Contrasting RDF Stream Processing Semantics*

Minh Dao-Tran, Harald Beck, and Thomas Eiter

Institute of Information Systems, Vienna University of Technology
Favoritenstraße 9-11, A-1040 Vienna, Austria
{dao,beck,eiter}@kr.tuwien.ac.at

Abstract. The increasing popularity of RDF Stream Processing (RSP) has led to developments of data models and processing engines which diverge in several aspects, ranging from the representation of RDF streams to semantics. Benchmarking systems such as LSBench, SRBench, CSRBench, and YABench were introduced as attempts to compare different approaches, focusing mainly on the operational aspects. The recent logic-based LARS framework provides a theoretical underpinning to analyze stream processing/reasoning semantics. In this work, we use LARS to compare the semantics of two typical RSP engines, namely C-SPARQL and CQELS, identify conditions when they agree on the output, and discuss situations where they disagree. The findings give insights that might prove to be useful for the RSP community in developing a common core for RSP.

1 Introduction

In interconnected information technologies such as Internet of Things and Cyber Physical Systems it is crucial to have simple access to data irrespective of their sources. The Semantic Web's RDF data model was designed to integrate such distributed and heterogeneous data. Recently, RDF Stream Processing (RSP) has been emerging to tackle novel problems arising from streaming data: to integrate querying and processing of static and dynamic data, e.g., information continuously arriving from sensors.

This has led to the development of data models, query languages and processing engines, which diverge in several aspects, ranging from the representation of RDF streams, execution modes [17], to semantics [2, 16, 5, 4, 12]. To deal with this heterogeneity, the RSP community[1] was formed to establish a standard towards a W3C recommendation.

A standardization must start from seeing the differences between existing approaches and thus comparing RSP engines is an important topic. Initial empirical comparisons were carried out in two benchmarking systems, namely SRBench [19] and LSBench [17]. The former defined functional tests to verify the query languages features by the engines, while the latter measured mismatch between the output of different engines, assuming they are sound, i.e., all output produced by them are correct. Later on, CSRBench [8] introduced an oracle that pregenerates the correct answers wrt. each engine's semantics, which are then used to check the output returned by the engine. YABench [14] follows the approach by CSRBench with the main purpose of facilitating joint evaluation of

* This research has been supported by the Austrian Science Fund (FWF) projects P24090, P26471, and W1255-N23.

[1] https://www.w3.org/community/rsp/

functional, correctness, and performance testing. However, this approach allows only partial comparison between engines by referring to their ideal counterparts.

Due to the lack of a common language to express divergent RSP approaches, the three works above could just look at the output of the engines and did not have further means to explain beyond the output what caused the difference semantically.

Recently, [9] proposed a unifying query model to explain the heterogeneity of RSP systems. It shows the difference between two approaches as represented by representative engines in the RSP community, namely C-SPARQL [2], $\text{SPARQL}_{\text{Stream}}$ [5] and CQELS [16]. This work identified types of datasets that C-SPARQL/$\text{SPARQL}_{\text{Stream}}$ can handle while CQELS cannot, and vice versa. However, it does not point out systematically when and how the engines agree on the output.

In the stream processing community, SECRET [10] was proposed to characterize and analyze the behavior of stream processing engines, but at the operational level.

Latterly, a **L**ogic-based framework for **A**nalyzing **R**easoning over **S**tream (LARS) was introduced [3]. LARS can be used as a unifying language which stream processing/reasoning languages can be translated to. It may serve as a formal host language to express semantics and thus allows a deeper comparison that goes beyond mere looking at the output of the respective engines. Furthermore, the model-based semantics of LARS is a means to formalize the intuition of agreement between not only RSP engines but also engines from other communities, and to identify conditions where this holds.

In this paper, we exploit the capability of LARS to analyze the difference between the semantics of C-SPARQL and CQELS by: (a) providing translations that capture the push- and pull- execution modes for general LARS programs, (b) providing translations from C-SPARQL, CQELS to LARS, (c) introducing a notion of *push-pull-agreement* between LARS programs, and (d) identifying *conditions* where C-SPARQL and CQELS agree on their output, by checking whether the translated LARS programs push-pull-agree.

Due to space reasons, (a) and (b) will briefly mentioned while (c) and (d) will be presented in details. For an extended version of this paper, we refer the reader to [7].

Our findings show that C-SPARQL and CQELS agree on a very limited setting, and give insights on their difference. This result might prove to be useful for the RSP community in developing a common core for RSP.

For the purpose of a theoretical comparison, we adopt as in [9] the assumption that execution time of RSP engines is neglectable compared to the input streams' rate.

2 Preliminaries

From RDF to RDF Stream Processing. RDF [6] is a W3C recommendation for data interchange on the Web models data as directed labeled graphs whose nodes are resources and edges represent relations among them. SPARQL [13], a W3C recommendation for querying RDF graphs, is essentially a graph-matching query language. As the underlying RDF graphs are static, SPARQL's one-shot queries are not able to give answers under dynamic input. RSP was thus introduced to express queries on streaming RDF data.

The two representative RSP engines are C-SPARQL and CQELS. Their query languages and semantics are inspired by the Continuous Query Language (CQL) [1] in which queries are composed of three classes of operators, namely stream-to-relation

	C-SPARQL	CQELS
execution mode	pull-based	push-based
snapshot creation	merge patterns on input streams into the default graph	apply patterns on input streams

Table 1: C-SPARQL vs. CQELS

(S2R), relation-to-relation (R2R), and relation-to-stream (R2S) operators (`RStream`, `IStream`, and `DStream`). The main adaptation for RSP is that R2R operators are SPARQL operators. However, C-SPARQL and CQELS diverge in two crucial aspects as presented in Table 1. This makes it non-trivial to compare the two engines.

To make the comparison possible, we propose a unified model of RSP queries that covers all of the above divergence. Extending the work in [18] for SPARQL queries, we model an RSP query as a quadruple $Q = (V, P, \mathcal{D}, \mathcal{S})$, where V is a result form, P is a graph pattern, $\mathcal{D}$ is a dataset, and $\mathcal{S}$ is a set of input stream patterns. Notably, $\mathcal{S}$ is a set of tuples of the form $(\mathbf{s}, \omega, g)$, where $\mathbf{s}$ is a stream identifier, ω is a window expression, and g is a basic RDF graph pattern. An RSP query Q in this model corresponds to a C-SPARQL query, denoted by $cs(Q)$ and to a set $cq(Q)$ of $2^{|\mathcal{S}|}$ CQELS queries.

Then, we need a common logic-based framework to compare the two engines. This leads to LARS whose flavor is described next.

A Logic-based Framework for Analyzing Reasoning over Streams. LARS [3] was developed from Answer Set Programming [11] to work with streams. Crucial extensions are:

- streams are represented as tuples $S = (T, \upsilon)$ where T is a time line and υ is an evaluation function mapping each time point t to a set of facts. We call S a *data stream*, if it contains only extensional atoms. The *projection* of a stream S to a predicate p is defined as $S|_p = (T, \upsilon|_p)$, where $\upsilon|_p(t) = \{p(\boldsymbol{c}) \mid p(\boldsymbol{c}) \in \upsilon(t)\}$. By $Ats(S) = \bigcup_{t \in T} \upsilon(t)$, we denote the set of all atoms appearing in S.
- window functions are generically defined to capture all types of windows in practice. $w_\iota(S, t, \boldsymbol{x})$ of type ι takes as input a stream S, a time point t, and a vector of parameters $\boldsymbol{x}$ for type ι and returns a substream S' of S;
- window operators of the form $\boxplus^{\boldsymbol{x}}_{\iota,ch}$ are the means to connect LARS formulas to the corresponding window functions $w_\iota(S, t, \boldsymbol{x})$. Window operators can be nested;
- different ways to refer to time in a LARS formula: operators $\Diamond$ and $\Box$ work similarly as in temporal logic, and operator $@_t$ refers to an explicit time point in a time line;
- output of a LARS program P wrt. to a data stream D at a time point t is represented as a set $\mathcal{AS}(P, D, t)$ of answer streams, that is, minimal interpretation streams I satisfying the reduct of P under I at t. As RSP queries return just a single answer at a time point, we consider in this paper LARS programs that have a single answer stream. By $AS(P, D, t)$, we directly refer to the single element of $\mathcal{AS}(P, D, t)$.

The following examples illustrate the idea of LARS:

- $\boxplus^{10}_{\tau} \Diamond tram$: check whether a tram appearance was reported in the last 10 time units.

- $\boxplus_{\tau}^{60}\Box\ \boxplus_{\tau}^{10}\ \Diamond tram$: check whether in the last 60 time units, in every interval of 10 time units, there was a tram appearance reported.
- $@_{T+5}\, expTramM \leftarrow \boxplus_{\tau}^{5} @_T\, tramB$: if a tram appeared at some time T in the last 5 time units at station B, we can expect it to appear at station M at $T+5$.

We next show how C-SPAQRL and CQELS can be translated into LARS.

3 Translating RSP Queries into LARS

We propose the following translations (see details in [7]):

(1) Given a window expression ω in C-SPARQL or CQELS, $\tau(\omega)$ translates it to a window operator of $LARS$.
(2) Given a LARS program P and a pulling period $U > 0$, the translations $\triangleright(P)$ and $\triangleleft(P, U)$ encode the push- and pull- modes by LARS rules, respectively.
(3) Given an RSP query Q, translation τ_1 applies on $cs(Q)$ while translation τ_2 applies on $Q' \in cq(Q)$ and return LARS programs. Both share the core from translation τ from SPARQL to Datalog in [18], and differ due to two approaches by C-SPARQL and CQELS to deal with streaming input. As an extension of τ, static RDF triples are represented as a 4-ary predicate of the form $\texttt{triple}(S, P, O, G)$ while triples arrving at a stream $\mathbf{s}$ at time t contributes to the evaluation function υ at t under a predicate $\texttt{striple}$, that is, $\texttt{striple}(s, p, o, \mathbf{s}) \in \upsilon(t)$. Auto-generated predicates of the form $\texttt{ans}_i$ are used to hold answers of intermediate translation, and $\texttt{ans}_1$ holds the answers of the queries.

So far, it has not been clear under which conditions the two engines will return the same output. Tackling this question now becomes possible at a formal level using LARS.

Given an RDF triple (s, p, o) and a basic graph pattern g, we say (s, p, o) *submatches* g, denoted by $sm(s, p, o, g)$, iff there exists a triple pattern $(S, P, O) \in g$ s.t. $[\![(S, P, O)]\!]_{\{(s,p,o)\}} \neq \emptyset$, where the notion of subgraph matching $[\![.]\!]$ is defined in [15]. Given a graph pattern P, let $trp(P)$ be the set of triple patterns appearing in P.

The following result identifies a class of RSP queries where the answer streams of the translated LARS programs by τ_1 and τ_2 coincide on the output predicate $\texttt{ans}_1$.

Theorem 1 *Let $Q = (V, P, \mathcal{D}, \mathcal{S})$ be an RSP query where $\mathcal{D} = (G, G_n)$ contains a default graph G and a set G_n of named graphs, and $\mathcal{S} = \{(\mathbf{s}_1, \omega_1, g_1), \ldots, (\mathbf{s}_m, \omega_m, g_m)\}$. Let $P_1 = \tau_1(cs(Q))$, $P_2 = \tau_2(Q')$, for any $Q' \in cq(Q)$, D be a data stream, and t be a time point. If*

$$\forall g \neq g' \in \{\{trp(P) \setminus \bigcup g_i\}\} \cup \{g_1, \ldots, g_m\} \colon g \cap g' = \emptyset, \qquad (\star)$$

$$\forall \texttt{striple}(s, p, o, \mathbf{s}_i) \in Ats(D) \colon sm(s, p, o, g_i) \text{ and } \forall g_j \neq g_i \in \mathcal{S} \colon \neg sm(s, p, o, g_j) \text{ and } \neg sm(s, p, o, trp(P) \setminus \bigcup g_j) \qquad (\star\star)$$

then $AS(P_1, D, t)|_{\texttt{ans}_1} = AS(P_2, D, t)|_{\texttt{ans}_1}$.

Condition ($\star$) requires that the graph patterns wrt. the static dataset and the input streams do not share triple patterns while ($\star\star$) makes sure that triples arrived at stream $\mathbf{s}_i$ are not allowed to enter any other stream or to stay in the static dataset. Combining these two conditions intuitively means that all input streams and static dataset have disjoint input. Then, the two approaches in building snapshots correspond as the distinction of input due to stream graph patterns in CQELS also happens for C-SPARQL. Thus, the answer streams produced by two translated LARS programs coincide on the output predicate.

4 RSP Semantics Analysis Based on LARS

In the previous section, (3) presents translations from RSP queries on either C-SPARQL or CQELS branches into LARS programs. Under condition ($\star$) and ($\star\star$) in Theorem 1, the two translated LARS programs from a C-SPARQL and a CQELS queries, rooted from the same RSP query, produce the same output predicate $\texttt{ans}_1$ (thus on `RStream` operator) when they are evaluated at the same time point.

However, C-SPARQL and CQELS are based on two different execution modes: push-based and pull-based, which are captured in (2) for general LARS programs. In order to theoretically analyze and compare the semantics of C-SPARQL and CQELS, we need to combine the above two results, together with taking into account the difference between `IStream` and `RStream` operators. But first of all, we must clarify what we mean by saying "C-SPARQL and CQELS agree on the output."

4.1 Agreement between C-SPARQL and CQELS

We propose a characterization of agreement between C-SPARQL and CQELS using LARS. For the core notion, we concentrate on the agreement on the resulted mappings after non-aggregate SPARQL operators such as AND, UNION, etc. Extending to aggregate will be discussed in Section 5.

Intuitively, the two semantics are considered to agree on a timeline T with a pulling period U, if (1) they both start at the same time point 0, and (2) for every interval $(i \cdot U, (i+1) \cdot U] \in T$, where $i \geq 0$, the union of outputs produced by CQELS in the interval coincides with the output produced by C-SPARQL at the right-end of the interval. To formalize the conditions for agreement, we need the notion of trigger time points and incremental output presented next.

Trigger Time Points. Let $t_1 < t_2$ be two time points. The set of *trigger time points* in a data stream D in the interval $(t_1, t_2]$ is defined as $ttp(t_1, t_2, D) = \{t \in (t_1, t_2] \mid \upsilon_D(t) \neq \emptyset\}$. For a time point $t \in T_D$ such that $t > 0$, the *previous trigger point* of t with respect to D is $prev(t, D) = \max(ttp(0, t-1, D))$ if $ttp(0, t-1, D) \neq \emptyset$ and is 0 otherwise.

Incremental Output. Next, we capture the incremental output strategy, i.e., the `IStream` operator by means of the difference between answer streams of two consecutive trigger time points. Let $I_t = AS(P, D, t)$. Then, the *incremental output* $inc(P, t)$ at a trigger time point t (i.e., $\upsilon_D(t) \neq \emptyset$) is $Ats(I_t \setminus I_{prev(t)})$ if $t > 0$, and $inc(P, 0) = Ats(I_0)$. Here, the difference between two streams $S_1 = (T, \upsilon_1)$ and $S_2 = (T, \upsilon_2)$ is defined as $S' = S_1 \setminus S_2 = (T, \upsilon')$ s.t. for all $t' \in T$, we have that $\upsilon'(t') = \upsilon_1(t') \setminus \upsilon_2(t')$.

Based on this, we define when two LARS programs, executed on push- and pull-modes, *agree* on an interval of time.

Definition 1 *Given two LARS programs P_1, P_2, a data stream $D = (T_D, \upsilon_D)$, and two time points $t_1 < t_2$ of T_D, let $A_1 = \bigcup_{t \in ttp(t_1,t_2,D)} inc(P_1, t) \cup \bigcap_{t \in ttp(t_1,t_2,D) \cup \{t_2\}} Ats(I_t)$ and $A_2 = Ats(AS(P_2, D, t_2))$. Let $R = \{p_1, \ldots, p_n\}$ be a set of predicates. We say P_1 and P_2* push-pull-agree *on D and R*

(i) during the interval $(t_1, t_2]$, *denoted by $P_1 \equiv_{t_1,t_2}^{D,R} P_2$, iff $A_1|_R = A_2|_R$;*

(ii) with pulling period U, *denoted by $P_1 \equiv_{U}^{D,R} P_2$, iff $P_1 \equiv_{t_1,t_2}^{D,R} P_2$, where $t_1, t_2 \in T_D$ such that there exists some $i \in \mathbb{N}$, where $t_1 = i \cdot U$ and $t_2 = (i+1) \cdot U$.*

Intuitively, push-pull-agreement during $(t_1, t_2]$ is established by comparing the answer stream evaluated at t_2 with the union of incremental answer computed at all trigger time points in the interval. The term $\bigcap_{t \in ttp(t_1,t_2,D) \cup \{t_2\}} Ats(I_t)$ ensures that for programs that always produce some output $p(\boldsymbol{c})$ at every time point, this output is also counted in comparing the incremental result and the result at t_2.

4.2 Agreement Conditions

Given an RSP query $Q = (V, P, \mathcal{D}, \mathcal{S})$, let $Q_1 = cs(Q)$ and $Q_2 \in cq(Q)$. We want to identify conditions guaranteeing that the LARS programs $\tau_1(Q_1)$ and $\tau_2(Q_2)$ agree on the output predicate $\texttt{ans}_1$, that is $\tau_2(Q_2) \equiv_{t_1,t_2}^{D,\texttt{ans}_1} \tau_1(Q_1)$.

Let $D = (T_D, \upsilon_D)$ be a data stream. The projection of D on an input stream identified by an IRI $\mathbf{s}$ is $D|_{\mathbf{s}} = (T_D, \upsilon_D|_{\mathbf{s}})$, where for all $t \in T_D$, we have that $\upsilon_D|_{\mathbf{s}}(t) = \{\texttt{striple}(S, P, O, \mathbf{s}) \in \upsilon_D(t)\}$. That is, we keep only facts with $\mathbf{s}$ as the stream identifier. The *snapshot* of D with respect to $\mathcal{S}$ at time point t is defined as:

$$sn(D, \mathcal{S}, t) = \bigcup_{(s,\omega,g) \in \mathcal{S}} w_{\tau(\omega)}(D|_{\mathbf{s}}, t).$$

Intuitively, for each input stream pattern $(\mathbf{s}, \omega, g) \in \mathcal{S}$, we apply the window function $w_{\tau(\omega)} = w_{\boxplus}$ to the projection of D on $\mathbf{s}$. Note that $\tau(\omega)$ translates the window expression ω to a window operator $\boxplus$, and $w_{\boxplus}$ is the window function behind $\boxplus$. The union of all substreams extracted by the window functions give us the snapshot. The following result identifies sufficient conditions where C-SPARQL and CQELS agree.

Theorem 2 *Let $Q = (V, P, \mathcal{D}, \mathcal{S})$ be an RSP query, where P contains neither* MINUS *nor* FILTER NOT EXISTS*, $\mathcal{D} = (G, G_n)$ contains a default graph G and a set G_n of named graphs, and $\mathcal{S} = \{(\mathbf{s}_1, \omega_1, g_1), \ldots, (\mathbf{s}_m, \omega_m, g_m)\}$ contains only time-based windows of the form* `[RANGE L]`*. Let $Q_1 = cs(Q)$, $Q_2 \in cq(Q)$, and $t_1 < t_2$. If ($\star$) and ($\star\star$) hold, and additionally*

$$\bigcup_{t \in ttp(t_1,t_2,D)} sn(D, \mathcal{S}, t) = sn(D, \mathcal{S}, t_2) \qquad (\star\star\star)$$

then

$$\tau_2(Q_2) \equiv_{t_1,t_2}^{D,\texttt{ans}_1} \tau_1(Q_1).$$

This result can be straightforwardly extended to check whether $\tau_1(Q_1) \equiv_{U}^{D,\texttt{ans}_1} \tau_2(Q_2)$, but is omitted due to space reason. The theorem shows that having agreement between C-SPARQL and CQELS is not easy to achieve, as discussed in the next section.

5 Discussion and Conclusion

Theorem 2 identifies sufficient conditions on which C-SPARQL and CQELS agree on their output, including (i) no MINUS or FILTER NOT EXISTS operator, (ii) only time-based windows with sliding size 1, (iii) only "disjoint" patterns and data in the static datasets and the input streams, and (iv) having the same snapshot collected in the interval as at the right end of the interval.

While (i)-(iii) correspond to useful fragments of queries for practical purposes, (iv) cannot be guaranteed in case of high throughput. The reason is that with dense input streams, the snapshots taken at time points near the left end of an interval will have high chances to collect more triples than the snapshot at the right end of the interval. Thus, having C-SPARQL and CQELS agreeing in practice is very unlikely, due to the strong semantic implications of push/pull-based querying. Consequently, data independent agreement conditions are unlikely to be found for queries that go beyond pure SPARQL.

One can easily find a counter example for the agreement when relaxing any of (i)-(iii) and keeping other conditions unchanged. For example, when FILTER NOT EXISTS or MINUS is allowed, the translated LARS programs are not positive. This takes away the monotonic property, i.e., having more input one any side (push- or pull-based) might lead to shrinking the output facts and introducing disagreement on the output.

For sliding windows with sliding size greater than 1, C-SPARQL can produce output that CQELS cannot, even when ($\star\star\star$) is satisfied. Intuitively, this is because the window only slides after a certain amount of time and might miss some most recent input. In this case, we think that pull-based is preferable over push-based execution.

Finally, if the static datasets and the input streams share patterns by which triples are matched for R2R operators, the result of C-SPARQL and CQELS will be different. For instance, if these datasets share the same pattern and the static dataset contains some triples matching this pattern, then C-SPARQL can produce output even when no input triple arrives at the stream, as it cannot distinguish where a triple comes from. Besides, the stream graph pattern of CQELS has no mapping due to empty input, and thus produces no output. However, this situation should not happen often in practice as merged input streams will usually be distinguishable by an implicit schema.

The core notion of agreement does not consider aggregate. When considering aggregate, we observe that only certain types of aggregate allow for tracing agreement between pull- and push-based executions. For example, for COUNT, we can say that CQELS agrees with C-SPARQL in an interval $(t_1, t_2]$ iff the sum of output values produced by the former during the interval equal to the output value returned by the latter at t_2. Similar extension can be done for MAX, MIN. However, with MEDIAN or AVG, one cannot reproduce the result from CQELS' output to match that from C-SPARQL. In general, we can only give agreement notion for aggregates that can be recursively defined.

Conclusion and Outlook. This paper utilizes LARS to give insights on the contrast between two RSP semantics implemented in two representative engines, namely C-SPARQL and CQELS. Compared to the closest work in [9], we made another important step forward by introducing a notion of agreement between the engines and identifying conditions for it to hold.

The theoretical result is based on the assumption that engine execution time is neglectable to the input rate. For further practical comparison, we envision future work where this condition is dropped. Implementing the proposed translations is also on our agenda. In another direction, we are investigating equivalence for general LARS programs. Once this result is available, one can have an automatic equivalence checker which takes any two translated LARS programs of two continuous queries from any two stream processing languages, tell whether the two original queries are equivalent, and possibly even enumerate their different outputs due to our model-based approach.

References

1. A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB J.*, 15(2):121–142, 2006.
2. 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.
3. H. Beck, M. Dao-Tran, T. Eiter, and M. Fink. LARS: A logic-based framework for analyzing reasoning over streams. In *AAAI*, 2015.
4. A. Bolles, M. Grawunder, and J. Jacobi. Streaming SPARQL - extending SPARQL to process data streams. In *ESWC*, pages 448–462, 2008.
5. J.-P. Calbimonte, Ó. Corcho, and A. J. G. Gray. Enabling ontology-based access to streaming data sources. In *ISWC (1)*, pages 96–111, 2010.
6. R. Cyganiak, D. Wood, and M. Lanthaler. RDF 1.1 Concepts and Abstract Syntax. http://www.w3.org/TR/rdf11-concepts/, 2014.
7. M. Dao-Tran, H. Beck, and T. Eiter. Contrasting RDF Stream Processing Semantics. Technical report, Institut für Informationssysteme, TU Wien, 2015.
8. D. Dell'Aglio, J. Calbimonte, M. Balduini, Ó. Corcho, and E. D. Valle. On Correctness in RDF Stream Processor Benchmarking. In *ISWC 2013*, pages 326–342, 2013.
9. D. Dell'Aglio, E. D. Valle, J.-P. Calbimonte, and O. Corcho. Rsp-ql semantics: a unifying query model to explain heterogeneity of rdf stream processing systems. *IJSWIS*, 10(4), 2015.
10. N. Dindar, N. Tatbul, R. J. Miller, L. M. Haas, and I. Botan. Modeling the execution semantics of stream processing engines with SECRET. *VLDB J.*, 22(4):421–446, 2013.
11. T. Eiter, G. Ianni, and T. Krennwallner. Answer Set Programming: A Primer. In *RW*, pages 40–110, 2009.
12. S. Groppe. *Data Management and Query Processing in Semantic Web Databases*. Springer, 2011.
13. S. Harris and A. Seaborne. SPARQL 1.1 Query Language. http://www.w3.org/TR/sparql11-query/, 2013.
14. M. Kolchin and P. Wetz. Demo: YABench - Yet Another RDF Stream Processing Benchmark. In *RSP Workshop*, 2015.
15. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34:16:1–16:45, September 2009.
16. D. L. 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 *ISWC (1)*, pages 370–388, 2011.
17. D. L. Phuoc, M. Dao-Tran, M.-D. Pham, P. Boncz, T. Eiter, and M. Fink. Linked stream data processing engines: Facts and figures. In *ISWC - ET*, pages 300–312, 2012.
18. A. Polleres. From SPARQL to rules (and back). In *WWW 2007*, pages 787–796, 2007.
19. Y. Zhang, P. Minh Duc, O. Corcho, and J. P. Calbimonte. SRBench: A Streaming RDF/S-PARQL Benchmark. In *ISWC*, pages 641–657, 2012.