliding window), some derivations may still be useful to solve the task. It follows that such derived data should not be consumed at the same time. We need models and techniques to manage consumption and expiration separately. In this way, the semantics of forgetting data becomes more precise, improving the quality of the engine answers.

Another approach is to exploit the semantics and knowledge about the data content to identify the relevant information. Ongoing research in stream processing is studying how to use knowledge about the streaming data to define windows, e.g. session windows in Google Dataflow [2] and frames [44]. The idea behind such techniques is to create windows by using specific information in the data (e.g. a session identifier in a server access log), rather than by using generic information such as time or number or elements. Moving from stream processing to stream reasoning, TEF-SPARQL [41] allows users to define facts as time-annotated elements by declaring a set of conditions on the input stream items. The approach presented in [99] introduces the notion of semantic importance, as a set of metrics assigned to the stream items, such as query contribution and provenance. These values lead the process of deciding which information should be consumed.

3.2. Towards sophisticated stream reasoning

As the stream reasoning name suggests, reasoning plays a crucial role. In this section, we describe two directions related to this topic. The first is the study of more expressive formalisms for deductive reasoning, such as temporal logics and, more in general, alternatives to description logics. The second is the integration of deductive reasoning with other types of reasoning, such as the inductive one.

3.2.1. Extending the range of ontological languages

After ten years of stream reasoning investigation, it appears that logic languages are most popular for stream reasoning. Most of the works reported in this paper use or slightly extend OWL 2 DL and its fragments.

In the next years, we think it is important to investigate other inference approaches and how they can be combined with OWL. First steps in this direction were taken using ASP [34,42]. The processing of data streams with Metric Temporal Logic was pioneered in [46] and it is now attracting again interests [93,18]. However, as already noticed in [25], several other logics also appear to be valid starting points, e.g. temporal action logic [33], step logic [36], active logic [35] and event calculus [88].

3.2.2. Integrating other types of reasoning

One of the characterising elements of stream reasoning is the presence of a model, possibly in combination with rich background knowledge. They are usually described through an ontological language, which enables the derivation of implicit information. When queries extend the set of operators as described above, an interesting challenge will be to investigate how several reasoning techniques can coexist.

For example, let's consider the system that integrates machine learning and deductive reasoning algorithms described in the end of Section 2.1.3. The authors of [11] built a system that pipes the results of an RSP engine into a machine learning system. However, this is just one possibility. The latter may also feed the former (as in [64]). Moreover, it is possible to exploit more interactive paradigms, where results of machine learning and reasoning techniques are continuously exchanged to achieve a given goal.

In addition to the machine learning mentioned above, we hope to observe an increasing number of explorations that study how to combine (deductive) stream reasoning with other techniques, such as probabilistic reasoning, planning, natural language processing, sentiment analysis.

3.3. Towards semantic streams

One of the ways to introduce semantics is through annotations: they can describe data in a machine-readable fashion, and can consequently be read and processed by systems. Even if this is a typical Semantic Web scenario, so far this direction has not deeply been investigated in stream reasoning. There is a potential value in annotating streams: it enables stream processors to access a description of the stream and to use it to take decisions, e.g. dynamic discovery and selection of data sources.

The stream descriptor can provide quantitative and qualitative information about the content, e.g. statistical data about frequency and size, information about the vocabularies adopted in the stream items and provenance. Such information may help the stream reasoner in taking some decisions on how to process the stream, even before starting to receive it, e.g. relevancy of the data w.r.t. the registered queries, query optimisation or need for data eviction techniques.

The description of the stream may provide knowledge about its content. We can indeed observe that stream content is heterogeneous in nature. A stream may bring states (e.g. the temperature in a room), producing data items in a periodic way (e.g. every 2 seconds) or when the state change (e.g. when the temperature increases of one degree). Streams may also describe sequences of actions (e.g. the log of the user interaction in a Web site).

When the query developer aims at describing a task for a stream reasoner, it is up to him to know what the stream carries and consequently to take proper decisions. By exploiting annotations about the stream it would be possible to improve the interoperability at application level: tools may assist the development of queries over the stream and help domain experts with limited technical skills.

3.4. Towards scalable stream reasoners

The progress on stream reasoning foundations sets the basis to build a new generation of more sophisticated stream reasoning frameworks. Researchers integrate reasoning processes in a gradual way, from the application of reasoning over the window content as an ontology, e.g. Streaming Knowledge Bases and Sparkwave, to more sophisticated solutions that take into account also the time dimension and the transient nature of the data stream items in the reasoning process, e.g. ExaStream. In the following, we will present the main challenges that stream reasoner researchers and engineers are going to cope with in order to build new engines.

Scalability is an open and exciting challenge: which order of throughput will stream reasoners be able to support?

As known, there are theoretical results that set some constraints to the velocity that stream reasoners may reach. Reasoning computational complexity is strictly related to the adopted ontological language: the query answering task of the three OWL 2 dialects can vary from AC^0 and polynomial (w.r.t. ABox size) up to NP and EXP (w.r.t. ABox, TBox and query sizes). In this sense, it is not possible to expect that a stream reasoner is going to reach the performance of stream processing solutions.

However, here we observe a trade-off similar the one between memory size and access time in computer systems, which is solved using a memory hierarchy. As proposed in [90], a hierarchy of processing steps of increasing complexity can tackle scalability. Technically, this is doable because reasoning can speed up by pushing down processing steps in the hierarchy (e.g., query rewriting) and by post-processing the results coming up from the layer underneath.

3.4.1. Approximating the results

A typical approach to scale stream processing systems is to move from complete and exact outputs to approximated ones. Such answers are acceptable in a wide range of scenarios, in particular when tasks require aggregations and small errors are tolerable, e.g. counting the number of people entering a square or calculating the average temperature in a building.

There are several ways to achieve approximation. Load shedding [92] techniques capture the idea that the system can produce the output by processing

a portion of the stream and throwing away the remaining part. Over the years, several load shedding techniques have been proposed, to select (with the minimal effort) the data to be evicted to minimise the error of the answer. [40,14] introduce data eviction in stream reasoning. The main difference of applying load shedding in stream reasoning is the more complex nature of the data items and the reasoning process itself. Removing data in aggregation operations can introduce errors that can be estimated and controlled. In stream reasoning, the eviction of a single fact may drastically affect the inference process, with high impact on the correctness of the answer.

Besides data eviction, that discards data before processing it, summarization exploits the idea that output can be computed starting with a summary of the input data rather than from the whole dataset. Both stream processing and reasoning have extensively used these techniques. Summaries in stream processing, where are also named sketches, are used to reduce the memory consumption of the engine and to approximate results of aggregations [32]. Several sketch methods are available, usually tailored to specific kinds of aggregation query, e.g. [20]. Summaries in reasoning follow a similar idea [38,102]: the ontology (or part of it) is transformed in a *smaller* representation, over which it is possible to perform reasoning tasks.

While the above approaches focus on data, other techniques work to simplify the processing, gaining in performance while introducing some degree of approximation. An example is [78]: authors propose methods to reason over ontologies represented in OWL 2 DL through inference processes of OWL 2 EL, i.e. a tractable fragment of OWL 2 DL. Axioms that cannot be treated in OWL 2 EL (e.g. inverse properties) are managed through ad-hoc rules, applied before and after the reasoning process. In this way, it is usually possible to apply faster algorithms to perform the reasoning task, losing the correctness guarantees.

3.4.2. Parallelizing and distributing the stream reasoners

Parallelization and distribution can be seen as an opportunity or as a challenge. So far stream reasoning was addressed bringing data streams and contextual (or background) knowledge in one single point. If this point is a cluster, parallelization and distribution (in the cluster) is an approach to engineer scalable and elastic stream reasoners. The first part of this section discusses this opportunity. However, data streams are parallel and distributed sources in nature. The same applies to many Web data sources to join data streams with. Pushing computation to those sources (see also Fog Computing as a broader research field) is the challenge presented in the second half of this section.

When talking about parallelization and distribution, the intuitive idea is that the processing can be split and executed at the same time in multiple locations, e.g. multiple processors in a machine or different nodes in a cluster or a cloud platform.

Looking at stream reasoning, we can find only some attempts in this direction, such as However, in the adjacent areas several investigations are available: in stream processing, e.g. [1,56], in big data processing, e.g. [101,19,58], and in SPARQL query evaluation, e.g. [45,81,86].

Engineering a distributed stream reasoner is a challenging task that touches several scientific and technical problems. Ideally, such a system should maximise the throughput, finding a perfect balancing between network load (i.e. how data route through the nodes) and machine load (i.e. the computation loads assigned to the nodes). It is, therefore, important to understand which are the best topologies, operators and data distributions to perform the stream reasoning task.

One initial study in this direction is [37]: authors propose to apply graph partitioning over linked data streams in the context of continuous query answering. The goal is to reduce the network load and consequently improve the performance of the system. Further work is needed to understand the problem of how to cope with the presence of inference processes in the context of reasoning.

Several contributions are available on parallel and distributed reasoning in non-streaming settings. One possible way to achieve this is to treat the data as a set of interconnected ontologies: first reasoning over each ontology is locally performed and the inference completes by exchanging messages, e.g. [87,9]. More recent work exploits new parallelization paradigms to perform the reasoning process, e.g. [95,71]. The usual problems of distributed reasoners are related to termination, i.e. decide when nodes can stop the computation, and to duplicates, i.e. the less duplicated derivations, the higher the performance. The problem in stream reasoning is exacerbated by the need to provide reactive answers.

The last direction we highlight is related to the data distribution problem. When considering scenarios like the Web, it often happens that the data is distributed, controlled by several actors and exposed through services, e.g. SPARQL endpoints or Web APIs, working with either pull or push mechanisms. In such contexts, it often happens that data cannot be centralised and permanently stored in local memory of the stream engine. For example, data cannot fit in the engine memory, it may change over time (and services may not publish notifications), or can be only stored for limited amounts of time for legal reasons. This setting poses interesting challenges to stream reasoning, where responsiveness is one of the most critical requirements. One of the possible ways to see the problem is the following: given the data that can be pushed to the processor, which is the needed contextual remote data to be pulled to solve the given task? In other words, the challenge is how to achieve the integration of the local and remote data without losing responsiveness. The initial effort [24,39] works in the setting of linked data integration of streaming and contextual data for query evaluation purposes. The idea is to adopt caches where to store a portion of the remote data, updating it depending on the recent stream content. Another relevant work is the one in [48], where authors study the problem of integrating distributed dynamic data and process it through a set of rules. Further techniques are required, since moving and replicating data in the processing nodes impacts the performance.

3.4.3. Reasoning outside the window

The stream-level entailment offers an additional opportunity for stream reasoning. The main challenge is on the resource usage. Given the absence of a sliding window, which introduces a consumption mechanism for the formula validities, a reasoner operating under stream-level entailment may require an unrealistic amount of memory and processing capabilities. In other words, is it possible to

build a generic framework to perform stream reasoning under stream-level entailment? Under which conditions would it work without exceeding the assigned amount of resources? Neither LARS nor EP-SPARQL, described in Section 2.3, is targeting such problems.

LARS allows defining formulas over the whole stream without limiting the reasoning effect to the window content. However, it is a theoretical framework and no implementations are available at the moment.

A stream is infinite by definition, but this does not mean that it requires infinite memory to perform all computations. If a stream is an infinite sequence of numbers, a DSMS can compute the average of those numbers incrementally with finite memory. This relates to the database notion of non-blocking operators. Our intuition is that reasoning outside the window is feasible for *non-blocking reasoning tasks*. For instance, it is possible to compute the materialisation of a stream under the DL-fragment of RDFS when the TBox is fixed. The inference can be applied to each stream item independently, avoiding the storage of streaming data to compute future derivations. That is the case of EP-SPARQL and Etalis: given a fixed schema, all the inference rules are triggered by one data item. In other words, the reasoning process does not need to access the past to compute new derivations. Etalis can still have problems with the memory management: the registered queries may require an infinite amount of memory to compute all the solutions. However, this is a problem related to the CEP-nature of the system rather than to the reasoning one.

Building systems that perform window-level entailment require coping with the problem to introduce controls on the resource usage, in particularly on the memory usage. A task is usually modelled through a schema, one or more data streams and a set of operations (represented through a query). We hypothesise the existence of a class of queries that guarantee an upper bound on resource usages, given an underlying ontological language. EP-SPARQL supplies evidence about this since, as explained above, it is possible to execute its query with a limited amount of memory usage.

Answers to the above question can enable the constructions of new algorithms to perform stream-level entailment, able to analyse the current scenario and decide which strategy to adopt to execute the query in a *safe* way avoiding to exceed the assigned resource need. For example, a system may decide to compute the correct answer when the conditions allow it, and move to approximated results in the other cases, by adopting stream item consumption and summaries.

3.5. Towards robustness to imperfect data

The road to the usage of stream reasoning in real environments goes through the ability to process imperfect data. Stream processing has always coped with imperfect data. However, deductive reasoning is sensible to noise and incomplete data: one single error may lead a system to an overall inconsistent state. In this section, we discuss the open problems related to stream reasoning with imperfect data, analyzing first the ones related to heterogeneity and then the ones related to noise.

3.5.1. Overcoming the heterogeneity

As we depicted above, reasoning offers a set of methods and solution to cope with the heterogeneity. In particular, such techniques focus on the problem of heterogeneity at schema-level: when models are different, OBDA is a solution to access such data through a conceptual shared model. Tackling this problem is important, but heterogeneity issues afflict the Stream Reasoning scenario in other ways.

Data streams can be heterogeneous because they are not synchronised. For instance, imagine two cameras monitoring the same street that report every minute the number of cars they counted. They report the same number only if they are in sync. If one camera starts the counting 20 seconds before the other one, the two counting will differ, but this second situation is normal in an open world. Similarly, a continuous query may require to join a data stream with background knowledge served by a pull API, e.g. to monitor the changes in the number of followers the users mentioned in a stream of microposts, because user profiles normally are only available via pull API. There is no guarantee that the API is returning values that are temporally aligned with the data stream. The solution of this problem requires both a rich semantic description of the data streams (see Section 3.3) and an extension of Stream Reasoning methods. A possible research direction is to model all sources as virtual data streams and to offer a synchronisation service to the upper layers of a next generation Stream Reasoner. Such a layer may also offer the opportunity to homogenise access to stored time-series and continuously computed predictions.

Another service that such a layer can provide is the on demand discretisation abstraction of quantities as facts. For instance, a stream reasoner may prefer the cameras of the example above to report the level of congestion in the street rather than the counting. However, different applications (or even the same application in different moments) may require different discretisations. A first step in this direction is reported in [94], where the authors report on a system able to answer continuous queries over data stream applying a rewriting method for query answering over temporal fuzzy DL-Lite ontologies.

Finally, heterogeneity can go beyond the data and affect the stream reasoner as well. Existing stream reasoning techniques differ from each other. It is evident when the goal is different, but it happens even when they perform the same task and user may expect the same output. There is an ongoing effort on studying heterogeneity in stream reasoners. RSEP-QL [30,29] is a reference model to explain heterogeneity in stream reasoners under simple entailment, while LARS [23] has been introduced to capture the behaviour of stream reasoners under more complex regimes. Studying and understanding heterogeneity of stream reasoners is important to achieve interoperability, standardization and comparison.

3.5.2. Copying with noise

Stream processing has always coped with noise [22]. We can distinguish two different types of noise, given that the streaming item is composed of some content and a time annotation.

The first one affects the stream item content. Sensors may break and stop to work. Or worse, they may degrade their precision and provide observations with

some degree of error, leading the processing to wrong results. The problem is not limited to the sensor-generated data: streams generated from human interactions may contain syntactical or semantical errors in the data items. In a stream reasoning scenario, this may lead to wrong conclusions and consequently to wrong decisions and actions.

When the stream has a very simple schema (e.g. a time series), statistical methods can supply solutions to manage the noise. However, when we consider more complex schemata, more sophisticated methods may be required. Recently, techniques to cope with noise in stream reasoning emerged [8,62]. The idea is to adopt machine learning to process noisy data and to learn models over which to apply deductive reasoning processes.

The other kind of noise is the one that affects the temporal annotations on the data item. When considering a single stream, the noise manifests in the out-of-order phenomena: for some reasons (e.g. during stream production or transmission), the stream engine is receiving stream items in a wrong order. However, the problem may become more complicated when considering multiple streams. Different sources are producing and publishing them, and this can lead to the introduction of noise, since they may adopt different, not perfectly aligned clocks to generate the temporal annotations. Moreover, one stream can be received sensibly before another one, since the transmission time between two points of a network can require different time.

Several solutions have been proposed to cope with noise in time in the context of stream and event processing, e.g. [89,65], and stream reasoning can get an advantage of such techniques as well. However, we believe that semantics can offer the opportunity to enhance the existing methods to manage such problems. Engines can use the rich and machine-readable descriptions of the data streams to monitor if the received data stream has the correct order. When not, the system would be aware of issues in the stream, with the possibility to take actions.

4. Conclusions

We are observing an impressive increase of the speed of data production and consumption. In this paper, we explained how stream reasoning aims at providing methods and tools to perform sophisticated analyses of such data.

In the beginning, stream reasoning grew with the idea of building such analyses on top of logical and deductive inference. DSMS, CEP and Semantic Web offered solid starting points to kick off the research. Through the years, we have observed the creation of languages, techniques and framework. Those studies pushed stream reasoning in a broader area, introducing reasoning techniques beyond the deductive ones. Semantic Scholar and Google Scholar count respectively around 320 and 1000 articles containing "stream reasoning"[4], published in different areas, from semantic web to artificial intelligence. However, there is still a lot of research to be done.

We presented the main directions over which stream reasoning research can continue. Stream reasoners should offer rich query languages, able offer a wider

[4]Checked at February 2017

set of operators to encode user needs, and the dedicated infrastructure to evaluate them. Reasoning has taken a more generic connotation, and now it includes inductive reasoning techniques in addition to logical ones. This trend will grow, combining different techniques to overcome their limits. Solutions that will need to be engineered in scalable frameworks, able to integrate and reason over huge amounts of heterogeneous data while guaranteeing time requirements. And it will be important to fill the gaps between theoretical models and reality, making stream reasoning solutions robust and able to cope with issues such as noise, heterogeneity. In parallel, it will be important to identify real problems and scenarios where stream reasoning may be a solution. Internet of Things and Industry 4.0 are examples of areas where to apply stream reasoning results. Moreover, it is necessary to develop benchmarking and evaluation activities, to compare and contrast the current solutions.

Results obtained up to now are important. In addition to the publications, some of the mature solutions have been exploited in real scenarios, such as social media analytics and turbine monitoring. We should get inspired by such results, and see them as the foundations to build new research and to reach new ambitious achievements.

References

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. B. Zdonik. The design of the borealis stream processing engine. In *CIDR*, pages 277–289, 2005.

[2] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *PVLDB*, 8(12):1792–1803, 2015.

[3] M. I. Ali, F. Gao, and A. Mileo. Citybench: A configurable benchmark to evaluate RSP engines using smart city datasets. In *International Semantic Web Conference (2)*, volume 9367 of *Lecture Notes in Computer Science*, pages 374–389. Springer, 2015.

[4] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *WWW*, pages 635–644. ACM, 2011.

[5] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic. Stream reasoning and complex event processing in ETALIS. *Semantic Web*, 3(4):397–407, 2012.

[6] A. Arasu, S. Babu, and J. Widom. CQL: A language for continuous queries over streams and relations. In *DBPL*, volume 2921 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2003.

[7] A. Bachmann, C. Bird, F. Rahman, P. T. Devanbu, and A. Bernstein. The missing links: bugs and bug-fix commits. In *SIGSOFT FSE*, pages 97–106. ACM, 2010.

[8] M. Balduini, I. Celino, D. Dell'Aglio, E. Della Valle, Y. Huang, T. K. Lee, S. Kim, and V. Tresp. Reality mining on micropost streams - deductive and inductive reasoning for personalized and location-based recommendations. *Semantic Web*, 5(5):341–356, 2014.

[9] J. Bao, D. Caragea, and V. Honavar. A tableau-based federated reasoning algorithm for modular ontologies. In *Web Intelligence*, pages 404–410. IEEE Computer Society, 2006.

[10] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. C-SPARQL: a continuous query language for RDF data streams. *Int. J. Semantic Computing*, 4(1):3–25, 2010.

[11] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, Y. Huang, V. Tresp, A. Rettinger, and H. Wermser. Deductive and inductive stream reasoning for semantic social media analytics. *IEEE Intelligent Systems*, 25(6):32–41, 2010.

[12] D. F. Barbieri and E. Della Valle. A proposal for publishing data streams as linked data - A position paper. In *LDOW*, volume 628 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[13] H. Beck, M. Dao-Tran, T. Eiter, and M. Fink. LARS: A logic-based framework for analyzing reasoning over streams. In *AAAI*, pages 1431–1438. AAAI Press, 2015.

[14] F. Belghaouti, A. Bouzeghoub, Z. Kazi-Aoul, and R. Chiky. POL: A pattern oriented load-shedding for semantic data stream processing. In *WISE (2)*, volume 10042 of *Lecture Notes in Computer Science*, pages 157–171, 2016.

[15] I. Botan, G. Alonso, P. M. Fischer, D. Kossmann, and N. Tatbul. Flexible and scalable storage management for data-intensive stream processing. In *EDBT*, volume 360 of *ACM International Conference Proceeding Series*, pages 934–945. ACM, 2009.

[16] J. Calbimonte, H. Jeung, Ó. Corcho, and K. Aberer. Enabling query technologies for the semantic sensor web. *Int. J. Semantic Web Inf. Syst.*, 8(1):43–63, 2012.

[17] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo. The MASTRO system for ontology-based data access. *Semantic Web*, 2(1):43–53, 2011.

[18] D. Calvanese, E. G. Kalayci, V. Ryzhikov, and G. Xiao. Towards practical OBDA with temporal ontologies - (position paper). In *RR*, volume 9898 of *Lecture Notes in Computer Science*, pages 18–24. Springer, 2016.

[19] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink$^{\text{TM}}$: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.

[20] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[21] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, 2012.

[22] G. Cugola, A. Margara, M. Matteucci, and G. Tamburrelli. Introducing uncertainty in complex event processing: model, implementation, and validation. *Computing*, 97(2):103–144, 2015.

[23] M. Dao-Tran, H. Beck, and T. Eiter. Contrasting RDF stream processing semantics. In *JIST*, volume 9544 of *Lecture Notes in Computer Science*, pages 289–298. Springer, 2015.

[24] S. Dehghanzadeh, D. Dell'Aglio, S. Gao, E. Della Valle, A. Mileo, and A. Bernstein. Approximate continuous query answering over streams and dynamic linked data sets. In *ICWE*, volume 9114 of *Lecture Notes in Computer Science*, pages 307–325. Springer, 2015.

[25] E. Della Valle, S. Ceri, F. van Harmelen, and D. Fensel. It's a streaming world! reasoning upon rapidly changing information. *IEEE Intelligent Systems*, 24(6):83–89, 2009.

[26] E. Della Valle, D. Dell'Aglio, and A. Margara. Taming velocity and variety simultaneously in big data with stream reasoning: tutorial. In *DEBS*, pages 394–401. ACM, 2016.

[27] E. Della Valle, S. Schlobach, M. Krötzsch, A. Bozzon, S. Ceri, and I. Horrocks. Order matters! harnessing a world of orderings for reasoning over massive data. *Semantic Web*, 4(2):219–231, 2013.

[28] D. Dell'Aglio, M. Balduini, and E. D. Valle. Applying semantic interoperability principles to data stream management. In *Data Management in Pervasive Systems*, Data-Centric Systems and Applications, pages 135–166. Springer, 2015.

[29] D. Dell'Aglio, M. Dao-Tran, J. Calbimonte, D. Le Phuoc, and E. Della Valle. A query model to capture event pattern matching in RDF stream processing query languages. In *EKAW*, volume 10024 of *Lecture Notes in Computer Science*, pages 145–162, 2016.

[30] D. Dell'Aglio, E. Della Valle, J. Calbimonte, and Ó. Corcho. RSP-QL semantics: A unifying query model to explain heterogeneity of RDF stream processing systems. *Int. J. Semantic Web Inf. Syst.*, 10(4):17–44, 2014.

[31] C. Dobbins, P. Fergus, M. Merabti, and D. Llewellyn-Jones. Monitoring and measuring sedentary behaviour with the aid of human digital memories. In *CCNC*, pages 395–398. IEEE, 2012.

[32] A. Dobra, M. N. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate

queries over data streams. In *SIGMOD Conference*, pages 61–72. ACM, 2002.

[33] P. Doherty, J. Gustafsson, L. Karlsson, and J. Kvarnström. TAL: temporal action logics language specification and tutorial. *Electron. Trans. Artif. Intell.*, 2:273–306, 1998.

[34] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. *Artif. Intell.*, 172(12-13):1495–1539, 2008.

[35] J. Elgot-Drapkin, S. Kraus, M. Miller, M. Nirkhe, and D. Perlis. Active logics: A unified formal approach to episodic reasoning. Technical report, 1999.

[36] J. J. Elgot-Drapkin. *Step-logic: Reasoning situated in time*. PhD thesis, University of Maryland at College Park, 1988.

[37] L. Fischer, T. Scharrenbach, and A. Bernstein. Scalable linked data stream processing via network-aware workload scheduling. In *SSWS@ISWC*, volume 1046 of *CEUR Workshop Proceedings*, pages 81–96. CEUR-WS.org, 2013.

[38] A. Fokoue, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas. The summary abox: Cutting ontologies down to size. In *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 343–356. Springer, 2006.

[39] S. Gao, D. Dell'Aglio, S. Dehghanzadeh, A. Bernstein, E. Della Valle, and A. Mileo. Planning ahead: Stream-driven linked-data access under update-budget constraints. In *International Semantic Web Conference (1)*, volume 9981 of *Lecture Notes in Computer Science*, pages 252–270, 2016.

[40] S. Gao, T. Scharrenbach, and A. Bernstein. The CLOCK data-aware eviction approach: Towards processing linked data streams with limited resources. In *ESWC*, volume 8465 of *Lecture Notes in Computer Science*, pages 6–20. Springer, 2014.

[41] S. Gao, T. Scharrenbach, J. Kietz, and A. Bernstein. Running out of bindings? integrating facts and events in linked data stream processing. In *SSN-TC/OrdRing@ISWC*, volume 1488 of *CEUR Workshop Proceedings*, pages 63–74. CEUR-WS.org, 2015.

[42] M. Gebser, O. Sabuncu, and T. Schaub. An incremental answer set programming based system for finite model computation. *AI Commun.*, 24(2):195–212, 2011.

[43] M. Giese, A. Soylu, G. Vega-Gorgojo, A. Waaler, P. Haase, E. Jiménez-Ruiz, D. Lanti, M. Rezk, G. Xiao, Ö. L. Özçep, and R. Rosati. Optique: Zooming in on big data. *IEEE Computer*, 48(3):60–67, 2015.

[44] M. Grossniklaus, D. Maier, J. Miller, S. Moorthy, and K. Tufte. Frames: data-driven windows. In *DEBS*, pages 13–24. ACM, 2016.

[45] S. Gurajada, S. Seufert, I. Miliaraki, and M. Theobald. Triad: a distributed shared-nothing RDF engine based on asynchronous message passing. In *SIGMOD Conference*, pages 289–300. ACM, 2014.

[46] F. Heintz and P. Doherty. Dyknow: An approach to middleware for knowledge processing. *Journal of Intelligent and Fuzzy Systems*, 15(1):3–13, 2004.

[47] Z. Huang and H. Stuckenschmidt. Reasoning with multi-version ontologies: A temporal logic approach. In *International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 2005.

[48] T. Käfer, A. Harth, and S. Mamessier. Towards declarative programming and querying in a distributed cyber-physical system: The i-vision case. In *CPS Data*, pages 1–6. IEEE, 2016.

[49] R. Keskisärkkä. Query templates for RDF stream processing. In *SR+SWIT@ISWC*, volume 1783 of *CEUR Workshop Proceedings*, pages 25–36. CEUR-WS.org, 2016.

[50] R. Keskisärkkä and E. Blomqvist. Semantic complex event processing for social media monitoring-a survey. In *Proceedings of Social Media and Linked Data for Emergency Response (SMILE) Co-located with the 10th Extended Semantic Web Conference, Montpellier, France. CEUR workshop proceedings (May 2013)*, 2013.

[51] E. Kharlamov, S. Brandt, M. Giese, E. Jiménez-Ruiz, S. Lamparter, C. Neuenstadt, Ö. L. Özçep, C. Pinkel, A. Soylu, D. Zheleznyakov, M. Roshchin, S. Watson, and I. Horrocks. Semantic access to siemens streaming data: the optique way. In *International Semantic Web Conference (Posters & Demos)*, volume 1486 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.

[52] E. Kharlamov, S. Brandt, E. Jiménez-Ruiz, Y. Kotidis, S. Lamparter, T. Mailis,

C. Neuenstadt, Ö. L. Özçep, C. Pinkel, C. Svingos, D. Zheleznyakov, I. Horrocks, Y. E. Ioannidis, and R. Möller. Ontology-based integration of streaming and static relational data with optique. In *SIGMOD Conference*, pages 2109–2112. ACM, 2016.

[53] E. Kharlamov, D. Hovland, E. Jiménez-Ruiz, D. Lanti, H. Lie, C. Pinkel, M. Rezk, M. G. Skjæveland, E. Thorstensen, G. Xiao, D. Zheleznyakov, and I. Horrocks. Ontology based access to exploration data at statoil. In *International Semantic Web Conference (2)*, volume 9367 of *Lecture Notes in Computer Science*, pages 93–112. Springer, 2015.

[54] E. Kharlamov, Y. Kotidis, T. Mailis, C. Neuenstadt, C. Nikolaou, Ö. L. Özçep, C. Svingos, D. Zheleznyakov, S. Brandt, I. Horrocks, Y. E. Ioannidis, S. Lamparter, and R. Möller. Towards analytics aware ontology based access to static and streaming data. In *International Semantic Web Conference (2)*, volume 9982 of *Lecture Notes in Computer Science*, pages 344–362, 2016.

[55] C. Kiefer, A. Bernstein, and A. Locher. Adding data mining support to SPARQL via statistical relational learning methods. In *ESWC*, volume 5021 of *Lecture Notes in Computer Science*, pages 478–492. Springer, 2008.

[56] A. Koliousis, M. Weidlich, R. C. Fernandez, A. L. Wolf, P. Costa, and P. R. Pietzuch. SABER: window-based hybrid stream processing for heterogeneous architectures. In *SIGMOD Conference*, pages 555–569. ACM, 2016.

[57] S. Komazec, D. Cerri, and D. Fensel. Sparkwave: continuous schema-enhanced pattern matching over RDF data streams. In *DEBS*, pages 58–68. ACM, 2012.

[58] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja. Twitter heron: Stream processing at scale. In *SIGMOD Conference*, pages 239–250. ACM, 2015.

[59] K. Kyzirakos, M. Karpathiotakis, K. Bereta, G. Garbis, C. Nikolaou, P. Smeros, S. Giannakopoulou, K. Dogani, and M. Koubarakis. The spatiotemporal RDF store strabon. In *SSTD*, volume 8098 of *Lecture Notes in Computer Science*, pages 496–500. Springer, 2013.

[60] D. Le Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *International Semantic Web Conference (1)*, volume 7031 of *Lecture Notes in Computer Science*, pages 370–388. Springer, 2011.

[61] F. Lécué. Diagnosing changes in an ontology stream: A DL reasoning approach. In *AAAI*. AAAI Press, 2012.

[62] F. Lécué. Towards scalable exploration of diagnoses in an ontology stream. In *AAAI*, pages 87–93. AAAI Press, 2014.

[63] F. Lécué and J. Z. Pan. Predicting knowledge in an ontology stream. In *IJCAI*, pages 2662–2669. IJCAI/AAAI, 2013.

[64] F. Lécué, S. Tallevi-Diotallevi, J. Hayes, R. Tucker, V. Bicer, M. L. Sbodio, and P. Tommasi. Smart traffic analytics in the semantic web with STAR-CITY: scenarios, system and lessons learned in dublin city. *J. Web Sem.*, 27:26–33, 2014.

[65] M. Liu, M. Li, D. Golovnya, E. A. Rundensteiner, and K. T. Claypool. Sequence pattern query processing over out-of-order event streams. In *ICDE*, pages 784–795. IEEE Computer Society, 2009.

[66] D. Luckham. The power of events: An introduction to complex event processing in distributed enterprise systems. In *RuleML*, volume 5321 of *Lecture Notes in Computer Science*, page 3. Springer, 2008.

[67] A. Margara, J. Urbani, F. van Harmelen, and H. E. Bal. Streaming the web: Reasoning over dynamic data. *J. Web Sem.*, 25:24–44, 2014.

[68] A. McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014.

[69] X. Meng, J. K. Bradley, B. Yavuz, E. R. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. Mllib: Machine learning in apache spark. *CoRR*, abs/1505.06807, 2015.

[70] A. Mileo, A. Abdelrahman, S. Policarpio, and M. Hauswirth. Streamrule: A nonmonotonic stream reasoning system for the semantic web. In *RR*, volume 7994 of *Lecture Notes in Computer Science*, pages 247–252. Springer, 2013.

[71] B. Motik, Y. Nenov, R. Piro, I. Horrocks, and D. Olteanu. Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In *AAAI*, pages 129–137. AAAI Press, 2014.

[72] B. Motik, Y. Nenov, R. E. F. Piro, and I. Horrocks. Incremental update of datalog materialisation: the backward/forward algorithm. In *AAAI*, pages 1560–1568. AAAI Press, 2015.

[73] B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. *J. Web Sem.*, 3(1):41–60, 2005.

[74] K. Mouratidis, S. Bakiras, and D. Papadias. Continuous monitoring of top-k queries over sliding windows. In S. Chaudhuri, V. Hristidis, and N. Polyzotis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 635–646. ACM, 2006.

[75] S. Muñoz, J. Pérez, and C. Gutierrez. Simple and efficient minimal RDFS. *J. Web Sem.*, 7(3):220–234, 2009.

[76] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, and J. Banerjee. Rdfox: A highly-scalable RDF store. In *International Semantic Web Conference (2)*, volume 9367 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2015.

[77] Ö. L. Özçep, R. Möller, and C. Neuenstadt. A stream-temporal query language for ontology based data access. In *Description Logics*, volume 1193 of *CEUR Workshop Proceedings*, pages 696–708. CEUR-WS.org, 2014.

[78] J. Z. Pan, Y. Ren, and Y. Zhao. Tractable approximate deduction for OWL. *Artif. Intell.*, 235:95–155, 2016.

[79] M. Perry and J. Herring. OGC GeoSPARQL - A Geographic Query Language for RDF Data. Technical report, Open Geospatial Consortium, 2012.

[80] M. Potamias, K. Patroumpas, and T. K. Sellis. Sampling trajectory streams with spatiotemporal criteria. In *SSDBM*, pages 275–284. IEEE Computer Society, 2006.

[81] A. Potter, B. Motik, Y. Nenov, and I. Horrocks. Distributed RDF query answering with dynamic data exchange. In *International Semantic Web Conference (1)*, volume 9981 of *Lecture Notes in Computer Science*, pages 480–497, 2016.

[82] E. Prud'hommeaux, S. Harris, and A. Seaborne. SPARQL 1.1 Query Language. Technical report, W3C, 2013.

[83] D. Puiu, P. M. Barnaghi, R. Toenjes, D. Kuemper, M. I. Ali, A. Mileo, J. X. Parreira, M. Fischer, S. Kolozali, N. FarajiDavar, F. Gao, T. Iggena, T. Pham, C. Nechifor, D. Puschmann, and J. Fernandes. Citypulse: Large scale data analytics framework for smart cities. *IEEE Access*, 4:1086–1108, 2016.

[84] Y. Ren and J. Z. Pan. Optimising ontology stream reasoning with truth maintenance system. In *CIKM*, pages 831–836. ACM, 2011.

[85] M. Rinne, M. Solanki, and E. Nuutila. Rfid-based logistics monitoring with semantics-driven event processing. In *DEBS*, pages 238–245. ACM, 2016.

[86] A. Schätzle, M. Przyjaciel-Zablocki, S. Skilevic, and G. Lausen. S2RDF: RDF querying with SPARQL on spark. *PVLDB*, 9(10):804–815, 2016.

[87] L. Serafini and A. Tamilin. DRAGO: distributed reasoning architecture for the semantic web. In *ESWC*, volume 3532 of *Lecture Notes in Computer Science*, pages 361–376. Springer, 2005.

[88] A. Skarlatidis, G. Paliouras, A. Artikis, and G. A. Vouros. Probabilistic event calculus for event recognition. *ACM Trans. Comput. Log.*, 16(2):11:1–11:37, 2015.

[89] U. Srivastava and J. Widom. Flexible time management in data stream systems. In *PODS*, pages 263–274. ACM, 2004.

[90] H. Stuckenschmidt, S. Ceri, E. D. Valle, and F. van Harmelen. Towards expressive stream reasoning. In *Semantic Challenges in Sensor Networks*, volume 10042 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2010.

[91] S. Tallevi-Diotallevi, S. Kotoulas, L. Foschini, F. Lécué, and A. Corradi. Real-time urban monitoring in dublin using semantic and stream technologies. In *International Semantic Web Conference (2)*, volume 8219 of *Lecture Notes in Computer Science*, pages 178–194. Springer, 2013.

[92] N. Tatbul, U. Çetintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Load shed-

ding in a data stream manager. In *VLDB*, pages 309–320, 2003.

[93] M. Tiger and F. Heintz. Stream reasoning using temporal logic and predictive probabilistic state models. In *TIME*, pages 196–205. IEEE Computer Society, 2016.

[94] A. Turhan and E. Zenker. Towards temporal fuzzy query answering on stream-based data. In *HiDeSt@KI*, volume 1447 of *CEUR Workshop Proceedings*, pages 56–69. CEUR-WS.org, 2015.

[95] J. Urbani, S. Kotoulas, J. Maassen, F. van Harmelen, and H. E. Bal. Webpie: A web-scale parallel inference engine using mapreduce. *J. Web Sem.*, 10:59–75, 2012.

[96] J. Urbani, A. Margara, C. J. H. Jacobs, F. van Harmelen, and H. E. Bal. Dynamite: Parallel materialization of dynamic RDF data. In *International Semantic Web Conference (1)*, volume 8218 of *Lecture Notes in Computer Science*, pages 657–672. Springer, 2013.

[97] R. Volz, S. Staab, and B. Motik. Incrementally maintaining materializations of ontologies stored in logic databases. *J. Data Semantics*, 2:1–34, 2005.

[98] O. Walavalkar, A. Joshi, T. Finin, and Y. Yesha. Streaming knowledge bases. In *In International Workshop on Scalable Semantic Web Knowledge Base Systems*, 2008.

[99] R. Yan, M. T. Greaves, W. P. Smith, and D. L. McGuinness. Remembering the important things: Semantic importance in stream reasoning. In *SR+SWIT@ISWC*, volume 1783 of *CEUR Workshop Proceedings*, pages 49–54. CEUR-WS.org, 2016.

[100] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: fault-tolerant streaming computation at scale. In *SOSP*, pages 423–438. ACM, 2013.

[101] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica. Apache spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016.

[102] X. Zhang, G. Cheng, and Y. Qu. Ontology summarization based on rdf sentence graph. In *WWW*, pages 707–716. ACM, 2007.