Journal Title
XX(X):1–36

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Navigating Streams: A Survey on 15 Years of Advances in Stream Reasoning Formalisms

Mathijs van Noort[1], Pieter Bonte[2] and Femke Ongenae[1]

Abstract

Stream Reasoning (SR) has emerged as a crucial paradigm for enabling real-time, intelligent decision-making over dynamic data streams, which are increasingly prevalent in domains such as Internet-of-Things (IoT), edge computing, and decentralized systems. Formalisms for SR are essential because they define the theoretical and practical foundations for reasoning under continuous, time-sensitive conditions. This paper presents a comprehensive review of SR formalisms introduced over the past 15 years, evaluating their evolution and impact. We propose a set of dimensions for assessing SR formalisms, considering both theoretical properties, such as expressivity, underlying paradigm, and stream representation, as well as practical metrics like citation count, software development, and real-world applications. Through a detailed analysis of the literature since 2009, at which the foundational paper "It's a Streaming World" was published, we score the existing SR formalisms based on these dimensions, highlighting the field's considerable progress. Our findings indicate that recent advancements have led to sophisticated SR formalisms capable of tackling increasingly complex reasoning tasks. However, further research remains essential, as no single formalism satisfies all possible requirements. The best choice depends on the specific needs of the intended application. To support this, we provide a broad overview of currently existing formalisms to help practitioners select the most suitable approach. Formalisms like DatalogMTL and LARS stand out for their strong theoretical foundations and promise for supporting advanced applications, while query-based formalisms show potential for addressing intricate reasoning challenges beyond basic querying. Additionally, SR formalisms that integrate RDF streams are particularly well-positioned to enhance interoperability across heterogeneous systems, opening concrete opportunities for deployment in IoT and edge computing scenarios.

Keywords

Stream Reasoning, Literature Review, Formalism, Temporal Reasoning

1 Introduction

SR refers to the ability to perform logical reasoning over data streams in real time, combining the principles of knowledge representation with continuous data processing. Unlike traditional temporal reasoning, which focuses on reasoning over time-stamped facts in a static dataset, SR addresses the challenges posed by unbounded, rapidly changing streams of information. This distinction is crucial because SR not only considers when events occur but also how to reason efficiently and correctly under strict time constraints. Formal SR approaches enable guarantees regarding correctness, complexity, and optimization, which are essential for building scalable and reliable systems.

In recent years, reasoning over streaming data has rapidly evolved from a niche research topic into a critical capability for modern data-intensive applications. The explosive growth in both the volume and velocity of data is reshaping how organizations operate, with businesses increasingly relying on real-time streams to drive agility, enhance customer experiences, and improve operational efficiency [1]. Unlike traditional batch processing, streaming data arrives continuously and without a predetermined endpoint, making it virtually unbounded [2, 3]. Moreover, its lifecycle is short: new information often invalidates previous observations, creating a dynamic environment where timely reasoning is essential. These characteristics underscore why SR is not just an extension of temporal reasoning but a necessity for today's data-driven systems, and why robust theoretical foundations are needed to ensure correctness, scalability, and optimization in reasoning systems..

An appeal by Della Valle et al. [2] in 2009 highlighted the importance of (continued) research into the area of SR, and subsequently suggests multiple routes into which said research should best proceed. One of said proposed routes is the formalization of the theoretical aspects of SR. A model theory for SR which covers inference methodology, the representation of a stream and of information within a stream had not been thoroughly explored at the time, but greatly facilitates the study of the behaviour and properties of a reasoning system. Since the publication of the paper titled '*It's a Streaming World! Reasoning upon Rapidly Changing Information*' in 2009, the field has seen significant growth

[1] IDLab, Ghent University-imec, BE
[2]Department of Computer Science, KU Leuven Campus KULAK, BE

Corresponding author:
Mathijs van Noort, IDLab, Ghent University-imec, Technologiepark-Zwijnaarde 126, 9052 Ghent, BE
Email: mathijs.vannoort@ugent.be

in both community engagement and research output, as illustrated by Figure 1.

The practical interest in SR often lies not in isolated facts but in reasoning over patterns and relationships across time and multiple streams. For example, consider a smart traffic management system: knowing the location of a single vehicle at one moment is of limited use, but reasoning over the evolving positions of thousands of vehicles allows the system to infer congestion patterns, predict traffic jams, and optimize routing decisions. This requires deductive reasoning over temporal and spatial constraints, which goes beyond detection of isolated events and highlights the added value of SR .

Back in 2009, the lack of formalization of SR's theoretical aspects was cited as a major hurdle to delivering robust solutions for emerging use cases. Formalisms for SR provide a structured foundation for both theoretical exploration and practical implementation, by describing the different conceptual elements and relations between them in a formal, typically mathematical way. They enable guarantees about correctness and complexity, facilitate optimization, and ensure that systems remain reliable under scaling and evolving requirements. This is analogous to relational algebra in databases, which underpins SQL's scalability and optimization [4]. By adhering to formal principles, SR systems can adapt to increasing data volumes and complexity without sacrificing performance or reliability.

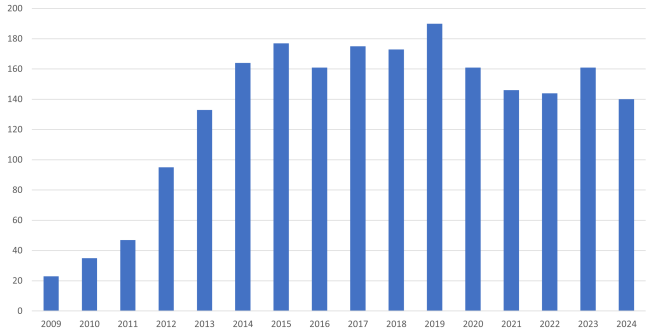

Figure 1. Amount of papers found by Google Scholar on search term 'Stream Reasoning' between 2009 and 2023, grouped by year of publication.

The scope of this survey is limited to deductive SR, which focuses on logic-based reasoning over streams [5]. While SR research also includes areas such as Complex Event Processing (CEP), Streaming Linked Data (SLD), and inductive SR approaches (e.g., machine learning), these are outside the focus of this work. To our knowledge, no prior survey has systematically examined SR formalisms at the level of knowledge representation and reasoning.

In this paper, we summarize the current state of the art in SR formalisms, highlighting their strengths and weaknesses. Furthermore, we review their adoption in real-world applications. Collecting and structuring these contributions provides a reliable gateway for new researchers, showcases the current capabilities of the field, and helps identify gaps for future research.

Our paper is structured as follows. Our approach to this literature review is outlined in Section 2. The necessary background to understand the formalisms covered in this paper is grouped in Section 3. Section 5 details the theoretical & practical usage dimensions that are used to evaluate the different SR formalisms that are subject to our study. The subsequent sections each cover the properties of the different formalisms in function of the described dimensions, where Section 6 covers the theoretical dimensions and Section 7 covers the usage dimensions. Our overall discussion of the outlined formalisms is bundled in Section 8. Lastly, we consider future research directions in in the concluding Section 9.

2 Research Questions and Protocol

To clearly define the scope and methodology of this survey, we first outline the research questions that guide our analysis. These questions establish the focus of our review and ensure that the evaluation of Stream Reasoning formalisms is systematic and objective. Following this, we describe the protocol adopted for conducting the literature review, which provides transparency and reproducibility in our approach.

2.1 Research Questions

To pinpoint the exact scope of this survey, we focus on answering the following questions:

Q1. What are the common constructions and expressions of a formalism for SR?

Q2. What are sensible criteria to evaluate a logical formalism, in particular a formalism's suitability for SR?

Q3. Given the criteria from the previous question, how do recently developed SR formalisms meet these criteria and how do these formalisms compare to one another?

Q4. Among the formalisms that have been developed since '*It's a Streaming World*' [2], which have been adopted in real-life applications?

Q5. What additional research can be done to improve and accelerate adoption of SR in practical use cases?

2.2 Protocol

Inspired by Brereton et al. [6], who discusses how to set up an adequate literature review within the software engineering domain, we abide by a strict protocol for the execution of the intended systematic literature review. This protocol aims to provide structure to our literature search and to minimize subjectivity.

First, we conduct a preliminary study to explore our search engine of choice, i.e. Google Scholar, with regards to various keywords combinations. We selected Google Scholar as our primary source because it provides comprehensive coverage of academic publications across multiple disciplines, including computer science and semantic technologies. Compared to alternatives such as Scopus or Web of Science, Google Scholar offers broader indexing of conference proceedings and preprints, which are highly relevant for SR research [7].

To do so, we identified a total of 14 key papers in the field is SR. This set of papers is constructed by taking the papers that cite 'It's a Streaming World' by Della Valle et al. [2] and selecting the most cited paper from each year from 2010 up until 2021, where the upper bound of 2021 reflects the inception of this survey's research work. These 12 papers are further supplemented by two prominent survey works by barbieri et al. [8] and Falzone et al. [9]. The full list of papers can be consulted in Appendix A.

To determine an appropriate search strategy, we first examined broad primary keywords commonly used to describe reasoning over temporally evolving data, such as "Stream Reasoning" and "Temporal Reasoning". This initial test showed that the broader terms returned an excessively large number of results. In particular, "Temporal Reasoning" yielded approximately 28.000 hits, covering a wide conceptual space that includes static temporal reasoning and temporal description logics rather than reasoning over streams. Even the more targeted terms "Reasoning over time" (∼2.090 results) and "Reasoning about time" (∼7.370 results) still produced literature whose scope extended beyond SR and led to substantial noise in the result set.

To narrow the scope further, we then combined these primary keywords with secondary terms related to formal specification, such as "formalism", "formal framework" or "formal semantics". Examples included queries such as "Stream Reasoning AND formalism". While this greatly reduced the number of results, it also proved too restrictive: key papers were retrieved only when the query was tailored almost exactly to those papers, indicating that such combinations lacked the generality required for a systematic survey and risked missing relevant contributions.

Based on these observations, the keyword that achieved the best balance between scope, size of the result set, and consistency with the key papers was the general term "Stream Reasoning". It was specific enough to avoid the conceptual drift present in broader temporal-reasoning keywords, yet broad enough to retrieve the central literature in the field without relying on overly narrow query formulations.

Keyword	Search results
'Stream Reasoning'	∼ 2.420
'Temporal Reasoning'	∼ 28.900
'Reasoning over time'	∼ 2.090
'Reasoning about time'	∼ 7.370

Table 1. Approximate amount of results per keyword between 2009 and 2024, as retrieved from Google Scholar

To subsequently gather the material relevant to our topic, the following protocol to identify relevant formalisms is followed:

(1) All search results (listed by Google Scholar) for the term "Stream Reasoning" are taken into consideration.

(2) The set of papers is reduced using a fixed set of boundary conditions to ensure that they are both informative and sufficiently recent:

- Given the stimulus to the field of SR provided by Della Valle et al. [2], we consider research articles from the year 2009 up until (and including) 2024.
- Only scientific articles or PhD theses are considered. Other resources, such as slideshows and extended abstracts, are not taking into consideration due to their limited capacity to serve as a stand-alone source of information.
- Search results that are collections of multiple articles, such as conference proceedings and journal editions, are not included as a whole. In case of relevant articles within these collections, we assume these will turn up in the search results individually.

(3) Based on an article's title, keywords, and abstract, it is evaluated in which context(s) the term "Stream Reasoning" is mentioned with respect to its relevance to the defined research questions above. It is consequently either included in our final set of articles or discarded. Papers are discarded when the term "Stream Reasoning" is mentioned only tangentially or in a non-technical context, such as in introductions without formal contributions. This ensures that the final set includes only works that provide substantive insights into SR formalisms or their theoretical foundations, maintaining the relevance and rigor of the review.

(4) Given the set of remaining articles after application of steps (1) - (3), an article cited by one of the selected articles that was not included in the initial search results, is additionally included into the final set if it provides essential underpinning for the citing paper and meets the criteria of steps (2) and (3).

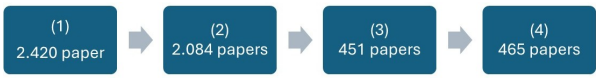

Figure 2. Amount of papers that remains after each step of the defined protocol

The resulting amount of papers throughout each step of the protocol is shown in Figure 2.

Lastly, we reconsider the set of all search results obtained after step (2) to identify systems and applications based on the formalisms selected through our protocol. For each of the papers, we examine whether the paper covers systems or real-life applications that implement or rely on one of the formalisms extracted via steps (1) - (4). This step aims to gauge the degree of adoption and research popularity of the identified formalisms.

3 Preliminaries

Below, we introduce several key concepts from the field of SR. This includes the notion of a data stream and the different ways this can be formally represented. Furthermore, we discuss several techniques to express pieces of data. Most notably, we summarize the most common ways of shaping and expressing temporal information.

3.1 Streams and time

The concept of a 'data stream' plays a central role in SR. Despite this, the concept can be interpreted in multiple different ways, depending on the context and intended purpose. Nonetheless, most works on SR agree on several different factors that stipulate a data stream and associated concepts. We outline this common basis below.

3.1.1 Time Representation Throughout the SR literature, a 'data stream' is unanimously understood as a collection of data which arrives sequentially over a period of time [2, 5, 17]. The sequential nature implies a certain order of the data. Said order can be imposed directly on the data itself, i.e. pieces of data A, B and C arrive back-to-back in sequence (A, B, C). Alternatively, time can be made explicit via the use of a *time domain*; an ordered set of numbers used to represent different points in time. The arrival of pieces of data is subsequently indicated using time points drawn from the time domain. Via the order of the time domain, an ordering of the (arrival of) the pieces of data follows.

Typical choices for a *time domain* include the sets $\mathbb{N}, \mathbb{Q}$ and $\mathbb{R}$, along with their natural order $\leq$. Discrete domains, such as $\mathbb{N}$ and $\mathbb{Z}$, can be considered more 'manageable', as the amount of time points is countable and, in case of bounded intervals over the domain, finite. Comparatively, dense domains, such as $\mathbb{Q}$ and $\mathbb{R}$, may better represent the continuous flow of time. Such a representation, with infinite time points over every considered interval, may however greatly increase the complexity of a reasoning system.

3.1.2 Representation of a stream As considered, a data stream encompasses a collection of data which arrives sequentially over a period of time. Time can be modelled as a *time domain* T in several different ways. We cover the most prevalent ways below;

First, given the 'collection' aspect of a data stream, one can opt to represent a data stream S as a timestamped set (or bag) of data points, for example $S = \{(t, \alpha)\}$, where $t \in T$ and α denotes a piece of information. Alternatively, to incorporate the sequential aspect directly, a stream can be represented as an ordered sequence, such as$S = (\alpha_1, \alpha_2, \ldots)$, with α_i pieces of information. A third perspective views a stream as a function that maps time points from a time domain T to the data arriving at those points, expressed as $S : T \to \Omega$ with $S(t) = \alpha$.

We denote each such piece of information by α and the corresponding universe of all possible pieces by Ω. Throughout this paper, we refer to these elemental pieces of information as *atoms*, with Ω representing the set of all possible atoms. These representations illustrate the flexibility in modeling streams within SR. While a formalism does not need to adopt one of these forms exactly, most approaches adhere to one of these underlying principles. For clarity, we distinguish four broad categories of stream representation:

- timestamped collections (e.g. sets)
- ordered sequences without explicit timestamps
- functions mapping time to atoms that holds true at said time
- other representations that deviate from the above

3.1.3 Windowing In real-time scenarios where SR is deployed, there is typically no knowledge of when a data stream will deliver its final element and terminate. Consequently, streams are often treated as theoretically infinite. This infinite nature not only conflicts with the finite storage resources of reasoning engines, but also with the practical need to produce timely answers rather than waiting indefinitely for all data to arrive. To resolve these challenges, engines may partition streams into finite segments called windows, a process known as *windowing* [2]. A window represents a manageable subset of the stream, enabling reasoning over recent or relevant data. The most common windowing strategies are *time-based* and *tuple-based* windows, which slide over the stream to extract sub-streams. Time-based windows select data within a specific time interval, while tuple-based windows select a fixed number of elements. Another way to characterize windows is by how they progress over the stream rather than by what they cover. *Sliding* windows advance continuously, allowing overlapping segments, whereas *tumbling* windows move in discrete steps, producing consecutive non-overlapping segments. By choosing an appropriate windowing strategy, reasoning engines can process data incrementally, ensuring timely results while still considering enough context for accurate reasoning. For illustration, consider a time-based sliding window of 10 seconds that updates every second, or a tumbling window that processes events in discrete 5-second blocks. Both strategies enable reasoning engines to deliver incremental results without waiting for the entire stream to finish.

3.2 Static relations

As SR is a field of study with a wide range of application areas, the exact information that is carried by the streams can vary greatly. RDF triples and predicate expressions are only two types of information that can be carried by a stream. We discuss several elemental structures and relations of these pieces of information commonly used within the SR environment. Here, our focus is on relations that do not consider a temporal dimension, which we refer to as *static relations*. This in contrast to temporal relations, discussed in Section 3.3.

3.2.1 First-Order Logic, Description Logics and Public Announcement Logic First-Order Logic (FOL) [22] represents information as logical expressions, often called propositions, which evaluate to either true or false, making it a Boolean logic. FOL uses several logical connectives to express relations between pieces of information:

1. conjunction $\wedge$ expresses the concurrence of two pieces of information, i.e. they holds simultaneously.
2. negation $\neg$ indicates the contrary of a given fact, i.e. when expression ϕ is true, $\neg\phi$ is false and vice versa.
3. disjunction $\vee$ express that one or both out of two facts hold.
4. $\phi \to \psi$, the implication, specifies that when ϕ holds, so does ψ. This construction is often employed to express

a rule, stating that when certain conditions (the rule body) are met, the conclusion (the rule head) holds.

5. $\forall$, the universal quantification, conveys a considered expression ϕ holds for all possible values for a specified variable x.

6. $\exists$, the existential quantification, denotes the expression ϕ holds for at least one (yet unspecified) instance for the given variable x.

When formulae are constructed using only the first four of the above operators, the logic is referred to as propositional logic. Its ability to express information on a abstract level has led to FOL being used in multiple different reasoning systems.

When formulae use only the first four operators, the logic is called propositional logic. Its abstract expressivity has made FOL widely used in reasoning systems. Furthermore, FOL includes equivalences such as:

$$\phi \wedge \psi = \neg\big(\neg\phi \vee \neg\psi\big) \qquad \forall x\phi = \neg\big(\exists x \neg\phi\big)$$
$$\phi \vee \psi = \neg\big(\neg\phi \wedge \neg\psi\big) \qquad \exists x\phi = \neg\big(\forall x \neg\phi\big)$$
$$\phi \rightarrow \psi = \neg\phi \vee \psi$$

These equivalences show that some operators can be omitted without losing expressivity. For example, $\phi \vee \psi$ can be consistently replaced by $\neg(\neg\phi \wedge \neg\psi)$ to eliminate every occurrence of the $\vee$ operator. Likewise, it is possible to omit $\rightarrow$, one of the two operators $\wedge$ and $\vee$ and one of the quantifiers $\forall$ and $\exists$ without loss of expressivity. For example, a logic that only considers operators $\neg, \wedge, \exists$ is able to construct a "semantic equivalent" for each formula constructed using all of the above FOL operators.

Full FOL is undecidable for logical consequence derivation, so fragments are often used to ensure decidability. Description Logics (DL) are such fragments, designed to balance expressivity and computational tractability. DL underpins many Semantic Web (SW) technologies [23, 24] and divides knowledge into two parts: the Terminological Box (TBox) (terminological knowledge, e.g., class hierarchies and constraints) and the Assertional Box (ABox) (assertional knowledge, e.g., individuals and their properties). DL-Lite is a lightweight member of this family, limiting the syntax for optimized for efficient query answering while maintaining decidability [25], making it particularly relevant for SR applications. Epistemic logic is a form of modal logic focused on reasoning about knowledge and belief in multi-agent systems. Public Announcement Logic (PAL) [26] builds on epistemic logic by modeling how knowledge changes after truthful announcements. For example, $K_a\phi$ means "agent a knows ϕ." Introspection ($K_a\phi \rightarrow K_a(K_a\phi)$) means if an agent knows something, it also knows that it knows it. Truthfulness ($K_a\phi \rightarrow \phi$) means agents cannot know something false. PAL adds $[!\phi]\psi$, meaning "after announcing ϕ, ψ holds," which updates the knowledge state by removing worlds where ϕ is false.

3.2.2 Set theory Instead of writing a single logical expression such as for FOL, one can group elements sharing a property into sets. For example, if C is the set of all sensors reporting temperature and D is the set of sensors reporting humidity, then:

- union $C \cup D$ = sensors reporting temperature or humidity,
- intersection $C \cap D$ = sensors reporting both,
- $\bar{C}$ sensors not reporting temperature,
- $C \subseteq D$ = all temperature sensors also report humidity.

This clarification specifies what C and D contain. These operators resemble logical connectives ($\wedge, \vee, \neg, \rightarrow$), but their semantics differ across formalisms.

3.2.3 Answer Set Programming (ASP) Answer Set Programming (ASP) is a declarative problem-solving paradigm rooted in non-monotonic logic programming. Unlike traditional logic programming, which typically relies on procedural interpretation of rules, ASP adopts a model-theoretic approach where solutions correspond to stable models (also called answer sets) of a logic program. This paradigm was introduced by Marek and Truszczyński [27] as an alternative to classical logic programming, emphasizing reasoning under incomplete or evolving information.

An ASP program consists of a finite set of rules of the form:

$$a \leftarrow b_1, \cdots, b_m \text{not} c_1, \cdots, \text{not} c_n$$

Here, a, b_i, c_j are atoms, and "not" denotes negation-as-failure, a key feature of non-monotonic reasoning. Intuitively, the rule states that a is true if all b_i are true and none of the c_j can be proven true. This differs from classical negation ($\neg$), as "not" reflects the absence of evidence rather than explicit falsity. ASP programs can express defaults, exceptions, and constraints naturally, making them suitable for complex reasoning tasks.

The semantics of ASP is based on the idea of finding self-consistent sets of beliefs. Given a set of facts as a candidate solution, to check if this set is acceptable, you ask: "If I assume these facts are true, do they justify themselves according to the rules?" If the answer is yes, and no smaller set would work, then this set is a stable model. In other words, an answer set is a collection of facts that does not contradict the program and does not rely on unsupported assumptions. This intuitive notion replaces the need for procedural execution with a declarative interpretation: the program describes constraints, and the solver finds all consistent worlds that satisfy them. ASP supports expressive constructs such as choice rules, constraints, and aggregates, enabling concise modeling of combinatorial problems. For example, a scheduling problem can be encoded by rules that generate candidate assignments and constraints that eliminate invalid ones. The solutions correspond to answer sets satisfying all constraints. Unlike traditional inference, ASP does not aim to derive all logical consequences; instead, it computes complete models that represent possible worlds consistent with the program. The theoretical foundation of ASP lies in Here-and-There Logic (HT) and Equilibrium Logic (EL), which generalize classical

logic to accommodate non-monotonic reasoning [28]. HT introduces a three-valued interpretation (here, there, and classical truth), while Equilibrium Logic characterizes stable models as equilibrium points in this extended semantic space. These logics provide a rigorous basis for ASP's semantics and guarantee properties such as consistency and minimality. ASP has become a powerful tool for knowledge representation and reasoning, particularly in domains requiring default reasoning, planning, and decision-making under uncertainty. Its declarative nature, combined with efficient solvers like Clingo [29], makes it practical for real-world applications, including stream reasoning scenarios where evolving data and constraints must be handled dynamically. For a more in-depth discussion of ASP, we refer to the primer by Eiter et al. [28]

3.3 Expressing temporal relations

The section above considers the truthfulness of information in a static context, i.e. a piece of information, e.g. "His name is John", is assumed to be true, or it is not. Information within a SR environment, however, often has a temporal dimension tied to it, i.e. a measurement is measured at a certain time, a person's address can change or a traffic light can change colour. As a result, the veracity of a piece of information is in part dependent on the moment it is considered.

Based on this 'temporal dimension' of the data, we consider two types of reasoning over data streams. On the one hand, a system can be oblivious to temporal connections between the data in the stream. These systems 'collect' a stream's data as it arrives, but do not take into account whether a data piece ϕ arrived before or after data piece ψ. We say these systems perform *reasoning over time*. On the other hand, there can be systems that do take into account the temporal dimension of the data by observing that data piece ψ arrived after data piece ϕ, a system can perhaps infer valuable (new) information, e.g. $(\psi \text{ after } \phi) \rightarrow \chi$. For these types of systems, we say the system performs *reasoning about time*.

When information can be true at a given point in time, several formulations follow intuitively, i.e. some pieces of information hold regardless of the point in time, other pieces hold over only a limited time interval. A system that wishes to incorporate these type of temporal expressions can do so in different manners.

As discussed in Section 3.2, FOL provides a foundation for expressing static relations but lacks constructs for time-dependent truth. To overcome this limitation, temporal logics extend FOL with operators that explicitly capture temporal aspects of data. Linear Temporal Logic (LTL), introduced by Pnueli [30], is one such extension. As a temporal logic, LTL has several additional operators over FOL, summarized in Table 2 and Table 3 for future and past temporal operators respectively. In short, $\bigcirc\phi$ and $\bullet\phi$ express that ϕ holds at the next, respectively the previous, point in time. ϕ **until** ψ states ϕ holds until ψ holds, i.e., it stops holding as soon as ψ appears, likewise for ϕ **since** ψ. $\Diamond\!\!\!\!+\phi$ and $\boxplus\phi$ express that ϕ holds at some point, respectively at all points in time, in the future. $\Diamond\!\!\!\!-$ and $\boxminus$ are their mirror operators towards the past.

In short, $\bigcirc\phi$ and $\bullet\phi$ express ϕ holds at the next, respectively the previous, point in time. ϕ **until** ψ states ϕ holds until ψ holds, i.e. it stops holding as soon as ϕ appears, likewise for ϕ **since** ψ. $\Diamond\!\!\!\!+\phi$ and $\boxplus\phi$ express that ϕ holds at some point, respectively at all points in time, in the future. $\Diamond\!\!\!\!-$ and $\boxminus$ are their mirror operators towards the past. Alternatively to $\boxplus, \boxminus, \Diamond\!\!\!\!+, \Diamond\!\!\!\!-$, where 'past' and 'future' are relative to a single time point, some formalisms employ an absolute version, $\Box$ and $\Diamond$, that hold over the entire timeline, rather than merely the past and/or future, i.e $\Diamond\phi$ means that ϕ is true at some point.

Symbol	Name
$\bigcirc$	'next' operator
until	'until' operator
$\boxplus$	'always in the future' operator
$\Diamond\!\!\!\!+$	'once in the future' operator

Table 2. LTL future temporal operators

Symbol	Name
$\bullet$	'previous' operator
since	'since' operator
$\boxminus$	'always in the past' operator
$\Diamond\!\!\!\!-$	'once in the past' operator

Table 3. LTL past temporal operators

Some information holds true over a specific interval over the time domain T. To capture this, Metric Temporal Logic (MTL) [31] extends LTL by annotating its temporal operators with explicit time bounds. In LTL, operators such as "eventually" ($\Diamond\!\!\!\!+, \Diamond\!\!\!\!-$), "always" ($\boxplus, \boxminus$), **since** and **until** express properties relative to an unbounded future or past. MTL refines these by introducing intervals $[a, b]$, allowing statements like $\Diamond\!\!\!\!+_{[a,b]}\phi$, stating "$\phi$ will holds at some point between a and b time units in the future", or $\boxminus_{[a,b]}\psi$, i.e. "ψ held at every time point between a and b time units in the past". Similarly, $\phi\mathbf{until}_{[a,b]}\psi$ requires that ψ becomes true somewhere between a and b time units in the future, while ψ stays true until ψ holds true.

This addition of metric bounds transforms LTL's qualitative reasoning into quantitative reasoning, making MTL suitable for real-time systems with strict timing constraints. For example, $\boxplus_{[0,10]}$(`temperature` <80) states that a given temperature remains below 80°C for the next 10 seconds, while $\Diamond\!\!\!\!+_{[2,5]}$`alarm` expresses an alarm will trigger within 2 to 5 seconds. By enabling precise timing specifications, MTL provides the foundation for monitoring, verification, and scheduling in time-critical applications.

Lastly, several logic-based temporal frameworks consider another temporal operator, i.e. the "holds at" operators $@_t$, which denotes a subsequent expression holds at a given time point t. For example, $@_t$ states that the formula ϕ is true specifically at time $t = 5$, regardless of the current evaluation point. This operator is useful for expressing absolute time references rather than relative ones, enabling comparisons across different timestamps or anchoring conditions to specific moments in time.

This operator makes an appearance in frameworks such as the Logic-based framework for Analytic Reasoning over Streams (LARS) [18], where it allows rules to refer explicitly to time points within a stream, and in STARQL [32], which supports timestamp-based bindings in Resource Description Framework (RDF) stream queries. Variants of this concept also occur in Metric LARS [33] and other timed extensions

of ASP, as well as in event-based languages like ETALIS Language for Events (ELE) [34], where explicit time anchoring is essential for complex event detection. By providing a way to pin formulas to exact time instants, $@_t$ complements interval-based operators like those in MTL, offering finer-grained temporal control in reasoning and querying tasks.

Note that the LTL and MTL operators both employ a notion of 'past' and 'future'. This indicates the notion of a certain 'present' t, meaning the operators are to be taken relative to this present. The intricacies of this relativity depend on the semantics deployed by the formalism.

3.3.1 Signal Temporal Logic (STL) Signal Temporal Logic (STL) is a formalism specifically designed for systems where behaviour is described by continuous signals rather than discrete events. It was introduced by Maler and Nickovic [35] as an extension of Metric Interval Temporal Logic (MITL), combining closed-interval temporal operators with real-valued signal predicates. STL formulas are evaluated over signals $s : \mathcal{R}_{\geq 0} \to \mathcal{R}^n$, mapping time to an n-dimensional vector of real values. Atomic predicates typically express constraints on these signals, such as $s_i(t) > c$, where $s_i(t)$ denotes the value of the i-th component at time t and c is a constant. These predicates allow STL to capture quantitative properties of dynamic systems, such as temperature thresholds or velocity bounds.

STL covers only future temporal operators, including $\mathbf{until}_I$, "always" ($\boxplus_I$), and "eventually" ($\oplus_I$), each parameterized by a time interval I similarly to MTL considered above. Analogously to MTL, this boundedness is crucial for monitoring and verification, as it ensures that satisfaction can be checked over finite traces rather than requiring infinite horizons.

A distinctive feature of STL is its robust semantics, which assigns a real-valued robustness score to each formula instead of a simple Boolean truth value. This score measures how strongly a signal satisfies or violates a specification. For instance, if a predicate requires $s_i(t) > 10$ and the actual value is 15, the robustness is positive and proportional to the margin (here, 5). Conversely, if the value is 8, the robustness is negative, indicating violation. Robust semantics enable quantitative reasoning, sensitivity analysis, and optimization, which are essential in control and verification tasks.

3.4 Datalog

Datalog is a declarative query and rule-based language rooted in FOL, widely used for knowledge representation and reasoning. At its core, Datalog expresses information through *facts* and *rules*. Facts represent atomic statements about the domain, such as `parent(alice,bob)`, while rules define logical implications of the form:

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z),
ancestor(Z,Y).
```

Here, the head of the rule (`ancestor(X,Y)`) is inferred whenever the body conditions hold. Variables in rules are universally quantified, and safety conditions require that all variables in the head appear in the body.

Unlike full FOL, Datalog omits function symbols, ensuring that its Herbrand universe is finite and that reasoning remains decidable. This restriction makes Datalog suitable for efficient query evaluation while still supporting recursion, which enables expressing transitive relationships such as reachability or hierarchical dependencies. The semantics of Datalog is based on the notion of *minimal models* and is often computed via a fixpoint procedure: starting from the known facts, rules are applied iteratively until no new facts can be derived.

Datalog serves as the foundation for many reasoning systems and stream reasoning formalisms, where its rule-based structure is extended with temporal operators or windowing constructs. Its declarative nature allows users to specify *what* should hold rather than *how* to compute it, making it a natural choice for expressing complex queries and constraints in dynamic environments. For a comprehensive introduction to Datalog and its theoretical underpinnings, we refer the reader to Abiteboul et al. [36].

3.5 Resource Description Framework (RDF)

RDF is a standard model for representing information on the Web in a structured, machine-readable way. At its core, RDF expresses data as a set of triples, each consisting of a subject, predicate, and object, which together form statements about resources. These resources are identified using Internationalized Resource Identifiers (IRIs), ensuring global uniqueness and interoperability. RDF graphs, which are collections of such triples, provide a flexible and extensible way to describe relationships between entities. This graph-based representation enables integration of heterogeneous data sources and supports reasoning over linked data. The formal concepts underlying RDF, including IRIs, literals, blank nodes, and the notion of an RDF graph, are defined in the W3C RDF 1.2 Concepts and Abstract Syntax specification [37].

Beyond its abstract syntax, RDF is grounded in a formal semantics that defines the meaning of RDF graphs and ensures consistency across implementations. RDF semantics introduces mappings, referred to as *interpretations*, from IRIs and literals to elements in a domain This formal foundation guarantees that RDF-based systems can reason correctly and predictably, supporting tasks such as ontology alignment and data integration. The details of these semantic conditions and entailment rules are provided in the W3C RDF 1.2 Semantics [23]

3.6 Querying

Querying is a fundamental operation in reasoning systems, as it allows users to retrieve relevant information from a knowledge base or a data stream. In the context of SR, queries are typically evaluated over dynamic, time-sensitive data rather than static datasets, which introduces additional complexity. Before discussing specific stream querying formalisms, we outline the basic concepts underlying query languages. A query can be understood as a declarative specification of the information a user wishes to obtain. Rather than describing how to compute the result procedurally, a query expresses the desired pattern or condition that the data must satisfy. For example, in a relational setting, a query might ask for all tuples where a certain attribute equals a given value. In logic-based settings,

queries often take the form of a formula or rule whose satisfaction determines whether a candidate answer is valid.

Queries typically involve *variables*, which act as placeholders for unknown values. When a query is evaluated, these variables are assigned concrete values drawn from the underlying domain, a process known as variable binding. For instance, in a query `?x hasTemperature ?t`, the variables `?x` and `?t` will be bound to specific entities and temperature values that satisfy the stated condition. The set of all such bindings that make the query true constitutes the *answer set* of the query. In stream querying, bindings are not computed over a static dataset but over a windowed view of the stream, as introduced in Section 3.1. This means that the query engine continuously evaluates the query against the most recent subset of the stream, producing updated bindings as new data arrives and old data expires. This incremental evaluation is essential for real-time applications, where answers must reflect the current state of the stream rather than its historical entirety. Query languages differ in their expressivity and underlying data model. For example, SPARQL [38] operates over RDF graphs and supports graph pattern matching, while Datalog-based languages use rule-based syntax to express recursive queries. In case of the latter, a query can be expressed in the form of rule `Head :- Body`, where where the head represents the query goal and the body specifies conditions that must hold for the goal to be satisfied. For example,

```
HighTemp(x) :- Sensor(x),
               Temperature(x,t),
               t > 30.
```

retrieves all sensors `x` reporting a temperature above 30°C.

Stream-oriented extensions such as C-SPARQL [39] and CQELS [40] augment these languages with temporal constructs, enabling queries like "retrieve all sensors reporting a temperature above 30°C in the last 10 seconds." These extensions often introduce operators for specifying windows, ordering, and temporal joins, which are crucial for reasoning about evolving data. Finally, the semantics of query evaluation in a streaming context can vary. Some formalisms adopt snapshot semantics, where the query is evaluated over the current window as if it were a static dataset. Others employ continuous semantics, where the query is treated as a standing request that produces new answers whenever the underlying stream changes. Understanding these distinctions is key to comparing stream querying formalisms, as they impact both performance and correctness guarantees.

4 Related Surveys

Surveying the landscape of SR has been the focus of several prior efforts, each emphasizing different aspects of the field. In this section, we briefly review existing surveys and related work, and position our contribution within this context.

The cooperative work "Grounding Stream Reasoning research" [5] summarizes the field of SR, more specifically the results and discussion of the yearly Streaming Reasoning workshops since 2015. It presents a breakdown of the SR field into four areas of research: (i) Stream Processing, (ii) Streaming Linked Data (SLD), (iii) deductive Stream Reasoning, and (iv) inductive Stream Reasoning. For each of these, Bonte et al. [5] provides a high-level overview of the field, but does not provide a detailed study of specific existing frameworks and their properties.

Several recent surveys do provide a more in-depth analysis of the current state-of-the-art in some subfields of SR. For example, Bonte et al. [41] surveys systems enabling RDF Stream Processing (RSP). It thereby focuses most strongly on the querying aspects of the field. They categorize the current RDF Stream Processing (RSP) systems based on their technical properties, e.g system architecture, operational semantics and accepted data formats. In this work, the field of SR is positioned as an extension of the field of RSP, by supplementing RSP systems with advanced reasoning capabilities. However, this extension is not discussed in depth, nor are the available formalisms surveyed. The earlier work of Bonte et al. [42] likewise covers aspects of RSP, more precisely the key ontologies used in RSP applications. They provide an overview of the modelling and knowledge representation properties of the considered ontologies, but similar to Bonte et al. [41], they do not cover the aptitude and ability for advanced reasoning of the considered systems. In 2019, Isah et al. [43] composed a comparison on existing open-source and commercial stream processing systems. Central in this comparison is the architectural composition and data flow models. From a practical perspective, this work arguments the inherent complexity of streaming data and the need for advanced reasoning capabilities, but does not focus on the underpinning formalisms or their properties.

Isah et al. [43], Bonte et al. [42] and Bonte et al. [41] each focus on different aspects of the SR field, but mainly fit within areas (i) and (ii) outlined by Bonte et al. [5]. As a result, the current literature on areas (iii) and (iv) remains dispersed across the field's growing range of academic works. With this work, we aim to provide a reference work for the research area of deductive SR. We consider the coverage of inductive SR to be beyond the scope of this work as despite their linguistic similarity, deductive and inductive SR employ vastly different techniques and tackle inherently different problems.

Lastly, several earlier works have contributed to a proper outline and definition of the SR field, and thereby to the scope of this survey. Both Margara et al. [14] and Dell'Aglio et al. [17] have identified requirements of SR systems. In particular, Dell'Aglio et al.[17] points out the need for more expressive frameworks, in order to better encode user needs. They stressed the emerging interest for ASP temporal logics for streaming data, as well as several other promising avenues such as Temporal Action Logic (TAL) and Event Calculus. As will be covered in Sections 6 and 7, these avenues were successfully explored and led to promising new SR formalisms such as LARS, Event Calculus for Run-Time reasoning (RTEC), DatalogMTL and others.

While Dell'Aglio et al. focus on system-level requirements such as scalability, timeliness, heterogeneity, and robustness, our survey evaluates SR formalisms along theoretical and usage dimensions. Their requirements define what SR systems must achieve operationally, whereas our

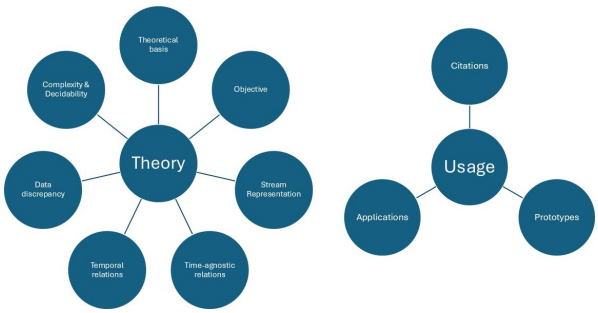

Figure 3. Overview of evaluation dimensions

dimensions study the semantic foundations, expressivity, complexity, and adoption of the underlying formalisms in a more formal manner. Together, these perspectives are complementary: system requirements ensure practical feasibility, while formalism dimensions guarantee the conceptual rigor needed to support those capabilities.

5 Dimensions of a SR formalism

Within this work, we focus on the formalisms that provide syntax and semantics for streaming data. In order to compare multiple different SR formalisms in a consistent manner, we outline 10 dimensions upon which to evaluate the formalisms: 7 dimensions focused on the theoretical properties and 3 dimensions capturing a formalism's usage and adoption.

These 7 theoretical dimensions were derived by analyzing recurring design choices and challenges across the surveyed literature. Specifically, they originate from common aspects that repeatedly appear in SR formalism proposals: (i) the underlying mathematical foundation, (ii) representation of streams and time, (iii) expressivity for static relations, (iv) expressivity for temporal relations, (v) handling of uncertainty and incomplete data, (vi) the intended reasoning objective (querying vs. reasoning), and (vii) computational complexity and decidability. Each of these aspects was identified during the preliminary review of key papers and refined to ensure coverage of both conceptual and practical concerns.

Related surveys such as Bonte et al. [41], Margara et al. [14] and Dell'Aglio et al. [17] touch upon several of these aspects, such as temporal operators ([14, 17, 41]), mathematical foundation ([41]), stream representation ([41]), and handling of uncertainty ([14, 17, 41]), but given their different scope, their treatment is less formal and granular; these works emphasize system architectures and application-level considerations rather than the theoretical underpinnings of SR formalisms (see Section 4). By contrast, our dimensions aim to provide a rigorous framework for analyzing the conceptual properties of SR formalisms alongside their practical adoption. This distinction motivates the depth and structure of our evaluation approach.

To ensure transparency and traceability, each dimension is explicitly linked to the comparative tables in Section 6 and 7. For example, Table 4 summarizes T1 (Theoretical Basis) and T3 (Static Expressivity), Table 5 addresses T4 (Temporal Expressivity), Table 6 covers T2 (Stream Representation and Windowing), and Table 7 captures T5 (Uncertainty Handling). Dimensions T6 (Objective) and T7 (Complexity) are discussed in the narrative of Section 7, while the usage dimensions U1-U3 are reflected in Table 9. These cross-references allow readers to easily navigate from the conceptual definition of a dimension to its empirical evaluation across formalisms. Together, these dimensions serve as different axes for our analysis of the various formalisms. They allow us to:

- Position newly developed formalisms relative to the state of the art.
- Identify the most suitable formalism for a given application.
- Compare multiple formalisms for specific use cases.
- Expose gaps in current SR research and opportunities for extension.

These objectives directly address the research questions introduced in Section 2.1. In particular, the theoretical dimensions in Section 6 operationalize **Q1** by revealing common constructions and expressions across formalisms, and they address **Q2** by providing sensible criteria for evaluating a formalism's suitability for SR. Furthermore, by applying these criteria to each formalism and comparing them across categories, the theoretical dimensions also contribute to answering **Q3**. The usage dimensions of Section 7 complement this by answering **Q4** and **Q5**, focusing on adoption in real-life applications and identifying directions for future research.

5.1 Structure of the Discussion

To structure our discussion of SR formalisms according to these dimensions, we adopt a two-tiered approach that distinguishes theoretical aspects from practical considerations. First, in Section 6, we address the seven theoretical dimensions, which capture the conceptual foundations and formal properties of each formalism. Next, in Section 7, we turn to the three usage dimensions, which reflect adoption and applicability in real-world contexts. This separation provides a clear distinction between what a formalism can express and how it is used in practice.

Within both sections, formalisms are grouped into five categories based on observed characteristics in the surveyed literature: (a) DL-based formalisms, (b) Temporal Logic-based formalisms, (c) SR query languages, (d) ASP-based formalisms, and (e) Miscellaneous formalisms. These categories emerged from the systematic protocol described in Section 2.2 and allow for a structured comparison across families of approaches rather than isolated formalisms.

In Section 6, we discuss each formalism in detail within its category, evaluating it against the seven theoretical dimensions T1–T7. This enables readers to consult the characteristics of a specific formalism while preserving comparability across related approaches. In Section 7, we summarize how the formalisms in each category score on the three usage dimensions U1–U3, highlighting patterns of adoption and maturity. Complementing the narrative, comparative tables provide a concise overview of theory dimensions (Tables 4–7) and

usage dimensions (Table 9), enabling two complementary modes of analysis: direct comparison across formalisms and in-depth examination of individual characteristics. This structure ensures transparency, facilitates navigation, and supports both high-level synthesis and detailed evaluation.

5.2 Theoretical Dimensions

The theoretical dimensions capture the conceptual and formal properties that define the essence of a SR formalism. They allow us to assess the underlying logic or mathematical foundation, the way streams and time are represented, and the expressivity offered for both static and temporal relations. In addition, these dimensions consider how a formalism addresses practical challenges such as data discrepancies, its intended reasoning objectives, and its computational characteristics in terms of complexity and decidability. Together, these aspects provide a structured lens for comparing formalisms beyond surface syntax, focusing on their ability to model dynamic environments rigorously and efficiently.

T1. Theoretical basis A formalism can be rooted in different mathematical theories. Some formalisms are rooted in mathematical logic, whereas others might stem from set theory, ASP, algebraic structures or other. A SR formalism can be developed as an extension of a existing formalism for static data. Alternatively, a SR formalism can be developed 'from scratch', i.e. no existing language or formalism for static data is taken as a basis, but rather an envisioned system or real-life application is modelled directly.

T2. Time and Stream representation As introduced in Section 3.1, there are multiple suitable ways of modeling the concept of a data stream. Understanding these differences is crucial because the chosen representation directly influences the semantics of reasoning, the complexity of algorithms, and the applicability of a formalism to real-world scenarios. For example, whether time is modeled explicitly or implicitly determines how temporal operators behave, while the choice of stream structure (set, sequence, or function) affects how incremental updates and windowing are implemented. These design choices ultimately impact expressivity, performance, and interoperability between systems. Within this dimension, we identify the structure of a data stream within the considered formalism. To describe said structure, we consider the following elements:

- the choice of mathematical construction, i.e. set, collection, function or other, as introduced in Section 3
- How is time represented, e.g. explicitly via time points or implicitly by (chronological) ordering of the elements?
- What are the atoms α that populate the stream?
- If applicable, over what time domains does the formalism consider the streams? What restrictions are imposed upon an eligible time domain?
- Whether or not the stream representation allows for (theoretically) infinite data streams or in case of finite streams, what determines their (maximal) length?
- Whether windowing is used and which operators are employed to define these windows, i.e. tuple-based, time-based, or other as introduced in Section 3.1?

Analysing these elements provides insight into the ability of a formalism to represent dynamic data and reason over it effectively. It also reveals potential limitations, such as constraints on time granularity or lack of support for infinite streams, which can be decisive for applications requiring real-time responsiveness or interoperability across heterogeneous systems.

T3. Expressivity of static relations The ability of a formalism to express relationships between atomic facts is a fundamental determinant of its reasoning power. Once the atoms $\alpha \in \Omega$ are identified, it becomes necessary to examine how these atoms can be combined into more complex statements. Because syntactic conventions differ substantially across formalisms, a purely syntactic comparison offers limited insight. Instead, we adopt a semantic perspective and evaluate whether a formalism supports three core logical constructs: conjunction (AND), disjunction (OR), and negation (NOT).

The choice of these three operators is motivated by their status as the minimal basis of classical propositional logic. Together, they constitute the foundational operations required for compositional reasoning. Conjunction enables the representation of simultaneous conditions, disjunction captures alternative possibilities, and negation allows for the explicit modeling of absence or contradiction. These operators are not only intuitive from a human reasoning perspective but also sufficient to derive other common constructs such as implication and equivalence. Consequently, their presence or absence provides a meaningful baseline for assessing the expressive adequacy of a formalism.

- **AND:** the concurrence of two atoms, i.e. two 'facts' that **hold true at the same time**. Examples include properties that hold simultaneously or measurements taken parallel to each other. This is represented by $\wedge$ and $\cap$ in FOL and set theory respectively.
- **OR:** the discretion of two atoms, i.e. the discretion is true if and only if **at least one of the atoms is true**, commonly known as (inclusive) OR. Within FOL, this is displayed via disjunction $\vee$. Set theory employs the union $\cup$ of two sets.
- **NOT:** the contrary or **opposite of an atom** α. Knowledge that a fact or assumption is not true can be equally valuable as the knowledge it is true. As such, being able to express the contrary can be an appreciable asset of a formalism's expressivity. Negation $\neg$ and complement $\overline{C}$ are known realisations within FOL and set theory.

Evaluating these operators allows us to determine whether a formalism supports the essential mechanisms for combining facts, which is indispensable for any non-trivial reasoning task.

T4. Temporal expressivity A broader interpretation of "reasoning over streams of data" is that reasoning may extend beyond isolated facts to include temporal dependencies between them. This distinction is captured by two complementary notions:

Reasoning over time refers to applying static reasoning techniques to data that arrives incrementally, without considering temporal relationships between facts. Reasoning about time goes further by explicitly modeling and exploiting temporal relations, enabling the detection of patterns such as persistence, succession, and causality.

The dimension of temporal expressivity evaluates whether a formalism supports reasoning about time and, if so, to what extent. To this end, we focus on a set of temporal operators that have become canonical in temporal logics such as LTL and MTL. These operators form a principled baseline for expressing temporal properties:

- A fact holds over a prolonged period of time, denoted by box-shaped operators $\boxplus, \boxminus$.
- A fact is known to hold at some point, but the exact point in time is not known, or only approximately), i.e. diamond-shaped operators $\diamondplus, \diamondminus$.
- A fact is known to hold continuously (at least) since or until a specified other fact is known to be true, operators $\mathbf{since}$ and $\mathbf{until}$.
- A fact will be true at the next point in time ($\bigcirc$), or was true at the previous point in time ($\bullet$).

This choice reflects a balance between expressivity and analysability: these operators are widely adopted in formal verification and stream reasoning, and they provide sufficient coverage for most practical scenarios without introducing excessive complexity.

Allen's Interval Algebra offers an alternative, interval-centric perspective by defining 13 qualitative relations between time intervals (e.g., before, meets, overlaps, during). While our selected operators are point-based and metric in nature, Allen's relations can often be encoded using combinations of these operators when intervals are represented explicitly. For example, A before B can be expressed by requiring that the end of A precedes the start of B, which corresponds to constraints formulated with $\mathbf{until}$. Conversely, formalisms that natively support Allen's relations (e.g., ELE) exhibit high temporal expressivity because they allow reasoning about interval relationships without reducing them to point-based semantics. In our evaluation, such formalisms are considered to support the full spectrum of temporal relations defined in this dimension.

Assessing these operators provides a principled baseline for comparing formalisms on their ability to reason about temporal patterns. The comparative results for this dimension are summarized in Table 5. In particular, dimensions T4 and T6 operationalize the distinction between reasoning over time and reasoning about time, which explicitly exploits temporal relations between facts. This distinction underpins our evaluation of temporal expressivity and formalism objectives, as it reflects two different interpretations of SR.

T5. Uncertainty in the data and in the stream Real-world streaming environments rarely provide complete or perfectly ordered data; instead, they exhibit uncertainty, vagueness, and temporal irregularities that challenge traditional stream reasoning approaches. Numerous works have identified uncertainty management as a key requirement for practical applications, as data can be imprecise, missing, contradictory, noisy, or unstructured [14]. For example, sensor networks may suffer from data loss or imprecise measurements, while social media streams often contain incomplete conversations or ironic statements that complicate sentiment analysis [17]. These issues demand reasoning models that explicitly account for different sources of uncertainty and incorporate them during computation. Beyond uncertainty, advanced capabilities such as fuzzy semantics and multi-valued approaches enable reasoning over vague concepts and conflicting information, moving beyond strict true/false dichotomies. Handling out-of-order or delayed events is equally essential in distributed settings to maintain temporal consistency. Several modeling paradigms have been proposed to address these challenges, including probabilistic frameworks for uncertainty, fuzzy set theory [44] for graded truth values, and logics such as Gödel or Łukasiewicz for multi-valued reasoning [45]. Collectively, these capabilities and their underlying formalisms significantly enhance the robustness and applicability of stream reasoning in complex, uncertain environments, forming the conceptual foundation for the implementation and evaluation strategies discussed in subsequent sections.

T6. Formalism objective In designing stream reasoning formalisms, we distinguish three fundamental objectives: stream querying, reasoning over time, and reasoning about time. *Stream querying* focuses on extracting relevant information from continuously evolving data, typically through declarative query languages that support temporal windows and incremental evaluation. For example, a query may retrieve all temperature readings above a threshold within the last five minutes from a sensor stream. With *reasoning over time*, we refer to the practice of performing traditional (non-temporal) inference on data, which just so happens to arrive sequentially. A typical example includes incremental updating of a knowledge base. For example, a reasoning engine may infer `sensor1 is operational` based on sequential temperature measurements `temperature(sensor1, 22°C)`, `temperature(sensor1, 23°C)`, ... and update the knowledge base accordingly. The inference itself is static, i.e. it does not depend on temporal patterns or relations between the observations, but it occurs incrementally as new data arrives. Essential here is that new knowledge is derived over time from pieces of incoming information (the 'what'), but the reasoning itself is 'static' and does not take into account temporal patterns or relations (the 'when') of the data. In contrast, *reasoning about time* explicitly includes the temporal dimension in the reasoning, such as detecting patterns or sequences of events; for instance, identifying that a traffic jam occurred after a series of congestion alerts and a road closure. CEP, aimed at recognizing higher-level event patterns over time, falls

under the umbrella of reasoning about time, as well as other frameworks which consider temporal relations in their inference process, for example via LTL operators $\boxplus$, $\diamondsuit$ etc.

Distinguishing these objectives is crucial in a SR context: while querying provides the foundation for accessing dynamic data, reasoning over time ensures consistency and incremental enrichment of knowledge, and reasoning about time enables richer interpretations of temporal structure. Although these categories are conceptually useful, they are not strictly disjoint. For instance, one might carefully craft queries in a stream querying framework such as C-SPARQL to detect a specific temporal pattern, which effectively constitutes a form of reasoning about time. Nonetheless, specifying the primary objective of a formalism helps clarify its strengths, weaknesses, and underlying design philosophy, and provides an additional perspective when comparing approaches with different goals. Each of these objectives has its proper place in the stream reasoning research landscape, ensuring SR systems can support both low-level data access and high-level temporal inference.

T7. Complexity and Decidability Complexity and decidability analyses play a crucial role in understanding the theoretical properties of stream reasoning formalisms. They provide insight into the trade-off between expressiveness and computational feasibility, guiding both the design of new approaches and the selection of suitable formalisms for practical applications. For instance, knowing whether query evaluation is tractable or whether reasoning remains decidable under certain restrictions helps anticipate scalability and implementation challenges. However, such results are still relatively scarce in SR literature, and when available, they often apply to specific fragments or under bounding criteria such as fixed window sizes or restricted recursion. Given these theoretical results are nonetheless an important characteristic of a SR formalism, we consider for each formalism whether reasoning tasks are decidable or undecidable, and we indicate known complexity bounds using standard classes such as $\mathtt{P}$, $\mathtt{NP}$, or $\mathtt{EXP}$, when available in the considered literature.

5.3 Usage Dimensions

The usage dimensions capture the practical footprint of a SR formalism within the research community and in real-world applications. While theoretical properties determine what a formalism can express and how it behaves, usage dimensions reveal whether these capabilities translate into adoption and impact. They measure academic visibility through citation counts, the availability of software prototypes or implementations that demonstrate feasibility, and evidence of deployment in concrete application domains. Together, these indicators provide insight into the maturity and relevance of a formalism beyond its conceptual design, highlighting which approaches have gained traction and which remain largely theoretical.

U1. Citations As each of the formalisms considered in our survey stems from a research paper, the amount of citations of the paper provides a direct and tangible metric for the degree in which the formalism is valued by the research community. The primary indicator in this regard is the amount of citations of the reference paper for the formalism. In case no single source paper can be identified, e.g. due to the formalism being introduced (by the same authors) in multiple venues and/or journals, we take each of these papers into consideration and sum their citation counts. The results are summarized in Table 8.

U2. Software prototypes To assess the availability of prototypes for a given stream reasoning formalism, we define a prototype as a piece of generic software that can be used across different applications to perform stream reasoning. In our analysis, we consider all prototypes referenced in the papers included in the secondary study (see Subsection 2.2). We restrict this dimension to high-level implementations and exclude solutions developed exclusively for one specific application, as those are addressed in the next dimension. In addition to dimension U1, Table 8 holds the prototypes of the considered formalisms.

U3. Applications To gauge a formalism's contribution toward tangible applications and products, we examine the real-life systems and solutions built upon the considered SR formalisms. In particular, we investigate the domains in which these formalisms have been applied, as summarized in Table 9, since application areas provide insight into their practical relevance and versatility. For this purpose, we rely on the material collected in the final step of our protocol, the secondary study, which reports implementations and deployments referenced in the literature.

6 Survey Results: Theory Dimension

This section examines the seven theoretical dimensions introduced in Section 5, which capture the conceptual foundations and formal properties of SR formalisms. By analyzing each formalism against these dimensions, we address **Q1**, **Q2**, and **Q3**: identifying common constructions and expressions, establishing criteria for evaluating suitability for SR, and comparing recently developed formalisms with respect to these criteria.

We begin with an overview of the surveyed formalisms grouped by dimension and category, supported by comparative tables for quick reference. Specifically, dimension T1 (Theoretical Basis) and T3 (Static Expressivity) are covered in Table 4, dimension T2 (Stream Representation and Windowing) is covered in Table 6, dimension T4 (Temporal Expressivity) is covered in Table 5, and dimension T5 (Uncertainty Handling) is covered in Table 7. Dimensions T6 and T7 are discussed in detail in the narrative, as their interpretation requires more context than tabular representation.

Following this overview, the subsequent subsections provide a detailed discussion of each category of formalisms: (a) DL-based, (b) Temporal Logic-based, (c) SR query languages, (d) ASP-based, and (e) Miscellaneous approaches. For each category, we describe how individual formalisms address the seven theoretical dimensions, highlighting design choices, strengths, and limitations. This structure enables readers to either consult the tables for direct comparison or explore the narrative for an in-depth understanding of specific formalisms.

Formalism	Basis	Logical relations				
		$\neg$	$\wedge$	$\vee$	$\rightarrow$	other operators
DL-Lite TQL	DL	✓	✓	R	R	-
DL-Lite TQL^{+}	DL	-	✓	-	-	-
DL-Lite TQL$^{\exists}$	DL	✓	✓	R	R	-
DL-Lite TQL$^{\exists,+}$	DL	-	✓	-	-	-
LTL$_{\mathcal{EL}}^{\mathbf{X},\mathbf{G}}$	DL $\mathcal{EL}$	-	✓	-	-	$\exists$
$\mathcal{EL}^{++}$ ontology streams	DL $\mathcal{EL}^{++}$	-	✓	-	-	$\exists$
DatalogM(I)TL	Datalog	-	✓	-	✓	-
DatalogM(I)TL^{FP}	Datalog	-	✓	-	✓	-
DatalogM(I)TL$^{\neg}$	Datalog	✓	✓	-	✓	-
DatalogM(I)TL$^{\exists}$	Datalog	-	✓	-	✓	$\exists$
MSTL	RCC-8	✓	✓	✓	✓	$\exists, \forall$
P-MTL	MTL	✓	✓	✓	✓	$\exists, \forall$
PMLTL	MLTL	✓	✓	✓	R	-
TEL	EL	✓	✓	✓	✓	$\exists, \forall$
TEL + past operators	EL	✓	✓	✓	✓	$\exists, \forall$
MEL	EL	✓	✓	✓	✓	-
LTPAL	PAL	✓	✓	R	R	-
ASTL	STL	✓	✓	✓	-	-
ProbSTL	STL	✓	R	✓	R	-
EvTL	STL	✓	R	✓	R	-
RobTL	STL	✓	✓	R	R	-
SDRL	-	✓	✓	R	R	-
TESLA	TRIO	✓	✓	R	R	$\forall$
TAL	PMON	✓	✓	✓	✓	$\forall, \exists$
C-SPARQL	SPARQL	✓	✓	✓	R	$\exists, \forall$
STARQL	SPARQL	✓	✓	✓	R	$\exists, \forall$
SPARQL_stream	SPARQL	✓	✓	✓	R	$\exists, \forall$
CT-SPARQL	SPARQL	✓	✓	✓	R	$\exists, \forall$
ELE	-	✓	✓	✓	R	-
EP-SPARQL	SPARQL	-	✓	✓	-	$\exists, \forall$
CQELS	SPARQL	✓	✓	✓	R	$\exists, \forall$
RSP-QL	SPARQL	✓	✓	✓	R	$\exists, \forall$
RSP-QL*	SPARQL	✓	✓	✓	R	$\exists, \forall$
C-ASP Language	ASP	✓	✓	-	✓	-
LARS	ASP	✓	✓	✓	✓	-
Metric LARS	ASP	✓	✓	✓	✓	-
wLARS	ASP	✓	✓	✓	✓	-
PrASP	ASP	✓	✓	✓	✓	-
Temporal Datalog	Datalog	-	✓	-	✓	-
RTEC	Event Calculus	-	✓	-	-	-

Table 4. Summary of the T1 (theoretical basis) and T3 (expressivity of static relations) dimensions for SR formalisms. ✓ indicates the relation is natively defined and R indicates the relation can be defined via recursion.

6.1 DL-based Formalisms

Description Logics (DL) have long been established as a cornerstone of knowledge representation and underpin many SW technologies, most notably Web Ontology Language (OWL) [24]. Their distinction between terminological knowledge (TBox) and assertional knowledge (ABox) makes them well-suited for scenarios where static domain knowledge must be combined with dynamic data streams. This separation allows DL-based approaches to incorporate evolving observations while preserving a structured conceptual model. Consequently, DL-based formalisms have been explored forSR both through direct application [46] and through extensions that introduce temporal or predictive capabilities [47].

6.1.1 DL-Lite TQL and its fragments Klarman and Meyer [47] combine the DL-Lite family of description logics with a set of temporal rules, taken from the Temporal Query Language (TQL) [48]. TQL is composed of formulae that tie together multiple queries using conjunction $\wedge$, negation $\neg$, **since** and **until**, i.e. $\neg q, q_1 \wedge q_2, q_1$ **since** q_2, q_1 **until** q_2. This results in **DL-Lite TQL**, a SR formalism with DL fragment DL-Lite as a theoretical basis (T1). With temporal rules of the form $\phi \Rightarrow \psi$, with ϕ and ψ formulae as above, it is able to infer new information ψ when ϕ is met. As ϕ and ψ can capture temporal relations, DL-Lite TQL is therefore aimed at reasoning *about* time (T6). It thereby distinguishes two strands of inference, i.e. inference over data streams towards the future ('prediction') and towards the past ('explanation').

Formalism	Temporal Relations										
	◯	●	since	until	□	⊞	⊟	◇	⟐	⟑	Interval
DL-Lite TQL	-	-	✓	✓	-	R	R	-	R	R	-
DL-Lite TQL^{+}	-	-	✓	✓	-	-	-	-	R	R	-
DL-Lite TQL$^{\exists}$	-	-	-	-	-	R	R	-	✓	✓	-
DL-Lite TQL$^{\exists,+}$	-	-	-	-	-	-	-	-	✓	✓	-
LTL$_{\mathcal{EL}}^{\mathbf{X},\mathbf{G}}$	✓	-	-	-	-	✓	-	-	-	-	-
DL $\mathcal{EL}^{++}$ + ontology stream	-	-	-	-	-	-	-	-	-	-	-
DatalogM(I)TL	-	-	✓	✓	-	✓	✓	-	✓	✓	✓
DatalogM(I)TL^{FP}	-	-	-	-	-	✓	✓	-	-	✓	✓
DatalogM(I)TL$^{\neg}$	-	-	✓	✓	-	✓	✓	-	✓	✓	✓
DatalogM(I)TL$^{\exists}$	-	-	✓	✓	-	✓	✓	-	✓	✓	✓
MSTL	✓	-	-	-	✓	-	-	✓	-	-	✓
P-MTL	✓	-	-	-	✓	-	-	✓	-	-	✓
PMLTL	-	-	-	✓	-	✓	-	-	✓	-	✓
TEL	✓	-	-	✓	-	✓	-	-	✓	-	-
TEL + past operators	✓	✓	✓	✓	-	✓	✓	-	✓	✓	-
MEL	✓	✓	✓	✓	-	R	R	-	R	R	R
LTPAL	✓	-	-	✓	-	R	-	-	R	-	-
ASTL	-	-	-	✓	-	✓	-	-	✓	-	✓
ProbSTL	-	-	-	✓	-	R	-	-	R	-	✓
EvTL	-	-	-	✓	-	R	-	-	R	-	✓
RobTL	-	-	-	✓	-	R	-	-	R	-	✓
SDRL	✓	✓	R	R	-	R	R	-	R	R	✓
TESLA	R	R	R	R	R	R	R	R	R	R	✓
TAL	-	-	-	✓	-	✓	-	-	✓	-	✓
C-SPARQL	-	-	-	-	-	-	-	-	-	-	-
STARQL	-	-	-	-	-	✓	✓	-	✓	✓	-
SPARQL_stream	-	-	-	-	-	-	-	-	-	-	-
CT-SPARQL	-	-	-	-	-	-	-	-	-	-	-
ELE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
EP-SPARQL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
CQELS	-	-	-	-	-	-	-	-	-	-	-
RSP-QL	-	-	-	-	-	-	-	-	-	-	-
RSP-QL*	-	-	-	-	-	-	-	-	-	-	-
C-ASP Language	-	-	-	-	-	-	-	-	-	-	-
LARS	-	-	-	-	✓	-	-	✓	-	-	-
wLARS	-	-	-	-	✓	-	-	✓	-	-	-
Metric LARS	-	-	-	-	✓	✓	✓	✓	✓	✓	✓
PrASP	-	-	-	-	-	-	-	-	-	-	-
Temporal Datalog	-	-	-	-	-	-	-	-	-	-	-
RTEC	✓	✓	✓	✓	-	-	-	-	-	-	-

Table 5. Summary of the T4 dimension (temporal expressivity) for the different surveyed formalisms. ✓ indicates the relation is natively defined and R indicates the relation can be defined via recursion.

The concept of a data stream (T2) is modeled as a sequence of (ABox) data $(\mathcal{A}_i)_{i\in\mathbb{Z}}$ over time domain $\mathbb{Z}$. As such, time is explicitly modeled as a discrete timeline $\mathbb{Z}$. The ABox sequence is tied to a fixed TBox and can theoretically be infinite. Each ABox A_i has to be consistent with the TBox of the stream. Klarman and Meyer [47] make no mention of windowing mechanisms for DL-Lite TQL.

The stream and TBox can be queried at a time $i \in \mathbb{Z}$. TQL takes a set of conjunctive queries, recursively combined with logical operators $\neg$, $\wedge$, **until** and **since**. In terms of non-temporal operators, DL-Lite TQL is thus equipped with negation, conjunction. Via recursion, is can also express disjunction and implication (T3). In terms of temporal operators, it has **since** and **until** at disposal. Via recursion, relations $\boxplus, \boxminus$, $\diamondplus$ and $\diamondminus$ can be expressed as well (T4). All temporal relations apply to either the past or future as a whole, with no possibility to specify time intervals.

All formulae are evaluated to be either true or false, thus with no provisions for uncertainty in the data. The streaming data sequence $(\mathcal{A}_{i\in\mathbb{Z}})$ is considered to in-order. As such, unreliability of the stream such as out-of-order arrival is not taken into account (T5).

Furthermore, Klarman and Meyer [47] introduces several fragments of the aforementioned TQL, restricting the available operators within either the static (T3) or temporal (T4) dimension, or both. A first fragment, TQL^{+}, consists of the formulae that do not contain negation $\neg$. As expressivity of disjunction and implication relies on negation, only disjunction and implication are lost (T3) as well as $\boxplus$ and $\boxminus$ (T4). A second fragment TQL$^{\exists}$ limits the **since** and **until**

Formalism	Representation		Windowing		
	Time	Stream	Tuple	Time	Other
DL-Lite TQL	$\mathbb{Z}$	sequence	-	-	-
DL-Lite TQL^{+}	$\mathbb{Z}$	sequence	-	-	-
DL-Lite TQL$^{\exists}$	$\mathbb{Z}$	sequence	-	-	-
DL-Lite TQL$^{\exists,+}$	$\mathbb{Z}$	sequence	-	-	-
LTL$_{\mathcal{EL}}^{\mathbf{X},\mathbf{G}}$	$\mathbb{N}$	sequence	-	-	-
DL $\mathcal{EL}^{++}$ + ontology stream	$\mathbb{N}$	sequence	-	-	-
DatalogM(I)TL	$\mathbb{Z}$ or $\mathbb{Q}$	mapping function	-	-	-
DatalogM(I)TL^{FP}	$\mathbb{Z}$ or $\mathbb{Q}$	mapping function	-	-	-
DatalogM(I)TL$^{\neg}$	$\mathbb{Z}$ or $\mathbb{Q}$	mapping function	-	-	-
DatalogM(I)TL$^{\exists}$	$\mathbb{Z}$ or $\mathbb{Q}$	mapping function	-	-	-
MSTL	ordered set $(T, <)$	mapping function	-	-	-
P-MTL	$\mathbb{N}$	mapping function	-	-	-
PMLTL	$\mathbb{N}$	sequence	-	-	-
TEL	Unspecified	sequence	-	-	-
TEL + past operators	Unspecified	sequence	-	-	-
MEL	Unspecified	sequence	-	-	-
LTPAL	$\mathbb{N}_{<n}$	sequence	-	-	-
ASTL	$\mathbb{N}$	mapping function	-	-	-
ProbSTL	$\mathbb{N}$	mapping function	-	-	-
EvTL	$\mathbb{N}$	sequence	-	-	-
RobTL	$\mathbb{N}$	sequence	-	-	-
SDRL	Unspecified	mapping function	-	-	-
TESLA	Unspecified	mapping function	-	-	-
TAL	linear $(T, <)$	set	-	-	-
C-SPARQL	Unspecified	sequence	✓	✓	-
STARQL	linear $(T, <)$	sequence	-	✓	-
SPARQL_stream	Unspecified	sequence	-	✓	-
CT-SPARQL	Unspecified	sequence	-	-	-
ELE	$\mathbb{Q}_{\geq 0}$	mapping function	-	-	-
EP-SPARQL	Unspecified	sequence	✓	✓	-
CQELS	$\mathbb{N}$	sequence	✓	✓	-
RSP-QL	$\mathbb{N}$	sequence	✓	✓	-
RSP-QL*	$\mathbb{N}$	sequence	✓	✓	-
C-ASP Language	Unspecified	sequence	✓	✓	-
LARS	$\mathbb{N}$	mapping function	✓	✓	partition
Metric LARS	$\mathbb{N}$	mapping function	✓	✓	partition
wLARS	$\mathbb{N}$	mapping function	✓	✓	partition
PrASP	$\mathbb{N}$	sequence	-	-	-
Temporal Datalog	$\mathbb{N}$	sequence	-	-	-
RTEC	$\mathbb{Z}$	sequence	-	✓	-

Table 6. Summary of the T2 dimension (time and stream representation) for SR formalisms.

operators to occur only with the general truth $\top$ as their first argument, i.e. $\top$ **until** ϕ and $\top$ **since** ϕ, which coincides with $\oplus$ and $\ominus$ respectively. This restriction removes the ability to express the **since** and **until** relations fully, but retains expressivity of the other temporal relations (T4). A third and last fragment proposed by Klarman and Meyer [47] is TQL$^{\exists,+}$, the intersection of TQL$^{\exists}$ and TQL^{+}. In this case, the only relations remaining are $\wedge$, $\oplus$ and $\ominus$ (T3, T4).

The complexity of fact checking in DL-Lite TQL when it comes to deduction, with the fact encoded as a TQL formula, is proven to be PSPACE-complete (T7). Regarding its fragments, deduction is NP-complete for TQL$^{\exists}$, TQL^{+} and TQL$^{\exists,+}$. In terms of decidability, no results were provided in the work of Klarman and Meyer [47].

6.1.2 LTL$_{\mathcal{EL}}^{\mathbf{X},\mathbf{G}}$ Extending DL with temporal operators is also done in **LTL**$_{\mathcal{EL}}^{\mathbf{X},\mathbf{G}}$, introduced by Tirtarasa and Turhan [49], where LTL operators $\bigcirc$ (next) and $\Box$ (global always) are incorporated into DL $\mathcal{EL}$ (T1). It aims to construct temporalized query concepts, which in turn enables one to answer temporal queries, more specifically temporal conjunctive queries as seen in the context of ontology-based data access. With a stream representation $I = (A_i)_{i\in\mathbb{N}}$, having A_i ABoxes and time domain $\mathbb{N}$ (T2) , LTL$_{\mathcal{EL}}^{\mathbf{X},\mathbf{G}}$ uses sets to represent concepts, e.g. the concept 'house' is represented by the set of all elements that are a houses (e.g. the set of townhouses, villa's, apartments, houseboats etc.). These concept are subsequently combined with static operators $\cap$ and $\exists$ (T3) and temporal operators $\mathbb{X}$ and $\mathbb{G}$, expressing relations $\bigcirc$ and $\boxplus$ (T4). By asserting whether

Formalism	Uncertainty			
	Probabilistic	Fuzzy	Multi-valued	Out-of-order data
DL-Lite TQL	-	-	-	-
DL-Lite TQL^{+}	-	-	-	-
DL-Lite TQL$^{\exists}$	-	-	-	-
DL-Lite TQL$^{\exists,+}$	-	-	-	-
LTL$_{\mathcal{EL}}^{\mathbf{X},\mathbf{G}}$	-	-	-	-
DL $\mathcal{EL}^{++}$ + ontology stream	-	-	-	-
DatalogM(I)TL	-	-	-	-
DatalogM(I)TL^{FP}	-	-	-	-
DatalogM(I)TL$^{\neg}$	-	-	-	-
DatalogM(I)TL$^{\exists}$	-	-	-	-
MSTL	-	-	-	-
P-MTL	✓	-	-	-
PMLTL	-	-	-	-
TEL	-	-	-	-
TEL + past operators	-	-	-	-
MEL	-	-	-	-
LTPAL	-	-	-	-
ASTL	-	-	$\mathbb{R}$	-
ProbSTL	✓	-	-	-
EvTL	✓	-	-	-
RobTL	-	-	-	✓
SDRL	-	-	-	-
TESLA	-	-	-	-
TAL	-	-	-	-
C-SPARQL	-	-	-	-
STARQL	-	-	-	-
SPARQL_stream	-	-	-	-
CT-SPARQL	-	-	-	-
ELE	-	-	-	-
EP-SPARQL	-	-	-	-
CQELS	-	-	-	-
RSP-QL	-	-	-	-
RSP-QL*	✓	✓	✓	-
C-ASP Language	-	-	-	-
LARS	-	-	-	-
wLARS	✓	✓	✓	-
Metric LARS	-	-	-	-
PrASP	✓	✓	semiring $\mathcal{R}$	-
Temporal Datalog	-	-	-	-
RTEC	-	-	-	-

Table 7. Summary of the T5 dimension (uncertainty in the data and in the stream) for SR formalisms.

instances adhere to these concepts, LTL$_{\mathcal{EL}}^{\mathbf{X},\mathbf{G}}$ perform an inference over temporal relations, i.e.

With roots in DL, LTL$_{\mathcal{EL}}^{\mathbf{X},\mathbf{G}}$ relies on binary assertion of statements over a sequence $(\mathcal{A}_i)_{i\in\mathbb{N}}$, not accounting for uncertainty in either data or the stream (T5). Tirtarasa and Turhan [49] does not consider concrete results on the complexity or decidability of LTL$_{\mathcal{EL}}^{\mathbf{X},\mathbf{G}}$.

6.1.3 $\mathcal{EL}^{++}$ ontology streams Another DL-based formalism (T1) is ontology streams over $\mathcal{EL}^{++}$ [50], which aims to identify the nature and cause of changes on an ontology level, where the ontology is seen as the combination of ABox and TBox. Here, a stream is a sequence of ontologies $(O_{t_0}, O_{t_1}, \ldots, O_{t_n})$ where O_t are ontologies. The sequence is finite and ranges from time point t_0 to t_n, taken from time domain $\mathbb{N}$ (T2). Aside from the intersection of concepts $\cap$ and the existential operator $\exists$ inherent to $\mathcal{EL}^{++}$ [51] (T3), Lécué [50] does not introduce new operators, neither temporal or non-temporal (T4). The overall objective of detecting changes in ontologies is modeled via the deduction of new information, whereas model checking is used to spot inconsistencies across ontologies in the stream. As it incrementally updates the ontology, but does not consider temporal relations between the incoming data, the $\mathcal{EL}^{++}$ ontology streams fall under reasoning over streams (T6).

With regards to the stream and the data within the stream, the formalism assumes the order of the stream is correct and the data itself, i.e. the ontology, is trustworthy (T5). From $\mathcal{EL}^{++}$, the formalism employs the concept of General Concept Inclusion axioms (CGIs). Checking if a given GCI stays invariant (i.e holds for) k consecutive moments over an ontology stream O_m^n is PTIME-complete [50] (T7).

6.2 Temporal Logic Formalisms

The existence of temporal logic languages predates the field of SR. As such, a recurring approach to SR formalism design is to enhance a state-of-the-art static formalism or language with temporal capabilities.

6.2.1 DatalogMTL and its fragments One such formalism is DatalogMTL [52], which starts from Datalog (T1) as theoretical basis for static data and incorporates operators from MTL into the syntax. DatalogMTL aims to specify and reason over complex events in real-time systems. As a recursive rule language, it is well-suited to tackle analysis involving recursive propagation of information. In DatalogMTL, the syntax remains rooted in Datalog's foundational principles, with formulae comprising of atomic assertions and temporal constraints expressed through temporal operators. These operators, specifically $\mathbf{since}_I$, $\mathbf{until}_I$, $\boxplus_I$, $\boxminus_I$, $\diamondplus_I$, $\diamondminus_I$, introduce temporal logic into the framework, enabling the specification of temporal patterns and constraints over data streams (T4). The integration of temporal logic enhances Datalog's expressive power, allowing for sophisticated reasoning over dynamic and evolving data streams. Furthermore, rules in DatalogMTL maintain the familiar structure of traditional Datalog, with a clear separation between rule heads and bodies. The rule body consists of formulae built using conjunction $\wedge$ and the six aforementioned temporal operators, whereas in the rule head, only formulae with operators $\boxplus$ and $\boxminus$ are permitted. Compound expressions are thereby allowed, such as $\left(\boxplus_{I_1} \phi\right) \mathbf{until}_{I_2} \left(\diamondminus_{I_3} \psi\right)$ ("ϕ always holds interval I_1 in the future up until ψ holds somewhere in I_2 in the past") in rule bodies and $\boxplus_{I_2} \boxminus_{I_2} \psi$ ("everywhere in interval I_1 in the future, ψ holds in interval I_2 in the past") in both bodies and heads. Since DatalogMTL only allows $\wedge$ in the body and supports implication via the rule construct, the only supported static relations are $\wedge$ and $\rightarrow$ (T3). As rules can contain temporal expressions, both in the rule head and body, DataloMTL is capable of reasoning about time (T6). Since it only considers logical entailment and a correct order of the stream, it does account for uncertainty in the data or the stream (T5).

DatalogMTL considers a timeline T, which can either be an integer timeline $\mathbb{Z}$ ('unary timeline') and rational timeline $\mathbb{Q}$ ('binary timeline'). A stream is modeled as a function S which maps a point t fomr the timeline to a set of grounded atoms, $S(t)$ (T2).

Brandt et al. [52] provides several complexity and decidability results for DatalogMTL. For a given DatalogMTL program, i.e. set of DatalogMTL rules, consistency checking is proven to be EXPSPACE-complete, even when only variable-free propositions are considered for atoms (T7).

DatalogMTL sports several fragments as well as extensions, each aimed at specific tasks.

DatalogMTL^{FP}[53] is dedicated to deriving information about the future based on information from the past. To achieve this, the only operators allowed in the rule body are the past operators $\boxminus$ and $\diamondminus$, whereas the rule head only includes the future operator $\boxplus$, limiting temporal expressivity (T4). Recall diamond and since/until operators are already disallowed in rule heads by regular DatalogMTL.

DatalogMTL$^{\neg}$ has been proposed by Cucala et al. [54] as a supplement to DatalogMTL, catering to a growing need of non-monotonic negation, where previous conclusions can possible be revoked due to newer information [55]. The rule bodies of original DatalogMTL are composed of multiple *atoms* connected using $\wedge$. The atoms are strictly *positive*, i.e. they do not contain negation $\neg$. DatalogMTL$^{\neg}$ introduces stratified negation as failure by extending the format of rule bodies to $A_1 \wedge \ldots \wedge A_n \wedge \neg A'_1 \wedge A'_m$, thus adding operator $\neg$ (T3). The term stratified indicates that negation is applied in layers (strata), ensuring that rules only negate facts computed in lower strata. This prevents cyclic dependencies through negation and preserves tractability. Importantly, this extension does not increase the complexity of fact entailment [54] (T7).

The atoms that provide the basis for the rules can contain variables, but DatalogMTL operates under the assumption that every variable in the rule head must also occur in (an atom of) the rule body. This prohibits 'existential rules', where one can conclude the existence of an (unspecified) instance based on several pieces of information. Nissl [56] introduces a template to incorporate existential rules into DatalogMTL, leading to the extension DatalogMTL$^{\exists}$. DatalogMTL$^{\exists}$ can be used with two semantics, either 'natural semantics' that align with those of Datalog fragment Datalog$^{\exists}$ which similarly allows existential rule in Datalog (cfr. Leone et al. [57]), or 'uniform' semantics. In short, with the latter an existential quantification $\exists$ must hold for a single instance over a given interval, whereas the former allows for multiple instances throughout an interval, as long as the existential quantification $\exists$ holds at each time point separately (T3). Nissl [56] provides complexity results for model checking under both semantics, being EXPSPACE-complete for uniform semantics under the Closed World Assumption (CWA), and undecidability in all other cases (T7).

DatalogMITL is a fragment of DatalogMTL identical in syntax, save for the exclusion of singleton intervals (e.g. $[1, 1]$) in the operators (T4) [53]. This follows from the MTL fragment MITL, which only allows for non-singleton intervals. A benefit of this limitation is that the required memory for an arbitrary reasoning algorithm can be controlled independently of the chosen time domain [53]. For DatalogMITL^{FP}, the forward-propagating fragment of DatalogMTL, Wałęga et al. [53] provides complexity results for fact entailment. In case of a binary encoding of the time points, checking whether a single piece of information at a specific time, denoted $\alpha@t$, holds for the given dataset, is PSPACE-complete. In case of a unary encoding, it is P-complete (T7).

As DatalogMITL^{FP} is a restricted fragment of both DatalogMTL^{FP} and of DatalogMTL, the complexity results of DatalogMITL^{FP} also serve as lower bounds on the complexity of the parent formalisms (T7).

6.2.2 Metric Spatio-Temporal Logic (MSTL) Another MTL-extended formalism is MSTL [58], which combines MTL with Region Connection Calculus (RCC-8) [59], a qualitative spatial reasoning framework aimed at describing spatial relations between regions (e.g. overlap, bordering, ...) (T1). The aim of MSTL is to perform spatio-temporal reasoning, more specifically to infer unobservable qualitative spatial relations and their evolution over time via the

introduction of the 'landmark' concept, i.e. a region that remains unchanged throughout time.

Focusing on the temporal part, we observe MSTL covers operators **next**, $\boxplus$ and $\diamondplus$. Additionally, it sports operators $\Box_{[i]}$ and $\Diamond_{[t_1,t_2]}$. These operators define the 'always' and 'eventually/once' temporal relations, but the semantics of MSTL dictate that the interval subscripts specify an absolute time interval, compared to the relative time intervals seen in other MTL-based formalisms (cfr. [53, 60, 61]). As a result, MSTL does not support the 'traditional' MTL-operators $\boxminus$ and $\blacklozenge$, despite its ability to describe 'always/once' relations that may occur before a considered time point (T4). Furthermore, MSTL has static operators $\neg, \wedge, \vee, \veebar, \rightarrow, \forall$ and $\exists$ (T3), to be combined freely with the aforementioned temporal operators, to construct their spatio-temporal formulae.

MSTL models time explicitly by means of an ordered set of time points $(T, <)$, which can thus be $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ or other. In the semantics of MSTL, a stream is not defined explicitly, but rather implicitly via an interpretation which maps every time point t and predicate P to a set of constants for which P holds at time t. No window operators are considered (T2).

By assessing the truth value of these spatio-temporal formulae, MSTL is able to perform reasoning about time over streams of data (T6). Problems of uncertainty of the data or the stream order are not considered in de Leng and Heintz [58] (T5),nor does it consider any results in terms of complexity or decidability (T7).

6.2.3 Predictive MTL (P-MTL) A final MTL-based SR formalism concerns P-MTL [61], a formalism for reasoning over uncertain states within a SR context. It aims to fill the gap between lower-level probabilistic inference on sensor data and the higher-level logical reasoning performed on the same information. Starting from MTL (T1), P-MTL incorporates states that can be predicted, thereby enabling temporal reasoning over uncertain information, in contrast to only certain information. It considers stochastic variables, for example the location of a ball, and predicts the value of said variable in a future state, based on a given state. The probabilistic aspect of the logic is embedded via probabilistic statements, say ϕ, which take values on $[0, 1]$, and evaluating these expressions using comparative expressions such as $\phi > 0.85$ or $\phi < 0.05$ (T5).

The stochastically evaluated formulae are formed by recursively combining predicate terms such as $P(\tau_1, \tau_n)$ with both static operators $\neg, \wedge\vee, \rightarrow, \forall, \exists$ (T3) and temporal operators $\bigcirc, \boxplus_{[t_1,t_2]}, \diamondplus_{[t_1,t_2]}, \Box$ and $\Diamond$ (T4). To this end, it is able to perform (stochastic) reasoning over time (T6).

Similar to MSTL above, a stream is implicitly defined as a mapping from a time point t and predicate P to a set of constants for which P holds at time t. The time points t are taken from a natural timeline $T = \mathbb{N}$. Formulae are evaluated on a single stream, without implementing window operators (T2).

Tiger and Heintz [61] states the construction of probabilistic statements ϕ being evaluated in comparative expressions such as $\phi < 0.05$, does not increase complexity or affect decidability compared to FOL expressions. Concrete effects on complexity and decidability of the evaluation over the temporal dimension are not given by Tiger and Heintz [61] (T7).

6.2.4 Predictive Mission-time Linear Temporal Logic (PMLTL) Predictive Mission-time Linear Temporal Logic (PMLTL) [62] is an extension of Mission-time Linear Temporal Logic (MLTL) (T1) aimed at predicting the environment of a system. It does so in a multimodal approach, i.e. by considering and evaluating multiple different scenario's (models) of the future.

PMLTL adopts the time representation of MLTL of finite timed traces $\pi_0\pi_1 \ldots \pi_n$, a finite sequence of sets of atoms. As a result, the time domain follows from the indexing of these timed traces, i.e. $\mathbb{N}$ (T2).

Starting from the atoms which populated the timed traces π_j, PMLTL constructs expressions using static operators $\neg, \wedge, \vee$ and by recursion, $\rightarrow$ (T3), and temporal operators $\boxplus_I, \diamondplus_I$, **until** (T4).

In order for a PMLTL formula to hold by a certain time point in the future (the 'deadline'), it needs to either be inferred from the observed data, or if it is predicted to be true in all of the different models of the future, based on the given timed trace. As such, it thus models the uncertainty of a statement concerning the future by considering multiple different models (T5). As formulae are able to express various temporal relations between the atoms, PMLTL is capable of performing reasoning about time (T6).

No analysis of complexity or decidability of PMLTL is performed in Aurandt et al. [62] (T7).

6.2.5 Temporal Equilibrium Logic (TEL) Temporal Equilibrium Logic (TEL) extends EL by incorporating past temporal operators from LTL. As a result, TEL can be used to tackle non-monotonic reasoning problems with a temporal dimension. From a theoretical point of view, Cabalar and Pérez Vega [63] extend HT with LTL operators, producing Temporal Here-and-Ther Logic (THT). Using the same minimisation principle that produces EL from HT (cfr. 3.2), TEL is produced from THT. In TEL, formulae consist of atomic assertions and temporal constraints expressed through formulae with static operators $\wedge, \vee$ and $\rightarrow$ (T3) and temporal operators $\bigcirc$, **until**, $\boxplus$ and $\diamondplus$ (T4). Negation $\neg\phi$ is additionally available as shorthand for the formula $\phi \rightarrow \bot$. By integrating (future) temporal operators into EL, TEL provides a natural and intuitive framework for reasoning about time (T6), specifically for temporal relations concerning the future, given only the 'future'-operators are considered. $\langle H, T\rangle$ Building upon EL, the basis for a stream in TEL is a pair $\langle H, T\rangle$. For TEL, $\langle H, T\rangle$ is a possibly infinite sequence of pairs of atom sets $\langle H_i, T_i\rangle$. The exact nature of 'time point' i and thereby the timeline T is not specified, but allows for both finite and infinite timelines (T2).

Similar to EL, TEL is able to deal with uncertainty in the data by distinguishing between a 'here' part H_i of what is known to be true at time i and a 'there' part T_i which could potentially be true at i (T5). Cabalar and Pérez Vega Cabalar and Pérez Vega notes complexity and decidability analyses as potential future work following the introduction of TEL.

In later work, Aguado et al. [64] presented a method of extending TEL with operators **since**, $\boxminus$, $\blacklozenge$ and $\bullet$ (T4). The extension can subsequently be reduced to the original TEL syntax via the introduction of auxiliary atoms.

6.2.6 Metric Equilibrium Logic (MEL) In a similar vein to TEL, Cabalaret al. [65] extends EL (T1) with MTL operators. It does so via the same procedure as TEL, starting from adding MTL operators to HT to obtain Metric Here-and-There Logic (MHT) and subsequently consider the minimal timed traces to arrive at the proposed MEL. MEL is equipped with the interval-variants of the operators $\mathbf{next}$, $\mathbf{previous}_I$, $\mathbf{since}_I$ and $\mathbf{until}_I$. Via recursive definition of the formulae, operators $\boxplus_I$, $\boxminus_I$, $\diamondplus_I$ and $\diamondminus_I$ also follow (T4). MEL retains the same static operators as TEL, i.e. $\wedge, \vee$ and $\rightarrow$ (T3). Likewise, dimensions (T2) and (T5-7) remained unchanged compared to TEL discussed above.

6.2.7 Linear Temporal Public Announcement Logic (LTPAL) Linear Temporal Public Announcement Logic (LTPAL) [66] extends PAL (T1) with operators from LTL, enabling reasoning about temporal properties and events within a framework of public announcements and temporal constraints. It is designed to reason over the output of AI systems, such as classifiers, on a uniform basis. In LTPAL, formulae consist of public announcements and temporal constraints expressed through LTL temporal operators $\mathbf{next}$ and $\mathbf{until}$ (T4). This in addition to static operators $\neg$, $\wedge$ (T3) inherited from PAL. Via recursion, LTPAL is also fitted with static operators $\vee$ and $\rightarrow$ (T3) and temporal operators $\boxplus$ and $\diamondplus$ (T4). The resulting set of operators allow for the specification of temporal patterns and dependencies over data streams, thus enabling reasoning about time in LTPAL (T6).

LTPAL models the flow of time via a model, referred to as a transition system. Here, a finite set of states describes changes over time and the LTPAL formulae are evaluated across the various states. As a result, the time domain is implicitly assumed to be a finite set of time points $(0, \ldots, n)$ (T2).

The formulae of LTPAL are evaluated on a binary basis, not accounting for uncertainty of the expressed relations. The modeling of the transition system as a set of finite states is assumed to be correct, i.e. no measures are taken w.r.t out-of-order data (T5).

Dehkordi et al. [66] does not provide details on the complexity or decidability of any evaluation (T7).

6.2.8 Accumulative Signal Temporal Logic (ASTL) Zhao et al. [67] considers service monitoring of IoT systems. It thereby focuses on Quality-of-Service (QoS) constraints and how one can estimate the satisfaction of these type of constraints. In order to quantify this satisfaction, Zhao et al. [67] adds a robustness function to STL (T1), leading to a new framework named Accumulative STL. As an extension, ASTL inherits most of STL's expressivity, both in terms of temporal and non-temporal relations. Negation $\neg$, conjunction $\wedge$ and disjunction $\vee$ are supported (T3), as well as future-oriented temporal operators $\mathbf{until}_I$, $\boxplus_I$ and $\diamondplus_I$ with interval semantics (T4). Likewise, ASTL inherits the stream representation of a stream as a function over time to a set for formulae $S : T \rightarrow \mathcal{F}$ from STL, where $T = \mathbb{N}$ (T2).

The key innovation in ASTL is the introduction of the robustness function $\hat{\rho}$, which measures the degree to which a given signal ω satisfies an ASTL formula ϕ at a specific time point t. The robustness function assigns a real-valued score to the satisfaction of the formula: high positive values indicate a strong satisfaction of the constraint, while negative values suggest a violation. This approach allows ASTL to handle uncertainties in dynamic, real-time IoT environments, where data is subject to variability and noise (T5).

No complexity and decidability results are provided by Zhao et al. [67] (T7).

6.2.9 Probabilistic Signal Temporal Logic (ProbSTL) Probabilistic Signal Temporal Logic (ProbSTL) [68] extends STL (T1) to better model uncertain or probabilistic events in signal-based systems. It is mainly catered towards the domain of Signal Reasoning, where the goal is to analyse and reason about continuous signals under uncertainty.

ProbSTL focuses on uncertainty in individual statements or conditions, particularly by introducing probability distributions over signals. It extends STL by allowing formulas to express probabilistic constraints, meaning that instead of a rigid Boolean outcome (true or false), the satisfaction of a formula is represented probabilistically, i.e. over interval $[0, 1]$ (T5). For example, ProbSTL allows one to specify constraints like "the probability that the signal stays above a threshold for at least 10 seconds is greater than 0.9.". As a result, ProbSTL is an example of a SR formalism for (probabilistic) reasoning about time (T6).

The most elemental stream in ProbSTL is referred to as a *signal*, modeled as a function from time domain T to a value domain ($\mathbb{D}$) for said signal (T2). ProbSTL assume a discrete time domain $T = \mathbb{N}$. Statements ϕ in ProbSTL are comprised of static operators $\neg$ and $\vee$ and temporal operator $\mathbf{until}_I$, applied to atoms $f_{\mathbb{B}}(l)$, which represent a Boolean evaluation of a statement l. Via recursion, the total set of static operators can be extended to $\neg, \vee, \wedge$ and $\rightarrow$ (T3), whereas the available temporal operators can be extended to $\boxplus_I$ and $\diamondplus_I$, in addition to $\mathbf{until}_I$ (T4).

Tiger and Heintz [68] identifies two factors that determine the complexity of evaluation a ProbSTL statement ϕ. The first is the length of ϕ, i.e. the amount of steps it takes to construct ϕ given the static and temporal operators, and the second complexity factor depends on the properties of the atoms. As a result, satisfiability checking is linear in terms of the length of formula ϕ (T7).

6.2.10 Evolution Temporal Logic (EvTL) Building on the theory of STL, Castiglioni et al. [69] introduces a probabilistic variant of STL (T1) called EvTL. Similar to ProbSTL, EvTL addresses uncertainty that affects the system state as a whole. While ProbSTL deals with uncertainty in individual signal values, EvTL aims to model the probability distribution of data within a streaming system. It evaluates whether the observed distribution of the data aligns sufficiently with a proposed distribution. It models the stochastic behaviour of a stream, rather than the contents of a stream itself (T5). By comparing the observed stochasticity to other distributions, EvTL performs reasoning about time over stochastic information (T6).

A data stream is not modeled explicitly in EvTL. Instead, Castiglioni et al. [69] considers an *evolution sequence*, which is a countable sequence of distributions of the data within a system. Following this, time is implicitly assumed to be discrete, more specifically $\mathbb{N}$, as each element of the sequence can be assigned an index $i \in \mathbb{N}$ (T2).

Statements in EvTL are centered around the atoms, which are expressions which compare the observation distribution of the data to a chosen distribution. To build statement formulae, STL makes use of the STL operators $\neg$, $\vee$ and $\mathbf{until}_I$. Via recursion, the available set of operators in hence $\neg, \vee, \wedge, \rightarrow$ for static operators (T3) and $\mathbf{until}_I, \boxplus_I, \diamondplus_I$ for temporal operators (T4).

Castiglioni et al. [69] does not go into detail regarding complexity or decidability of the EvTL formalism (T7).

6.2.11 Robustness Temporal Logic (RobTL) Robustness Temporal Logic (RobTL) [70] is another extension of STL (T1) designed to deal with uncertainty in systems that process continuous signals, particularly in scenarios where the signals are subject to varying degrees of noise, error, or delay. While formalisms like STL and its variants, such as ProbSTL, focus on modelling specific conditions or probabilistic events, RobTL emphasizes robustness against perturbations of the data, i.e. against out-of-order arrival (T5).

In line with ProbSTL and EvTL, RobTL opts to represent time implicitly by means of a sequence of data states, the aforementioned evolution sequence. As such, time is modeled implicietly as a set $\mathbb{N}$ (T2).

RobTL uses atoms that express the distance between the considered evolution sequence and a perturbation of the sequence. With these atoms, RobTL constructs formulae using the static operators $\neg, \wedge$ and by recursion $\vee$ and $\rightarrow$ (T3), and temporal operator $\mathbf{until}_I$, which in turn induces $\boxplus_I$ and $\diamondplus_I$ (T4). By evaluating these expressions, RobTL performs inference over the temporal relations between behaviour of data within a given system over time, thus performing reasoning about time (T6).

The complexity or decidability of RobTL is not evaluated in Castiglioni et al. [69].

6.2.12 Signal Diagnostic Processing Language (SDRL) Signal Diagnostic Processing Language (SDRL) [71] was proposed to enable the formulation of complex diagnostic tasks to detect anomalies in engineering and maintenance processes at runtime. In contrast to the other SR formalism discussed, SDRL is not based on a pre-existing formalism (e.g. EL for TEL and MEL, STL for ProbSTL, EvTL and RobTL) (T1). The recursively defined signal processing expressions only include temporal operators $\mathbf{next}^i$ and $\mathbf{since}^i$, where $\mathbf{next}^i := \mathbf{next}(\mathbf{next}^{i-1})$. Furthermore, SDRL sports aggregator functions $(\min, \max, \text{avg}, \text{sum})$ for signals. The divergent choice of temporal operators makes it difficult to compare to the previously considered formalisms.

SDRL reasons on signals, which take on different value at different time points. For a single signal s, the value at time point t is defined as $f_s(t)$. As a result, the concept of a stream is thus represented as a function from a time point to a value (T2). No specifics on the time domain are considered in Savkovic et al. [71].

The temporal operators of SDRL express the duration $(duration(\geq, t))$ and how long before or after $(before[t], after[t])$ one signal occurs relative to another. Using these operators, relations akin to $\mathbf{until}_I, \mathbf{since}_I, \boxplus_I, \boxminus_I, \diamondplus_I, \diamondminus_I$ can be expressed (T4). Additionally, SDRL has static operators $\wedge$ and $\neg$, which in turn induce operators $\vee$ and $\rightarrow$ (T3). By evaluating the various temporal expressions across different signals, SDRL performs reasoning about time (T6).

Savkovic et al. [71] encodes SDRL into a custom extension of DatalogMTL, DatalogMTL$_{NR}$ with aggregator functions. Via DatalogMTL$_{NR}$, Savkovic et al. [71] shows fact checking (i.e. checking whether a given signal processing rule is fired) in SDRL is PSPACE-complete. Furthermore it is AC^0 in terms of the size of the signal and ontological data considered (T7).

Within SDRL itself, concepts of uncertainty of the data or out-of-order arrival of the data is not taken into account (T5).

6.2.13 TRIO-based Event Specification Language (TESLA) The CEP language TESLA [72] was introduced with the objective to provide a formal specification of real-time systems, such as the control of industrial processes. It is rooted in the temporal logic TRIO, introduced by Morzenti et al. [73] (T1). TESLA was introduced to overcome the limitations of several CEP and data stream processing languages, notably the lack of rigorous semantics of the former and the limited complex event recognition in the latter.

TESLA assumes an influx of timestamped events, for example `Temp`$(10°)$@12, to arrive in order of ascending timestamps. As such, a data stream is (implicitly) modeled as a sequences of timestamped data. No specific assumptions are made regarding the nature of the time domain (T2).

Central in TESLA are the patterns used to infer a complex event. As a CEP formalism, it performs reasoning about time (T6). The patterns themselves are constructed using connectives `and`, `not` as well as constructions such as `first-within`, `last-within`, k-`within` or `each-within`. Patterns furthermore allow for explicit quantification over time, quoted to be one of TESLA's main strengths [72]. As a result, the static relations expressible by TESLA are $\neg, \wedge, \vee$ and $\rightarrow$ (T3). In terms of temporal relations, the ability to quantify directly over time allows one to construct expression that capture the same relations as $\boxminus_I, \boxplus_I, \diamondminus_I, \diamondplus_I, \mathbf{since}_I, \mathbf{until}_I$, as illustrated in [72] (T4).

As considered above, TESLA assumes an in-order arrival of data. Furthermore, it does not consider any uncertainty regardign the data itself (T5). Lastly, Cugola and Margara [72] does not analyze the complexity or decidability of TESLA (T7).

6.2.14 Temporal Action Logic (TAL) Within the field of Knowledge Representation, we find the class of logics called 'Temporal Action Logics', designed to represent reasoning over such actions and change. TAL [74] has found its way into the field of SR via systems such as DyKnow and TALPlanner [75]. Although it does not build upon existing (non-temporal) formalisms directly, it has been translated into FOL (T1). For its representation of time, TAL does not impose conditions on the time domain except linearity, meaning the most common time domains $(\mathbb{N}.\mathbb{Z}, \mathbb{R}etc.)$ are valid under TAL, but branching time domains are not allowed. The concept of a stream in TAL is most closely embodied by a *narrative*, which is a set of timestamped expressions (T2).

TAL is expressed via a base language, $\mathcal{L}(ND)$, which considers *timed formulae*. These timed formulae are constructed starting from atoms (t, f), which represents

the value of an expression f at time t. Formulae are built using static connectives $\neg, \wedge, \vee$ and $\rightarrow$ (T3), as well as additional operators $\mathbf{until}_I, \boxplus_I$ and $\diamondplus_I$ (T4). Given its ability to express and infer from temporal patterns, TAL performs reasoning about time (T6).However, this explicit quantification induces an increase in reasoning complexity, because formulas are expressed in full first-order logic with arithmetic constraints over time variables. This means that satisfiability and entailment require handling nested quantifiers, variable instantiations, and constraint solving, which significantly raises the computational burden compared to propositional or modal temporal logics.

Within TAL, the data at hand is strictly assumed to be correct and correctly time-stamped. Therefore, TAL does not incorporate handling of uncertainty of data or of the data stream into account (T5).

Doherty and Kvarnström [74], nor the later work of Doherty et al. [75] explore the theoretical complexity or decidability of the formalism (T7).

6.3 SR Query Languages

Query languages play an important role in SR by enabling efficient querying and filtering of streaming data in real time. They focus on retrieving information rather than complex reasoning. However, they are included in our survey as selecting relevant data based on specified conditions can itself be seen as a form of reasoning. This perspective aligns with the broader understanding of SR as discussed in Bonte et al. [5].

As mentioned before, the domain of SR is strongly embedded in the Semantic Web domain. With the prominent position of SPARQL as query language for RDF data, it is unsurprising that SPARQL has frequently been chosen as a basis for a query language over RDF streams. As covered in 3.6, stream querying formalisms retrieve data from a stream by means of specific queries, which stipulate conditions to be met. Once retrieved by the query engine, batches of data, for example induced by windowing, can be reasoned upon using non-temporal reasoning engines (e.g. DL or Datalog reasoners). As stream querying formalisms do not perform explicit reasoning, complexity and decidability results, when provided, are not comparable to other SR formalisms discussed. As such, we do not consider dimension T7 in the discussion of this category.

6.3.1 Continuous SPARQL (C-SPARQL) Introduced in 2009, C-SPARQL [76] takes inspiration from earlier stream query languages, such as CQL [77], and applies it to the RDF-specific SPARQL language (T1). C-SPARQL considers RDF Streams $S = \left(\langle{:}s_i{:}p_i{:}o_i\rangle, t_i\right)$ of RDF triples $\langle{:}\, s : p : o\rangle$, timestamped with $t_i \in T$. The time domain T from which the timestamps are taken can be chosen freely. In the examples of Barbieri et al. [76], RDF literals of type `:xsd:dateTime` are used. Note that the RDF stream is ordered by its timestamps in a non-decreasing manner. The RDF stream can be infinite in length. In order to query the infinite stream, C-SPARQL employs time- and tuple-based windows (T2).

In the querying of the data, C-SPARQL provides several operators in order to specify the desired data. Using the `WHERE` clause, conditions can be imposed on the data to subsequently match these conditions to specific instances. The implicit conjunction of conditions within the `WHERE` clause, in addition to SPARQL keywords `NOT` and `UNION`, allows one to express negation $\neg$, conjunction $\wedge$ and disjunction $\vee$ of various conditions (T3). InC-SPARQL, the windows of the queries are the main way of extracting the desired temporal data. Beyond the window operators, C-SPARQL does not provide explicit means to express temporal operators [78] (T4).

With the introduction of RDF data streams and syntax for windowing and aggregation, C-SPARQL enables us to reason over data that arrives over time, in contrast to the static data of (base) SPARQL. Via the timestamp function and `WHERE` clause, C-SPARQL allows to retrieve data based on temporal patterns, thus allowing for reasoning about time, where the reasoning is encapsulated inside the query conditions (T6). Our study did not provide us with papers that research the computational complexity of C-SPARQL (T7). Furthermore, C-SPARQL does not have dedicated techniques to handle uncertainty of data or out-of-order arrival of data (T5).

6.3.2 Streaming and Temporal ontology Access with a Reasoning-based Query Language (STARQL) Özçep et al. [32] observed existing stream data management systems at the time employed a "bag of assertions" semantics, i.e. these systems pooled incoming information together in a " bag" and disregarded their chronological order. As a result these systems suffer some loss of (temporal) information. STARQL was introduced [32] to resolve this, building on a basis of SPARQL (T1). STARQL's semantics are defined over *synchronous streams*. These synchronous streams are represented by an infinite sequence of chronologically ordered timestamped atoms, where the atoms take on the form of DL-Lite ABox assertions adn the timestamps stem form a linear timeline $T(,<)$. STARQL proceeds by applying time-based window operators to this infinite sequence to extract a *temporal Abox*, a set of timestamped Aboxes, which is subsequently converted to a finite set of (non-temporal) Aboxes (T2). This allows for classical SPARQL queries within a single ABox, while enabling temporal reasoning, i.e. reasoning about time, by performing inter-ABox reasoning (T6). Similar to C-SPARQL, STARQL is equipped with conjunction, disjunction and negation (T3). keywords `FORALL` and `EXISTS` provide ways to universally or existentially quantify. The `HAVING` clause allows to define conditions for inter-ABox reasoning, thus allowing to impose conditions that mimic $\boxplus, \boxminus, \diamondplus, \diamondminus$ to be imposed on the data queries (T4). For example, $\boxplus A(x)$ can be modeled as `HAVING FORALL i > NOW IN SEQ: A(x)`, where `SEQ` is the sequence used to represent the data stream.

STARQL assumes the data to be true, i.e. without uncertainty, and to arrive in order. It does not consider methodology for handling data uncertainty in its formalism (T5).

6.3.3 SPARQL$_{stream}$ SPARQL$_{\text{stream}}$ is an extension to SPARQL 1.1 (T1) proposed by Calbimonte et al. [79], designed to provide ontology-based access to streaming data sources stored in relational models. Similar to C-SPARQL, it represent streams by means of RDF streams $S = \left(\langle{:}\, s_i :\right.$

$p_i : o_i\rangle, t_i\big)$ of RDF triples $\langle : s : p : o\rangle$. Windowing operators over time allow for information extraction from a data stream. Compared to C-SPARQL, $\text{SPARQL}_{\text{stream}}$ only allows for time-based windows, argumenting tuple-based windows may not be able to capture graph patterns spanning multiple triples (T2). $\text{SPARQL}_{\text{stream}}$ inherits the static relations from SPARQL, i.e. $\wedge$, $\neg$ and $\vee$ as well as quantification (T3). Despite the representation of streams as sets of time-stamped triples, Calbimonte et al. [79] makes no note of being able to express temporal relations on the data, suggesting the window operators work in a "bag of assertions" fashion, which leaves one unable to identify temporal relations between data points within the same window (T4). As a result, $\text{SPARQL}_{\text{stream}}$ is used for stream querying and can , in conjunction with an RDF reasoner, be used for reasoning over time (T6). $\text{SPARQL}_{\text{stream}}$ does not tackle problems concerning uncertainty in data or data streams (T5).

6.3.4 Continuous Temporal SPARQL (CT-SPARQL) Although with C-SPARQL one can reason about data that arrives in the form of a stream, reasoning about temporal relations between the data is not accommodated, i.e. C-SPARQL is able to reason *over* time, but not *about* time. Argumented by Yang [78], C-SPARQL windowing causes the data to lose its (time-)sequential nature, as windows transform the section of the RDF stream into an RDF graph without time annotation. CT-SPARQL aims to retain the sequential nature using a SEQ relation between events [78]. Furthermore, it breaks down an RDF stream into multiple indexed RDF graphs, representing snapshots over time.

CT-SPARQL shares its SPARQL foundation (T1), stream representation (T2) and static expressivity (T3) with C-SPARQL. The expression 'A `SEQ` B' captures the tuples (A_i, B_j) where A_i and B_j are events of type A and B respectively and B_j is the first of type B to come after A_i. Yang [78] state any complex temporal relation can be constructed using the `SEQ` operator, with a formal discussion thereof left for future work. No further work of said author has been published which considers such discussion (T4). Despite temporal expressivity being not yet formally explored, the `SEQ` operator provides a basis for derivation of temporal patterns. As a result, CT-SPARQL is able to derive data which matches temporal patterns, i.e. reasoning about time (T6).

Uncertainty in data or data streams are not addressed within the CT-SPARQL framework (T5).

6.3.5 ETALIS Language for Events (ELE) ELE, introduced by Anicic et al. [80], defines the formal syntax and semantics of Event-Driven Transaction Logic Inference System (ETALIS), a system for CEP. It is rooted in a Logic Programming (LP)-approach to CEP (T1), thus oriented towards reasoning about time (T6). It represents streams by means of *event streams* ϵ, a mapping of grounded event predicates to a set of time points in $\mathbb{Q}^+$, indicating at which time points a given grounded event predicate. As a result, the time domain for ELE is $\mathbb{Q}^+$ (T2). It supports the static operations of $\wedge$, $\vee$ and $\neg$ via constructs `AND`, `OR` and `NOT` (T3). For its temporal relations, it relies on Allen's interval algebra (see Allen [81]). Allen's interval algebra uses 13 temporal relations, e.g. `SEQ`, `DURING`, `FINISHES` and others, to describe the temporal ordering and dependency of events specifically. ELE is stated to support all of the temporal relations of Allen's interval algebra (T4). Temporal patterns akin to the temporal relations considered in T4 can be constructed using constructins with Allen's interval relations, e.g. $A\mathbf{until}_{[5,10]}B$ aligns with expression A `DURING` (0 `SEQ` (B `DURING` (0 `SEQ` 5))).

ELE itself does not capture uncertainty of data or out-of-order problems with event streams. Outside of the specification of the ELE formalism, Anicic [34] considers extensions of the ETALIS system which addresses out-of-order arrival algorithmically (T5).

6.3.6 EP-SPARQL The order of events is of substantial importance for CEP applications, as complex events are oftentimes defined by a specific sequence of (basic) events. Event Processing SPARQL (EP-SPARQL) [11] is a SPARQL-based language, grounded in the operational semantics of ELE [34], (T1) which goes beyond just the `SEQ` operator of CT-SPARQL. In line with the SPARQL (SPARQL)-based formalisms above, EP-SPARQL employs RDF streams $S = \big(\langle : s_i : p_i : o_i\rangle, t_i\big)$ and supports time- and tuple-based windows (T2). It furthermore inherits its static expressivity from pure SPARQL, i.e. $\wedge$ via juxtaposition and $\vee$ via `UNION`, but no `NOT` construct. Anicic et al. [11] notes negation can nonetheless be expressed by combination of the `OPTIONAL` and `FILTER` clauses (T3). In terms of temporal expressivity, EP-SPARQL uses four binary temporal operators: `SEQ`, `EQUALS`, `OPTIONALSEQ` and `OPTIONALEQUALS`. Additionally, new functions `getDURATION()`, `getSTARTTIME()` and `getENDTIME()` allow to specify conditions on duration, start time and end time of a specified event pattern (T4). Further research is required to assess if and how the temporal operators can be used to express temporal patterns such as $\boxplus_I$, $\diamondsuit_I$, $\mathbf{since}_I$ and others.

With the above, EP-SPARQL is able to extract data from RDF streams based on temporal pattern matching, inducing a form of reasoning about time (T6). The data itself is implicitly assumed to be correct and to be arriving in-order, hence no methods for data uncertainty are integrated in EP-SPARQL (T5)

6.3.7 Continuous Query Evaluation over Linked Streams (CQELS) C-SPARQL and its derivatives above are engineered towards querying of streams, but provide no integrated data processing capabilities [40]. As a result, the processing engine in charge of the windowing and the query engine in charge of answering queries are decoupled. To avoid the overhead involved in the communication between the two separate engines, the SPARQL-based framework (T1) of CQELS [40] was proposed to serve as a unified theoretical framework for modelling, querying and processing RDF Streams and (static) data. Similar to C-SPARQL, an RDF stream is represented as a set of timestamped RDF triples over a time domain of $\mathbb{N}$. A window is represented by a function $\omega : \mathbb{N} \to 2^{\mathbb{N}}$ mapping a time point to a set of time points. CQELS is equipped to handle both time-based and triple-based window operators (T2). It supports conjunction $\wedge$ and disjunction $\vee$ via juxtaposition and UNION ($\cup$) respectively (T3). Aside from selection using window operators, CQELS does not support temporal operators (T4). As a result, CQELS only performs reasoning over time (T6).

Phouc [40] does not address challenges of data uncertainty or possible out-of-order arrival of data (T5).

*6.3.8 RSP-QL and RSP-QL** Dell'Aglio et al. [82] proposes RSP-QL as overarching model for multiple RSP systems, including the aforementioned C-SPARQL and CQELS. The motivation behind RSP-QL's development is to address the limitations and inconsistencies between different RSP models. With its role as an overarching abstract model for other RSP systems, RSP-QL is a formalism in the most abstract sense, as it is not a language itself, but rather an abstraction model that can be shaped into multiple different languages. Simultaneously, it can be seen as significant step towards standardisation of RSP systems. RSP-QL extends the formalism of SPARQL (T1). It queries RDF streams $S = \big(\langle : s_i : p_i : o_i \rangle, t_i\big)$ over time domain $T = \mathbb{N}$. Windows are modeled using a general mapping from a stream S to a set of RDF triples, with further specification for both time- and tuple-based windows (T2). For static expressivity, RSP-QL inherits the conjunction $\wedge$ and disjunction $\vee$ from SPARQL (T3). It does not provide means to express temporal relations, focussing only on retrieval of the temporal data (T4). As a result, RSP-QL is oriented to enable reasoning over time, but not about time (T6).

RSP-QL does not provide built-in methods for handling uncertainty in data (T5). Following this observation Keskisärkkä et al. [83] introduces the extension RSP-QL* with statement-level annotations for RDF statements. These annotations can serve a variety of purposes, most notably as measure of uncertainty or fuzziness. To realize these annotations, one employs nesting of triple patterns as seen in RDF* and SPARQL* [84].

6.4 ASP-based Formalisms

ASP-based formalisms excel in SR scenarios that require non-monotonic reasoning and dynamic adaptation, as they naturally support rule-based modeling where new information can override previous assumptions. This makes them particularly effective for handling evolving knowledge and conflicting constraints in streaming environments.

6.4.1 C-ASP Pham et al. [85] considers several limitation of C-SPARQL and CQELS with regards to SR, most notably their limited ability to handle non-determinism and 'default' values, i.e. values that could be overwritten later. To resolve these shortcomings, they propose the ASP-based reasoning system (T1), (C-ASP). C-ASP takes RDF Streams $(\langle : s : p : o \rangle, t)$ for input. eh exact time domain is not specified by Pham et al. [85]. Using time- and tuple-based windows, triples are extracted. On the selected triples, C-ASP applies a set of ASP rules via the ASP reasoner Clingo and subsequently outputs a new RDF stream.

Within the ASP rules C-ASP follows the ASP-core2 language standard for ASP rules [86]. As a result, C-ASP has static expressivity of $\wedge$ and $\neg$ within its rules. In this case, recursion can not be applied to obtain $\vee$. The $\rightarrow$ is captured by the concept of a rule itself (T3). Due to its reliance on non-temporal ASP rules (T4), a triple's timestamp can not be taken into account in the entailment, thus C-ASP can not reason about time, only over time (T6).

As considered above, the ASP approach to non-deterministic data one form of data uncertainty, i.e. where conclusions are not 'final' and can be rescinded by future information. Pham et al. [85] illustrates this approach to tackle incomplete information problems (T5).

Complexity results of reasoning using C-ASP are not considered in Pham et al. [85] (T7).

6.4.2 Logic-based framework for Analytic Reasoning over Streams (LARS) and its extensions C-ASP has demonstrated the ability to process data at high frequencies, which suggests scalability. However, as stressed by Della Valle et al. [2], a theoretical basis for SR needs to find a balance between expressivity and scalability. With the aim of providing formal semantics for SR, LARS, was introduced by Beck et al. [18], rooted in ASP (T1).

Streams are modelled by means of a function $v : T \subseteq \mathbb{N} \rightarrow \mathcal{P}(\mathcal{G})$, which projects a time $t \in T$ onto a set of atoms of the form $p(c_1, \ldots, c_n)$, with p a predicate and c_i constants. The time domain T is fixed to be $\mathbb{N}$. Furthermore, LARS introduces an abstract definition of a window on a stream as a function that projects a stream S and time point $t \in \mathbb{N}$ to a substream $S' \subseteq S$. Beck et al. [18] further stipulates time-based, tuple-based and partition-based windows. The latter cuts several snippets out of a given streams and pastes them together into a new stream (T2).

Using formulae, LARS is able to denote and assert meaningful statements about the data within a stream. With atoms $p(x_1, x_n)$, with p a predicate and x_i a constant or variable, LARS formulae are inductively defined using the static operators $\neg$, $\wedge$, $\vee$ and $\rightarrow$ (T3) and the temporal operators $\Box$, $\Diamond$ and $@_t$. LARS empplys absolute semantics for the temporal operators, i.e. $\Box$ and $\Diamond$ express "always" and "once" considered over the whole timeline, independent of the time point at whihc the formula is evaluated (T4). Additionally, LARS provides two window operators to be used in the formulae, denoted in Beck et al. [18] by $\boxplus^w$ and $\triangleright$, integrating window operators into the formulae to create a substream of an originally considered stream. With the ability to express a wide range of temporal patterns, LARS is equipped to perform reasoning about time (T6), but does not account for data uncertainty (T5).

In terms of complexity, Beck et al. [18] discusses both model checking and model satisfiability. First for LARS formulae, the monotone core of the LARS language, Beck et al. [18] notes both model checking and satisfiability are PSPACE-complete in case of grounded formulae. Model checking and satisfiability diverge in case of non-grounded formulae: model checking remains PSPACE-complete, whereas satisfiability increases to NExpTime-completeness. Regarding the LARS programs for non-monotonic reasoning, grounded programs likewise have a model-checking and satisfiability complexity of PSPACE-complete, as well as model checking for non-grounded LARS programs. Satisfiability increases to $\text{NExpTime}^{\text{NP}}$ for non-grounded programs. Via restrictions imposed upon the window operators, the complexity can be reduced further. For details we refer to Beck et al. [18]. Each of these complexity results assume the necessary window evaluation is executable in polynomial time (T7).

LARS imports the LTL operators $\Box$ and $\Diamond$. However, in combination with the LARS window operator $\boxplus^w$, temporal relations akin to the (more granular) MTL operators $\boxplus_I$,

$\boxminus_I$, $\diamondplus_I$, $\diamondminus_I$ can be deduced. This is shown in Funk [33], who subsequently proposed a fragment of MTL called 'Metric LARS'. Metric LARS captures the MTL formulae that can be mapped to LARS formulae. This mapping shows LARS also has access to the MTL operators, with relative semantics, in addition to the absolute temporal operators $\Box$ and $\Diamond$ (T4). Furthermore, complexity bounds can be derived via MTL. From results presented in Alur and Henzinger [87], it follows that satisfiability checking of Metric LARS formulae is `ExpTime`-complete (T7).

Intent on use cases with uncertain and probabilistic data, Eiter and Kiesel [88] imports ideas from Weighted Logic by expanding the classical Boolean evaluation of LARS formulae to an evaluation over a semiring* $\mathcal{R}$, referred to as Weighted LARS (wLARS). By extending the truth values of a formula from a Boolean $\{0, 1\}$ to a (larger) semiring, wLARS tailors to several quantitative challenges, such as probabilistic and preferential reasoning, as well as reasoning under weak constraints (T5).

As an extension of LARS, the `PSAPCE`-completeness of LARS provides a lower bound of `PSPACE`-hardness for evaluating wLARS expressions, as any choice of semiring gives raise to an evaluation problem at least as complex as the Boolean case. Via a restriction of wLARS to 'efficiently encodeable' semirings, more accurate complexity bounds can be obtained. For details, we refer to Eiter and Kiesel [88] (T7).

6.4.3 Probabilistic ASP (PrASP) Moving away from LARS, a different approach to ASP-based (T1) probabilistic reasoning is given by Nickle and Mileo [89] in the form of PrASP, which allows for non-monotonic probabilistic reasoning. Here, formulae (FOL formulae or AnsProlog rules) are annotated with point-valued probabilities (T5). PrASP works in conjunction with CQELS and performs reasoning over RDF streams, represented via mapping $S = (\langle : s_i, : p_i : o_i \rangle, t_i)$ (T2). As PrASP uses FOL syntax, the static expressivity is given through operators $\neg, \wedge, \vee$ and $\rightarrow$ (T3), and no temporal operators are included (T4). As a result, PrASP is suited for probabilistic reasoning over data streams, but not about time (T6). Among the publications considered within this survey, none cover complexity results for PrASP (T7).

6.5 Miscellaneous Formalisms

The subsections above collect several formalism which start from a common basis, i.e. DL, temporal logic, query languages and ASP. There is, however, a significant portion of SR formalisms which cannot be allocated to one of such categories or warrant to be seen as a category of its own. The lack of a category does not imply the formalism in question is less significant to the field of SR.

6.5.1 Temporal Datalog DatalogMTL has not been the only endeavour which extends Datalog for SR. Temporal Datalog [90] extends the syntax of (negation-free) Datalog (T1) with time constants and time variables, in addition to the 'object' constants and variables. Cruz-Filipe et al. [90] stipulates the ability of Temporal Datalog to formulate hypothetical answers, i.e. answers based on currently available data that can be overturned by future data, as a means of resolving issues created by out-of-order arrival (T5). Streams are represented as a dataset of atoms $P(c_1, \ldots, c_n)$, where c_i are constants of which at most one (say c_n) is a temporal constant that serves as a timestamp, taken from $\mathbb{N}$. The set of atoms, and thereby the stream, can be infinite in nature. Window operators are not defined for Temporal Datalog (T2). Temporal Datalog relies on Datalog rules of the form $\alpha \leftarrow \alpha_1 \wedge \ldots \alpha_n$ where α, α_i are expressions $P(x_1, \ldots, x_n)$ with x_i constants or variables. As a result, Temporal Datalog only considers static operators $\wedge$ and $\rightarrow$ (T3). However, Cruz-Filipe et al. [90] propose an extension which adds negation $\neg$ to the rule body. By recursion, disjunction $\vee$ can also be expressed in this extension. Temporal Datalog does not accommodate temporal operators directly (T4). However, Temporal Datalog does nonetheless provide a means for reasoning about time (T6), via the use of temporal constants and variables. Cruz-Filipe et al. [90] provides no complexity results for Temporal Datalog (T7).

6.5.2 Event Calculus for Run-Time reasoning (RTEC) In order to perform event recognition at run-time, Artikis et al. [91] introduces RTEC. RTEC is a dialect of Event Calculus (T1), a formalization by Kowalsi and Sergot [92] based the Horn fragment of FOL. A stream is represented as a sequence of events over integer timeline $\mathbb{Z}$, where the elements are (simple) events represented by predicate expressions. A windowing mechanism is applied to these streams, with only time-based windows of uniform length WM (T2). Inference is done in a rule-based manner where rule bodies are formed by conjunctive predicate expressions. As such, RTEC only supports conjunction $\wedge$. Opposite predicates, e.g. $\neq$ as opposite of the predicate $=$, may serve as negation is specific cases, but no general negation operator is available in RTEC (T3). The interval predicates of RTEC follow the expressivity of Event Calculus. In terms of our temporal expressivity dimension T4, near all temporal relations can be modelled. `holdsFor` can be used to model $\bigcirc$ or $\bullet$, the operators that refer to the next and previous time point respectively. Likewise, a combination of `holdsAt` and `terminatedAt` can model **since** or **until** relations (T4). Oriented towards CEP, RTEC is capable of reasoning about time (T6). There is no accomodation for data uncertainty handling considered (T5).

Artikis et al. [91] does not report on the complexity of RTEC in terms of fact entailment or satisfiability (T7). The focus of its complexity results lie in the interval manipulation introduced by RTEC to improve the overall reasoning efficiency. For details, we refer to Artikis et al. [91].

RTEC has been extended by future work In the later works of Mantenoglou et al. [93] and Mantenoglou and Artikis [94], aimed at resolving challenges regarding cyclic temporal dependencies of events. Hpwever, as these works focus on the algorithms rather than the underlying formalism, the expressivity is not impacted directly.

*A set with addition $+$ and multiplication$\times$, where both $+$ and $\times$ are associative and have a neutral element, $+$ is commutative and $\times$ is distributive over $+$.

7 Survey Results: Usage Dimension

A formalism is in the first place a theoretical description. Studying this theoretical side provides valuable insight into the properties of the formalism, but SR is by nature an application-oriented field. putting the SR formalisms into practice is where a formalism's added value can truly be put to the test. Testing every SR formalisms considered in this study goes beyond the scope of this literature study. Instead, we resort to three metrics as indication of a formalism's applicability: the amount of citations a given formalism has generated (U1), the amount of prototype implementations of the formalism (U2) and the amount of application-based papers that build upon the formalism (U3). The latter two are distilled from the "secondary study", where the full set of considered papers is revisited. For further details, we refer to subsection 2.2. The results across these dimensions are summarized in Table 9 and discussed in detail in the following subsections.

7.1 DL-based Formalisms

The category of DL formalisms has comparatively fewer citations compared to the other formalism categories (U1). Lécué [50], $\mathcal{EL}++$ ontology streams accumulated a total of 18 citations since 2012. The DL Lite TQL data streams proposed by Klarman and Meyer [47] have 8 citations since 2013. The formalism $\text{LTL}_{\mathcal{EL}}^{\mathbf{X},\mathbf{G}}$ has only been cited once thus far.

Among the papers considered in our survey, none introduce application-independent software prototypes (U2).

Several real-life use cases using a DL formalism were identified (U3), each of which employ the description logic $\mathcal{EL}^{++}$ with ontology streams. Via a multitude of traffic-related data sources on traffic in the city of Dublin, Ireland, Lécué et al. [110] diagnose traffic congestion, Lécué and Pan [111] perform knowledge discovery on bus delays Chen et al. [112] and counteract concept drift within bus delay predictions [112]. Within the same work, Chen et al. [112] employ the same principles for concept drift to a different application, i.e. air quality predictions in the city of Beijing, China based on factors such as humidity, wind speed and direction, and air pollutants.

7.2 Temporal Logic Formalisms

In terms of academic citations (U1), we observe the following results for temporal logic-based formalisms; The work of Wałęga et al. [53] on DatalogMTL and the fragmentsDatalogMITL and DatalogMTL^{FP}, has accumulated 60 citations since 2019. Fragments DatalogMTL$\neg$ [54] and $\text{DatalogMTL}^{\exists}$ [56] have received 33 and 0 citations since 2021 and 2023 respectively. MSTL [58] and P-MTL [61] received 22 and 30 citations respectively. For the EL-based formalisms of TEL and MEL, we observe a citation count of 57 for TEL [63]. The subsequent works on extending TEL with past operators, i.e. Aguado et al. [64] and with metric operators, i.e. MEL [65], have gathered 6 and 9 citations respectively. ProbSTL [68] has been cited 37 times since 2010. With a total of 329 citations, the work of Cugola and Magara [72] on TESLA has gathered significant attention in the research community since publication in 2010. Furthermore, the work of [75] on TAL has accumulated 59 citations since 2013. The remaining temporal logic-based formalisms fall within the single-digit citation range. For a detailed overview, we refer to Table 8.

Several formalisms have seen software implementations (U2). TAL is supported by two notable prototypes: TALPlanner, a temporal planner, and DyKnow, a middleware for stream reasoning from multiple sources [75, 99]. DatalogMTL has an implementation in the form of MeTeoR [95]. EvTL is implemented via SPEAR for ULTRON[†], which extends the SPEAR tool for model checking [97]. TEL has been realized in the temporal ASP solver Telingo[‡] [96]. Finally, DOTR combines the RDFox store with the T-Rex engine for temporal reasoning based on TESLA [98].

The secondary study identified three distinct use cases (U3) employing temporal logic-based formalisms. One of these focuses on monitoring the sensors within Unmanned Aerial Vehicle (UAV) control systems, where HDRC3 [114] uses TAL to ensure correct semantics in planning components such as TALPlanner and TFPOP, modeling constraints as TAL statements to eliminate unsafe or infeasible plans. SDRL is applied in two industrial IoT scenarios within the work of Kharlamov et al. [113], monitoring the sensors of trains and of turbines, encoding diagnostic programs more efficiently than previous rule languages. The work of Zhoa et al. [67] illustrates the ability of ASTL to combine qualitative and quantitative constraints via a case study within the domain of sensor monitoring.

7.3 SR Query Languages

When it comes to academic popularity, multiple query languages rank highly in terms of citations (U1). Most notably are CQELS, EP-SPARQL, published in 2011, and ELE, published in 2012, with respectively 558, 547 and 244 citations. The work of Barbieri et al. [39] on C-SPARQL has only been cited 372 times since 2009. However, we note that several different publications in the years 2009-2010 may serve as potential source for citing C-SPARQL. In addition to the publication Barbieri et al. [39] (372 citations) ,two conference publications [10, 142] (251 and 197 citations) and a journal publication [143] (381 citations) may each serve as a citation source for C-SPARQL. RSP-QL's introductory publication [82] has received 164 citations since 2014 and its extension RSP-QL* [83] has accumulated 17 citations since 2019. The formalisms of $\text{SPARQL}_{\text{stream}}$, STARQL and CT-SPARQL have been cited considerably less, with citation counts of 15, 24 an 1 since 2013, 2014 and 2018.

Several query-based formalisms have been introduced alongside their software implementations (U2). C-SPARQL was introduced together with its engine [39], and the term "C-SPARQL" is used interchangeably for both the language and the engine throughout the literature. Similarly, CQELS was introduced with its native engine [40]. EP-SPARQL and ELE are both implemented in the ETALIS system [102], which serves as their primary implementation. Additional engines building on ETALIS include Retalis [103],

[†] https://github.com/vale-castiglioni/Ultron
[‡] https://github.com/potassco/telingo

Formalism	Year of publication	Citations	Software Prototypes
DL-Lite TQL and fragments	2013	8	
$\text{LTL}_{\mathcal{EL}}^{\mathbf{X,G}}$	2021	1	-
DL $\mathcal{EL}^{++}$ + ontology stream	2012	20	-
DatalogMTL, DatalogMTL^{FP}	2019	60	[95]
DatalogMTL$^{\neg}$	2021	33	-
DatalogMTL$^{\exists}$	2023	0	-
MSTL	2016	24	-
P-MTL	2016	33	-
PMLTL	2024	1	-
TEL	2007	57	[96]
TEL + past operators	2017	6	-
MEL	2022	9	-
LTPAL	2020	1	-
ASTL	2022	8	-
ProbSTL	2020	37	-
EvTL	2022	0	[97]
RobTL	2022	4	-
SDRL	2018	1	-
TESLA	2010	329	[98]
TAL	2013	59	[75, 99]
C-SPARQL	2009	372	[39]
STARQL	2014	24	[100]
SPARQL_stream	2013	15	[101]
CT-SPARQL	2018	1	-
ELE	2012	244	[102–104]
EP-SPARQL	2011	547	[102, 105]
CQELS	2011	587	[40]
RSP-QL	2014	164	[21]
RSP-QL*	2019	17	[83]
C-ASP Language	2019	14	[85]
LARS	2015	242	[106–108]
wLARS	2020	26	-
Metric LARS	2021	0	-
PrASP	2014	20	-
Temporal Datalog	2020	7	-
RTEC	2012	59	[109]

Table 8. Summary of citations (U1) and software prototypes (U2) of SR formalisms

μCEP [104] and μReasoner [105], . Retalis integrates ELE with Synchronized Logical Reasoning Language (SLR) [144] and is available on GitHub[§]. μCEP explicitly refers *the ETALIS language*, which most likely implies μCEP is a software implementation forELE. i.e. Anicic et al. [12]. We therefore consider it a software implementation for ELE. Diwold et al. [105] notes EP-SPARQL as language within ETALIS. As a result, we consider μReasoner a software implementation for EP-SPARQL. STARQL has been implemented in the EXASTREAM system [100], which is part of the OptiqueVQS framework [145]. $\text{SPARQL}_{\text{stream}}$ relies on the morph-stream implementation[¶] [101]. As an abstract reference model, RSP-QL does not produce a materialized engine itself but provides standardization for RSP engines. To this end, the RSP4J library [21] has been developed. RSP-QL* has a software prototype in the RSPQLStarEngine, available on GitHub[‖] [83]. For CT-SPARQL, Yang [78] only notes a design architecture for an implementation AIMRS, without reference to an existing implementation.

Within our secondary study, a total of 30 use case studies were found that build upon query-based formalisms (U3). These use cases span several different formalisms, most notably C-SPARQL, CQELS, EP-SPARQL, STARQL and $\text{SPARQL}_{\text{Stream}}$. The majority of 14 use cases employ C-SPARQL, followed by 4 for ELE, 4 for RSP-QL, 3 for $\text{SPARQL}_{\text{stream}}$, 2 for EP-SPARQL, 2 for STARQL and 1 for CQELS.

With regards to C-SPARQL, a significant portion of its use cases focuses on human activity tracking; we observe several medical and E-health applications [120, 121], sport and athleticism applications [122–124] and more generally the monitoring of person location and activity [125, 126]. Furthermore, our secondary study contained two applications in the social media domain [118, 119], two in sensor monitoring [115, 116] and one in industrial

[§] https://github.com/procrob/Retalis
[¶] https://github.com/jpcik/morph-streams
[‖] https://github.com/keski/RSPQLStarEngine

Formalism	Urban Traffic	Sensor Monitoring	Smart Grid	Industrial Production	Social Media	Human Activity Tracking	Maritime Traffic	Robotics	Cloud Computing
DL $\mathcal{EL}^{++}$ ontology streams	[110–112]	[112]	0	0	0	0	0	0	0
ASTL	0	[67]	0	0	0	0	0	0	0
SDRL	0	[113]	0	0	0	0	0	0	0
TAL	0	[114]	0	0	0	0	0	0	0
C-SPARQL	0	[115, 116]	0	[117]	[118, 119]	[120–126]	[127]	[128]	0
STARQL	0	[100, 129]	0	0	0	0	0	0	0
$\text{SPARQL}_{\text{stream}}$	[130]	[130, 130]	0	0	0	0	0	0	0
ELE	0	0	[131]	[104]	0	0	0	[103, 132]	0
EP-SPARQL	0	0	[105]	[117]	0	0	0	0	0
CQELS	0	0	0	0	0	[133]	0	0	0
RSP-QL	0	[134]	0	[135, 136]	0	[137]	0	0	0
LARS	0	0	0	0	0	0	0	0	[138]
RTEC	[91]	0	0	0	0	0	[139–141]	0	0

Table 9. Summary of dimension U3, listing the applications per formalism, grouped per application domain. Formalisms without application-based papers in the considered literature are omitted from the table.

production [117], maritime traffic monitoring [127] and robotics [128] respectively. CQELS has been used by Ali et al. [133] as a basis for an IoT-enabled meeting management application. Several use cases employ the ETALIS system, and thereby either ELE or EP-SPARQL, for CEP. Xu et al. [131] considers smart grid applications, more specifically for efficient energy consumption in office buildings, which uses ELE as their underlying formalism. Likewise, use cases in robotics [132] and industrial production [117] show the applicability of ELE and EP-SPARQL respectively to robotics and industrial automation. Other applications of ELE include the robotics use case considered in Ziafati et al. [103] and the industrial production setting considered in Ren et al. [104]. STARQL-based systems have been applied to monitoring large networks of appliances, illustrated by Siemens gas and steam turbines [100, 129, 146]. In Calbimonte [130], $\text{SPARQL}_{\text{stream}}$ has been used in sensor monitoring monitoring, such as for flood detection networks and environmental hazard forecasting platforms, and urban traffic monitoring, in the form of a bike sharing platform. Lastly, several applications allow for any RSP-QL-compliant language to be used within their system. These include the DIVIDE system for E-Health monitoring via IoT devices [137, 147] and Streaming MASSIF, used for sensor monitoring to gauge COVID-19 risk in office settings [134]. Additionally, the human activity tracking application considered in Calvaresi and Calbimonte [148] is RSP-QL-compliant. The works Charbonnier et al. [135] and Charbonnier et al. [136] employ a system build upon RSP4J for quality assurance in industrial production.

7.4 ASP-based Formalisms

When considering the academic citations (U1), we observe for the ASP-based formalisms that PrASP has received 20 citations since 2014, C-ASP 14 and LARS 242. In addition, wLARS and Metric LARS account for 26 and 0 citations. C-ASP, wLARS and Metric LARS are thereby relatively recent, dating between 2019 and 2021. The work of Beck et al. [149] on LARS was published in 2015.

The most notable ASP-based formalism in terms of developed software prototype appears to be LARS (U2). Within our secondary study, three software implementations of LARS were found; Ticker[**] [106], Laser[††] [107] and Laser2[‡‡] [108]. The chronologically first implementation of Ticker employs the ASP solver Clingo to repeatedly solve a static version of the LARS program at hand. Laser is a subsequent implementation (in Python), optimized to avoid duplicate computations. As a result, Bazoobandi et al. [107] reports a significant difference in run-time between Ticker and Laser. Laser2 further optimizes Laser by identifying rules that can, over a finite time interval, not be satisfied. Bazoobandi et al.[108] reports an improved run-time over Laser. The C-ASP language is implemented by Pham et al. [85] into a C-ASP engine, which uses an internal Clingo ASP solver as seen in LARS' Ticker. Nickles and Mileo [89] states the development of an (early-stage) prototype of PrASP, but does not provide details on the public availability of said prototype.

With regards to applications (U3) of the ASP-based formalisms considered in Section 6.4, we note that PrASP was introduced by Nickles and Mileo [89] in 2015, LARS in 2017 and C-ASP in 2019. As a result, each of the covered ASP-based formalism have been introduced recently relative to the other formalisms. The DL- and query-based formalisms considered in the previous two subsection were introduced between the years 2009 and 2014. The

[**] https://github.com/hbeck/ticker
[††] https://github.com/karmaresearch/Laser
[‡‡] https://bitbucket.org/hrbazoo/laser/src/master/

community has subsequently had more time to develop (and report on) use case cases based on these formalisms than on the aforementioned ASP-based formalisms. This is subsequently reflected in the amount of use cases found in our secondary study, as only one ASP-based use case built was found: Beck et al. [138] implements LARS for intelligent administration of content-centric networks by leveraging LARS' expressivity in caching strategies.

7.5 Miscellaneous Formalisms

Regarding academic citations (U1), the formalisms of Temporal Datalog, introduced in 2020, and RTEC, introduced in 2012, have gathered 7 and 59 citations respectively.

Artikis et al. [91] notes RTEC has been implemented, available on GitHub [109] . No software implementations for Temporal Datalog were observed in the considered literature.

The software prototype of RTEC is subsequently put to use in both the domain urban traffic monitoring and in the domain of maritime traffic monitoring. The application in the former domain stems from the introductory paper of Artikis et al. [91] as a primary proof-of-concept on the traffic of Helsinki. RTEC has thereafter been implemented in several scenario's, aimed at recognizing the activity of one or more vessels. RTEC is used to model the event patterns used to monitor the various activities a ship can partake in [139–141]

8 Discussion

Within this work, we set out to provide a systematic overview of the theoretical underpinning of SR. We observe that the propositions for SR formalisms have been numerous, but the highly diverse demands and explored directions have left the academic field in an unstructured state. Based on the directions indicated by the literature, we have outlined a series of expectations upon which SR formalisms, both old and new, can be assessed. The existing SR formalisms were subsequently evaluated across these different dimensions in order to obtain a global overview of the research results obtained thus far. This overview helps to highlight the capabilities of existing technologies as well as to identify research gaps.

The resulting overview of prerequisites for a SR formalism as presented in Section 5 aims to serve as a guideline for both past and future formalisation efforts. The structure we have presented here serves as a first step towards a 'comparative framework' of various SR formalisms. By means of the proposed dimensions, one is able to more strongly argument a choice for a particular formalism, by formally comparing different options for a given use case. Furthermore, it allows to position a newly introduced formalism within the current state-of-the-art.

Throughout the surveying process, several elements naturally emerged as common traits shared by all or most SR formalisms. First, it stands out that not every single formalism has the same objective when it comes to SR. the term 'Stream Reasoning' can be interpreted in different ways by different formalisms. Some formalisms are only aimed at intelligently extracting and delivering data from streams, whereas others aim to extensively process and derive complex conclusions from such data. This large diversity is highly determinant for a formalism's capabilities. As such we classify it via dimension T6. Although not unique to SR, most formalisms have their roots in pre-existent theories, which can range from mathematical theories (e.g. set theory, FOL) to existing reasoning formalisms. We capture the roots of the various SR formalisms in the dimension T1, labelled "theoretical basis". Furthermore, we consider the representation of a stream. All formalisms have some way of representing streams, either explicitly or implicitly, as the subject of their operations. Given the high variability observed and practical implications of different choices, this shaped into our dimension T2 "stream representation". Another element is the ability to relate different pieces of information to one another. In order to deduce meaningful insights from data, one cannot look at every piece of data in a vacuum, but has to consider how the different information can be connected to one another. The wide variety to relate different pieces of information, both with and without taking account the temporal aspect, has lead to dimensions T3 and T4, "static relations" and "temporal relations" respectively. In addition to these expressive aspects, many SR settings must contend with imperfect, incomplete or disorderly data, which motivates dimension T5. This dimension captures whether a formalism can accommodate uncertainty, probabilistic information or out-of-order arrival in the stream, an increasingly important consideration given the volatility and heterogeneity of real-world data sources.

Coming back to our research questions considered in Section 2, we can thus answer **Q1**: "What are the common constructs and expression of a formalism for SR?". Based on the different formalisms introduced over the past 15 years, we have identified five elements that are common to most SR formalisms; a specific objective (T6) a theoretical basis (T1), a representation of streams (T2) and methods of expressing relations between pieces of information, either time-agnostic (T3) or time-dependent (T4).

In the second research question **Q2**, "What are sensible criteria to evaluate a logical formalism, in particular a formalism's suitability for SR?" , we set out identify the criteria upon which a SR formalism can be evaluated. In this search for tangible comparative metrics, the dimensions that best fit this description are dimensions T7, the complexity of a formalism, and U1, U2 and U3, the dimensions that describe the usage and popularity of a formalism. Additionally, dimensions T3 en T4 also lend themselves to this evaluation; T3 and T4 chart the expressivity of a formalism, which allows to compare different formalisms in terms of their capabilities.

Recall research questions **Q3** and **Q4**, "Given the criteria from the previous question, how do recently developed SR formalisms meet these criteria and how do these formalisms compare to one another?" and "Among the formalisms that have been developed since *'It's a Streaming World'* [2], which have been adopted in real-life applications ?" (Q4). There we have asked ourselves how the formalisms introduced in the past 15 years score on the various dimensions, and which ones have been translated to real-life applications. A detailed breakdown for each SR formalism is given in the sections 6 and 7 for the theoretical and usage dimensions respectively. For each of the dimensions, the results can be summarized as follows.

T1. Dissection of the existing state-of-the-art of SR formalisms according to dimension T1, the theoretical basis of the formalism, shows a relatively diverse spectrum, with several clusters as shown in Table 4. The field of SR is not dominated by a single mathematical paradigm upon which all formalisms are built. Instead, we observe different clusters such as formalisms based on DL, on ASP, on query languages and other. Each of these clusters stems from different backgrounds and application areas (i.e. ASP from non-monotonic reasoning, DL being tied to Semantic data), but each of the observed areas has grown towards supporting reasoning over streams in their own manner. It is this diversity that allows the field of SR to be applied to many different application areas.

T2. In terms of dimension T2, stream representation, we considered the different approaches to modelling streams in the context of SR, as summarized in Table 6. First, the representation of time itself shows three dominant options in specifying the time domain; (a) a discrete domain, such as $\mathbb{N}$ or $\mathbb{Z}$, (b) dense time domain $\mathbb{Q}$ and (c) the time domain is not specified. In addition, MSTL, TAL andSTARQL have very little assumptions on the time domain. This could allow for more exotic time representations, sch as branched timelines, but examples thereof have not yet been explored within the literature. Second, on the representation of a stream, many of the covered formalisms rely on the model of an RDF stream as a set of time-stamped RDF triples. This is the case for many of the query languages, such as C-SPARQL, CQELS and ELE, as well as for the answer set based PrASP and C-ASP formalisms. Other formalisms prefer to not annotate each piece of information individually, but group the information per time point. This leads to a chronological sequence of sets of information. Examples of this approach can be found in DL Lite TQL and its fragments, TEL and MEL, Temporal Datalog and $\mathcal{EL}++$ ontology streams. A last common approach consists of describing a data stream as a mapping from a given time point to a set of information that holds at said time point. This is seen in DatalogMTL (and its fragments), LARS (and its fragments and extensions), STARQL and ProbSTL. For most observed formalisms, the concept of a stream in not restricted to finite streams and therefore naturally supports the idea of endlessly continuing data flows, in line with real-world streaming environments. Exceptions exist in formalisms such as PMLTL and LTPAL, which operate over finite timed traces or finite transition systems. Lastly, in terms of windowing, we observe that the query-based formalisms support this practice the most, with time-based windows being the most popular mechanism. In addition, LARS also supports windowing. With LARS, the windowing is incorporated into the syntax itself via a dedicated window operator, incorporating the concept of windows deeper into the reasoning process.

The high diversity in time and stream representation allows one to pick the representation most suitable to an intended use case. Simultaneously, this diversity can be an obstacle when attempting to communicate between systems of different formalisms, in particular those with different stream representations.

T3, T4. When it comes to expressivity, we can observe from Tables 4 and 5 that nearly all of the considered formalisms are able to express the basic concepts of conjunction, disjunction and negation, with or without relying on recursion. Among all static relations, it stands out that conjunction $\wedge$ is the most prominent, with every observed formalism supporting it (albeit via recursion in the case of ProbSTL and EvTL). Negation appears to not be as vital as conjunction, with many but not all formalism supporting it. Most notably, the ASP-based formalisms consistently support negation as the only category considered, which can be linked to their disposition towards non-monotonic reasoning. A more prominent disparity can be found in their temporal expressivity. We observe the whole spectrum, ranging from none (e.g. PrASP, $\mathcal{EL}++$ streams) to roughly all of our specified temporal relations (e.g. MEL, DatalogMTL, EP-SPARQL). For each of the different categories aside from the 'remainder' category, at least one SR formalism meets a fair amount of our predetermined criteria. Among DL-based formalisms, (unfragmented) DL-Lite TQL posseses its own variations of $\mathbf{since}$, $\mathbf{until}$, $\boxplus$, $\boxminus$, $\diamondplus$ and $\diamondminus$. Likewise for query-based formalisms, C-SPARQL, EP-SPARQL and ELE allow for various complex temporal expressions either via direct timestamp specification or via temporal comparatives. When it comes to temporal logic-based formalisms, the best scoring formalisms are TEL and TESLA, along with MEL and DatalogMTL for those that support interval semantics. Within the ASP-based formalisms, LARS and its extensions are the only ones equipped with temporal expressivity. When comparing the best formalisms within each category, we observe that the query-based languages provide the most extensive tools for temporal expressions. None of the query languages does however support interval semantics. DatalogMTL, on the other hand, does. DatalogMTL only compromises on 'next' and 'previous' operators, which by themselves only apply to discrete time domains (or otherwise strictly ordered sets of information). LARS contrasts with DatalogMTL by employing an absolute time semantics, compared to the relative semantics of DatalogMTL; the operators $\Box$ and $\Diamond$ are used instead of $\boxplus$, $\boxminus$, $\diamondplus$ and $\diamondminus$. Similar to dimension T2, the variability in expressivity lends itself well to tailored solutions and closed systems, but can become an obstacle for open-ended systems containing multiple different formalisms. This even extends to formalisms with seemingly identical operators, e.g. DatalogMTL's $\boxplus$, $\boxminus$ and MEL's $\boxplus$, $\boxminus$, due to differences in underlying semantics.

T5. When it comes to handling various types of discrepancy in data streams, we observe in Table 7 that there exists only limited support in the current literature. The formalisms that account for data uncertainty are mostly oriented towards probabilistic uncertainty, as is the case for temporal logic formalisms P-MTL, ProbSTL and EvTL. Three formalisms, being RSP-QL*, wLARS and PrASP, support a versatile approach to tackling uncertainty, as they can be used to express probability, as well as fuzziness or other other uncertainty measures. Among the considered formalisms, only RobTL explicitly incorporates the handling of out-of-order data into the formalism.

T6. Across the surveyed landscape, a clear division can be observed with respect to a formalism sets out to do: (i) stream querying, (ii) reasoning over time, or (iii)

reasoning about time. Query languages (e.g., C-SPARQL, STARQL, CQELS, RSP-QL) are engineered primarily for continuous, windowed retrieval and filtering of evolving RDF data, yet carefully crafted queries can sometimes emulate simple temporal inferences; this makes them broadly adoptable but typically limits temporal semantics to what can be encoded as selection conditions over windows. In contrast, logics like DatalogMTL, TEL/MEL, TESLA, TAL, and LARS target inference that is intrinsically temporal; operators such as **since**, **until**, $\boxplus$, $\diamondplus$ etc. are part of the language itself, so the "when" becomes a first-class citizen alongside the "what". A third group (e.g., EL++ ontology streams) performs incremental, non-temporal updates to knowledge bases as data arrives, without committing to temporal pattern semantics, useful for change tracking and explanation. From a larger perspective, we observe a degree of sequentiality in the objectives of the various formalisms; stream querying formalisms were introduced relatively early, primarily between 2009 and 2014. This compared to formalisms for reasoning about time, of which the majority was introduced after 2016. An explanation for this contrast can be found in the sequentiality of data processing: before complex reasoning about time can be considered, one must first be able to retrieve, i.e. query, said data. As a result, SR query languages were introduced first, Furthermore, reasoning over time is more common for earlier SR formalisms, whereas later formalisms move towards reasoning about time, with more detailed temporal relations (see T4) as well. As a result, there is a noticeable trend towards more complex reasoning over the past 15 years. In addition, many SR formalisms are motivated by concrete needs within specific application domains rather than from purely theoretical motivation. The broader categories as well as several stand-alone formalisms each reflect this orientation, with examples such as TAL for modelling actions and change, SDRL for industrial diagnostics and LTPAL for interpreting knowledge produced by AI systems.

T7. A synopsis of the complexities of the SR formalisms covered in this work is hindered by sparsity. the amount of SR formalisms for which complexity results were retrieved, either through their introductory paper(s) or subsequent papers, is limited. Across the limited results, we observe that both DatalogMTL, SDRL and LARS are shown to be `PSPACE`-complete with regards to fact checking, suggesting `PSPACE`-completeness is not merely characteristic for these three formalisms, but more generally of reasoning about time.

U1–U3. The theoretical side of SR formalisms does not necessarily reflect their added value to real-life applications. From the results of our study into the formalism's applications and academic reception, summarized in Tables 9 and 8, we observe first that the two metrics of citations and applications exhibit largely the same patterns: formalisms with many applications tend to also have a large amount of citations, and formalisms without observed applications also tend to have a lower citation count. The most notable example thereof is C-SPARQL, with both a high amount of applications (7) as well as citations (372). Nonetheless, we must not ignore several promising SR formalisms who do not (yet) have a noteworthy amount of applications, but whose citation rate indicates significant traction in the SR community. Most notably, the collections of LARS and DatalogMTL formalisms have received a discerning amount of citations in the time since their publication in 2015 and 2018 respectively, which forebodes a growing impact on future SR developments.

Furthermore, the existence and availability of software prototypes does not necessarily translate to a high amount of applications. TAL and C-SPARQL are examples of formalisms with both software prototypes and a high amount of applications, but on the contrary LARS, TEL and TESLA do have software prototypes, but do not boast a significant amount of applications. Reasons for the lack of a one-on-one relation between software availability and observed applications may include, but are not limited to, the narrow scope of our secondary study, the properties of the formalisms themselves or the recency of introduction of a formalism.

Looking at the usage trends across the different categories of SR formalisms, we observe that the category of query-based SR formalisms outranks the other categories in terms of both applications and citations. As such, we conclude that query-based formalisms have thus far been more popular. A possible explanation for this popularity lies in their intended use; the focus on querying data streams makes these formalisms applicable to a wide range of applications. However, the "reasoning" within these formalisms is often-times limited to reasoning by means of query constructions. This in contrast to other, especially logic-based, SR formalisms which employs rule constructs to perform reasoning. As the query-based formalisms are not strictly tailored towards reasoning, but more broadly to any type of stream querying (and processing), the metrics for the usage dimension of observed applications and academic citations is skewed in favour of the query-based formalisms.

Aside from the theoretical properties of SR formalisms, we have studied the adoption of the covered formalisms towards real-life use cases. To this end, we considered the implementations seen in the total set of papers we have taken into account for this survey. This set yielded a large diversity of application domains, illustrating the wide applicability of SR as a field. Between the formalisms themselves, we have observed that some formalisms have seen a significantly higher implementation rate than others, indicating that several formalisms are favoured by the (research) industry over others. In examining the adoption of the surveyed SR formalisms across real-life applications, we observe that query-based formalisms stand out with a substantially higher number of use cases than other categories. Several factors contribute to this trend. First, many query languages are comparatively mature: C-SPARQL, EP-SPARQL, STARQL, CQELS and $\text{SPARQL}_{\text{stream}}$ were introduced between 2009 and 2014, providing nearly a decade or more for the research community to explore implementations and deploy these systems in practical settings. In contrast, a number of more novel formalisms such as LARS, SDRL and DatalogMTL were introduced later, between 2015 and 2018, and therefore have had fewer years in which to be translated to real-world applications. As such, numerical comparisons between formalism adoption must be interpreted in light of these differing timelines. Lastly, we note that the category

of query formalisms typically functions at a low level within the applications; the continuous queries defined in these languages often operate close to the source of the streams, such as the direct monitoring of IoT devices, health parameters or small sensors (e.g. [115, 131, 133]). As such, the primary objective of the formalisms in these applications is the retrieval of specific data. The requirements of querying on data streams are possibly less high compared to more high-level reasoning on streaming data that is targeted by other formalisms such as ASP- or temporal logic based SR formalisms.

In addition to overall popularity, we also observe that several formalisms are strongly represented in one or more application domains. ELE has found its way into the robotics and smart grid domains, STARQL's use cases are each focused on monitoring industrial processes and C-SPARQL is a common choice for various forms of human activity tracking. This shows in part that different formalisms tailor to the needs of different fields of application, thus that SR is by no means limited to select cases or areas. Looking at such formalisms individually, it is only natural a formalism sees increasing use in one or more domains; the development of said formalism could have been driven by a need in a specific area, resulting in a evident candidate field for real-life applications. Alternatively, a prominent standalone use case can emphasise a formalism's suitability for a certain application area, inducing subsequent use cases. The presence of such a specialization towards certain use cases can thus be seen as an asset towards further SR development.

9 Conclusion & Future Perspectives

As discussed in Section 4, several parts of the SR ecosystem have already been the subject of benchmarking and comparative studies. Most notably, RSP systems have been surveyed with a focus on technical properties such as architecture, operational semantics and supported data formats [41]. Complementary system-level perspectives, such as those of Margara et al. [17] and Dell'Aglio et al. [17], have further clarified requirements and helped bring SR out of isolated case studies and towards broader applicability. Building on these efforts, the dimensions outlined in this work serve as a first step towards a comparison framework specifically targeted at SR formalisms, enabling an objective analysis at the level of syntax, semantics and reasoning objectives.

The current state of the art shows significant progress in expressivity; however, support for different kinds of uncertainty remains limited. Where uncertainty is addressed, it is often through specialised extensions tailored to particular niches. Extending more general formalisms to accommodate incomplete, noisy or out-of-order data remains an important avenue for future work. At present, wLARS and RSP-QL* stand out as prominent examples of such extensions, and similar directions could be explored for other formalisms.

In recent years, several SR formalisms with strong temporal capabilities have emerged, enabling increasingly complex reasoning. To translate these theoretical advances into practice, the field would benefit from a greater number and variety of real-world deployments. DatalogMTL and LARS are among the strongest formalisms from a theoretical standpoint, making them prime candidates for complex applications. In parallel, it is worth exploring how query-based formalisms can support richer reasoning patterns that go beyond straightforward stream querying. Formalisms that operate over RDF streams are especially well-positioned for interoperability in domains such as IoT, edge processing and decentralised ecosystems. It is up to the SR community to identify the most promising opportunities for demonstration and implementation of these advances.

Despite the high level of abstraction of the above formalisms, there is no single formalism that covers all requirements. Our analysis shows that different formalisms exhibit different strengths and that these strengths are closely tied to their primary objectives. Query-oriented approaches prioritise windowed access to evolving data, which favours scalability and deployment but limits temporal expressivity. Temporally expressive logics enable rich pattern detection and causality, yet they often carry higher computational costs and stricter modelling assumptions. Non-monotonic rule frameworks excel at revising conclusions as streams evolve, but their semantics can be difficult to reconcile with snapshot-based query evaluation. In several cases, objectives can even hinder one another. For example, grafting MTL interval operators and stratified negation from DatalogMTL$^\neg$ onto C-SPARQL would require reasoning across window boundaries and fixpoint strata, which undercuts C-SPARQL's snapshot-over-windows execution model.

In light of these trade-offs, the creation of a single SR formalism that "checks all the boxes" across objectives, expressivity and complexity appears highly unfeasible. A more practical route for building SR systems that support multiple objectives is to construct mappings between state-of-the-art formalisms, so that different components specialise in complementary tasks. For instance, one subsystem can provide robust stream querying and window management, another can perform temporal rule-based inference and a third can handle uncertainty. Crucial to this approach is the correctness of the mappings: translations must preserve meaning across languages so that temporal relations, static operators and uncertainty annotations align semantically rather than merely syntactically. The dimensional analysis provided in this work (T1–T7) offers a principled basis for designing and validating such mappings, by making explicit which constructs and semantics must be preserved when moving between formalisms and where objective-driven constraints are likely to interact.

The field of SR has undergone a period of substantial growth, yielding a wide range of ideas and tools. While this diversity underscores SR's broad applicability, it can also appear overwhelming to new researchers and external stakeholders. In this light, the usage dimensions offer a practical guide: C-SPARQL, EP-SPARQL and EL++ ontology streams emerge as prominent options with a high degree of adoption, which likely reflects their versatility and level of abstraction across many use cases. A deeper understanding of the theoretical capabilities that underpin such formalisms will aid the development of SR systems that are both expressive and accessible. The systematic review of the state of the art enables us to identify research gaps that warrant further attention. As the field

has been largely application-driven, we highlight common, cross-cutting requisites, including temporal expressivity, flexible windowing and the treatment of probabilistic and fuzzy data. Several aspects remain comparatively underexplored, such as delayed or out-of-order arrival and its implications for temporal reasoning. Although delayed data can be mitigated algorithmically, for example through buffering or caching, its theoretical implications can only be fully understood when accompanied by sufficiently formal semantic treatments.

We hope this work will provide a basis for new researchers to participate in the formal aspects of SR and support seasoned researchers in future efforts. By synthesising 15 years of developments in SR formalisms, we aim to provide a stable point of reference and structure for ongoing research. Open challenges remain, but recent advances also show the field's potential. In particular, LARS and DatalogMTL exhibit extensive temporal expressivity. Future work may focus on showcasing these capabilities through impactful real-life applications.

Funding

This research is funded by the FWO Project FRACTION (Nr. G086822N).

References

1. IDC. Worldwide global datasphere structured and unstructured data forecast, 2024–2028. Technical report, International Data Corporation, 2024.
2. Della Valle E, Ceri S, Van Harmelen F et al. It's a streaming world! reasoning upon rapidly changing information. IEEE Intelligent Systems 2009; 24(6): 83–89.
3. Bonte P and Tommasini R. Streaming linked data: A survey on life cycle compliance. Journal of Web Semantics 2023; 77: 100785.
4. Ceri S, Gottlob G and Translating S. Into relational algebra: Optimization, semantics, and equivalence of sql queries, software engineering. IEEE Transactions 1985; 11(4): 324–345.
5. Bonte P, Calbimonte JP, de Leng D et al. Grounding stream reasoning research. Transactions on Graph Data and Knowledge 2024; 2: 1–47.
6. Brereton P, Kitchenham BA, Budgen D et al. Lessons from applying the systematic literature review process within the software engineering domain. Journal of systems and software 2007; 80(4): 571–583.
7. Martín-Martín A, Orduna-Malea E, Thelwall M et al. Google scholar, web of science, and scopus: A systematic comparison of citations in 252 subject categories. Journal of informetrics 2018; 12(4): 1160–1177.
8. Barbieri D, Braga D, Ceri S et al. Stream reasoning : Where we got so far. In Proceedings of the 4th Workshop on New Forms of Reasoning for the Semantic Web: Scalable & Dynamic.
9. Falzone E, Tommasini R and Della Valle E. Stream reasoning: From theory to practice. In Reasoning Web International Summer School. Springer, 2020. pp. 85–108.
10. Barbieri DF, Braga D, Ceri S et al. Querying rdf streams with c-sparql. ACM SIGMOD Record 2010; 39(1): 20–26. DOI: 10.1145/1860702.1860705.
11. Anicic D, Fodor P, Rudolph S et al. EP-SPARQL: a unified language for event processing and stream reasoning. Proceedings of the 20th international conference on World wide web 2011; DOI:10.1145/1963405.1963495.
12. Anicic D, Rudolph S, Fodor P et al. Stream reasoning and complex event processing in ETALIS. Semantic Web 2012; .
13. Balduini M, Della Valle E, Dell'Aglio D et al. Social listening of city scale events using the streaming linked data framework. In international semantic web conference. Springer, pp. 1–16.
14. Margara A, Urbani J, Van Harmelen F et al. Streaming the web: Reasoning over dynamic data. Journal of Web Semantics 2014; 25: 24–44.
15. Ye J, Dasiopoulou S, Stevenson G et al. Semantic web technologies in pervasive computing: A survey and research roadmap. Pervasive and Mobile Computing 2015; 23: 1–25.
16. Roffia L, Morandi F, Kiljander J et al. A semantic publish-subscribe architecture for the internet of things. IEEE Internet of Things Journal 2016; 3(6): 1274–1296.
17. Dell'Aglio D, Della Valle E, van Harmelen F et al. Stream reasoning: a survey and outlook: A summary of ten years of research and a vision for the next decade. Data Science 2017; 1(1-2): 59–83.
18. Beck H, Dao-Tran M and Eiter T. LARS: A logic-based framework for analytic reasoning over streams. Artificial Intelligence 2018; 261: 16–70.
19. Eiter T, Ogris P and Schekotihin K. A distributed approach to LARS stream reasoning (system paper). Theory and Practice of Logic Programming 2019; 19(5-6): 974–989.
20. Noor MHM, Salcic Z and Wang KIK. Ontology-based sensor fusion activity recognition. Journal of Ambient Intelligence and Humanized Computing 2020; 11(8): 3073–3087.
21. Bonte P, Tommasini R, Valle ED et al. RSP4J: Towards an unifying API for RDF Stream Processing. openreview.net, 2021.
22. Weiss W and D'Mello C. Fundamentals of model theory. Topology Atlas, 2000.
23. W3C. RDF 1.2 semantics. https://www.w3.org/TR/rdf12-semantics/, 2024. W3C Recommendation.
24. W3C. Owl 2 web ontology language document overview. https://www.w3.org/TR/owl2-overview/, 2012. W3C Recommendation.
25. Calvanese D, De Giacomo G, Lembo D et al. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. Journal of Automated reasoning 2007; 39(3): 385–429.
26. Van Benthem J, Van Eijck J and Kooi B. Logics of communication and change. Information and computation 2006; 204(11): 1620–1662.
27. Marek VW and Truszczyński M. Stable models and an alternative logic programming paradigm. In The logic programming paradigm: A 25-year perspective. Springer, 1999. pp. 375–398.
28. Eiter T, Ianni G and Krennwallner T. Answer set programming: A primer. Springer, 2009.
29. Gebser M, Kaufmann B, Kaminski R et al. Potassco: The potsdam answer set solving collection. Ai Communications 2011; 24(2): 107–124.
30. Pnueli A. The temporal logic of programs. In 18th Annual Symposium on Foundations of Computer Science (SFCS 1977). IEEE, pp. 46–57.

31. Koymans R. Specifying real-time properties with metric temporal logic. Real-time systems 1990; 2(4): 255–299.
32. Özçep O, Möller R and Neuenstadt C. OBDA stream access combined with safe first-order temporal reasoning. Technical report, Hamburg Univ of Technology 2014; .
33. Funk NM. Real time logics and reasoning answerset semantics for metric temporal logic. PhD Thesis, TU Wien, 2021.
34. Anicic D. Event Processing and Stream Reasoning with ETALIS. PhD Thesis, Karlsruher Institut für Technologie (KIT), Karlsruhe, Germany, 2012.
35. Maler O and Nickovic D. Monitoring temporal properties of continuous signals. In Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-30206-3, pp. 152–166.
36. Abiteboul S, Hull R and Vianu V. Foundations of Databases. Addison-Wesley, 1995. ISBN 0-201-53771-0.
37. W3C. RDF 1.2 concepts and abstract syntax. https://www.w3.org/TR/rdf12-concepts/, 2024. W3C Recommendation.
38. W3C. Sparql 1.1 query language. https://www.w3.org/TR/sparql11-query/, 2013. W3C Recommendation.
39. Barbieri DF, Braga D, Ceri S et al. C-SPARQL: SPARQL for continuous querying. In Proceedings of the 18th international conference on World Wide Web. pp. 1061–1062.
40. Phuoc DL. A native and adaptive approach for linked stream data processing. PhD Thesis, NUI Galway, 2013.
41. Bonte P, Callé C, Curé O et al. Languages and systems for RDF stream processing, a survey: P. bonte et al. The VLDB Journal 2025; 34(4): 50.
42. Bonte P, Ongenae F and Tommasini R. A holistic view over ontologies for streaming linked data. Semantic Web 2024; 15(5): 2005–2033.
43. Isah H, Abughofa T, Mahfuz S et al. A survey of distributed data stream processing frameworks. IEEE Access 2019; 7: 154300–154316.
44. Zadeh LA. Fuzzy sets. Information and control 1965; 8(3): 338–353.
45. Hájek P. Metamathematics of fuzzy logic, volume 4. Springer Science & Business Media, 2001.
46. Calbimonte JP, Mora J and Corcho O. Query rewriting in RDF stream processing. In Proceedings of the 13th International Conference on The Semantic Web. Latest Advances and New Domains-Volume 9678. pp. 486–502.
47. Klarman S and Meyer T. Prediction and Explanation over DL-Lite Data Streams. In International Conference on Logic for Programming Artificial Intelligence and Reasoning. pp. 536–551. DOI:10.1007/978-3-642-45221-5_36.
48. Gutiérrez-Basulto V and Klarman S. Towards a unifying approach to representing and querying temporal data in description logics. In International conference on web reasoning and rule systems. Springer, pp. 90–105.
49. Tirtarasa S and Turhan A. Towards reverse engineering temporal queries: Generalizing EL concepts with Next and Global. In Della Valle E, Eiter T, Le Phuoc D et al. (eds.) Informal Proceedings of the Stream Reasoning Workshop. p. 14.
50. Lécué F. Diagnosing changes in an ontology stream: A DL reasoning approach. Proceedings of the AAAI Conference on Artificial Intelligence 2012; 26(1): 80–86.
51. Baader F, Brandt S and Lutz C. Pushing the EL envelope. In Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005). pp. 364–369.
52. Brandt S, Kalaycı EG, Ryzhikov V et al. Querying log data with metric temporal logic. Journal of Artificial Intelligence Research 2018; 62: 829–877.
53. Wałęga PA, Kaminski M and Grau BC. Reasoning over streaming data in metric temporal datalog. Proceedings of the AAAI Conference on Artificial Intelligence 2019; 33(01): 3092–3099.
54. Cucala DJT, Wałęga PA, Grau BC et al. Stratified negation in datalog with metric temporal operators. Proceedings of the AAAI Conference on Artificial Intelligence 2021; 35(7): 6488–6495.
55. Cabalar P, Dieguez M, Schaub T et al. Towards metric temporal answer set programming. Theory and Practice of Logic Programming 2020; 20(5): 783–798.
56. Nissl M. Temporal Reasoning in Knowledge Graphs. Artificial Intelligence Systems for Reasoning with Time in Vadalog. PhD Thesis, Technische Universität Wien, 2023.
57. Leone N, Manna M, Terracina G et al. Efficiently computable datalog∃ programs. KR 2012; 12: 13–23.
58. de Leng D and Heintz F. Qualitative spatio-temporal stream reasoning with unobservable intertemporal spatial relations using landmarks. Proceedings of the AAAI Conference on Artificial Intelligence 2016; 30(1).
59. Randell DA, Cui Z and Cohn AG. A spatial logic based on regions and connection. In Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning. pp. 165–176.
60. Wałęga PA, Grau BC, Kaminski M et al. DatalogMTL: computational complexity and expressive power. In Proceedings of the 28th International Joint Conference on Artificial Intelligence. pp. 1886–1892.
61. Tiger M and Heintz F. Stream reasoning using temporal logic and predictive probabilistic state models. In 2016 23rd International Symposium on Temporal Representation and Reasoning (TIME). IEEE, pp. 196–205.
62. Aurandt A, Jones PH, Rozier KY et al. Multimodal model predictive runtime verification for safety of autonomous cyber-physical systems. In International Conference on Formal Methods for Industrial Critical Systems. Springer, pp. 220–244.
63. Cabalar P and Pérez Vega G. Temporal equilibrium logic: a first approach. In International Conference on Computer Aided Systems Theory. Springer, pp. 241–248.
64. Aguado F, Cabalar P, Diéguez M et al. Temporal equilibrium logic with past operators. Journal of Applied Non-Classical Logics 2017; 27(3-4): 161–177.
65. Cabalar P, Diéguez M, Schaub T et al. Metric temporal answer set programming over timed traces. In International Conference on Logic Programming and Nonmonotonic Reasoning. Springer, pp. 117–130.
66. Dehkordi AH, Alizadeh M and Movaghar A. Linear Temporal Public Announcement Logic: a new perspective for reasoning the knowledge of. arXiv preprint arXiv:200903793 2020; .
67. Zhao D, Zhou Z, Cai Z et al. ASTL: Accumulative STL with a novel robustness metric for IoT service monitoring. IEEE Transactions on Mobile Computing 2022; 22(10): 5751–5768.

68. Tiger M and Heintz F. Incremental reasoning in probabilistic signal temporal logic. International Journal of Approximate Reasoning 2020; .
69. Castiglioni V, Loreti M and Tini S. EvTL: A Temporal Logic for the Transient Analysis of Cyber-Physical Systems. arXiv preprint arXiv:220413357 2022; .
70. Castiglioni V, Loreti M and Tini S. RobTL: A Temporal Logic for the Robustness of Cyber-Physical Systems. arXiv preprint arXiv:221211158 2022; .
71. Savkovic O, Kharlamov E, Xiao G et al. Theoretical characterization of signal diagnostic processing language. In Proceedings of the 31st International Workshop on Description Logics, volume 2211. CEUR Workshop Proceedings.
72. Cugola G and Margara A. TESLA: a formally defined event specification language. In Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems. pp. 50–61.
73. Morzenti A, Mandrioli D and Ghezzi C. A model parametric real-time logic. ACM Transactions on Programming Languages and Systems (TOPLAS) 1992; 14(4): 521–573.
74. Doherty P and Kvarnström J. Temporal action logics. Foundations of Artificial Intelligence 2008; 3: 709–757.
75. Doherty P, Heintz F and Kvarnström J. Robotics, temporal logic and stream reasoning. In LPAR (short papers). pp. 42–51.
76. Barbieri DF, Braga D, Ceri S et al. Continuous queries and real-time analysis of social semantic data with C-SPARQL. In Workshop on Social Data on the Web. pp. 1–12.
77. Arasu A, Babu S and Widom J. The CQL continuous query language: semantic foundations and query execution. The VLDB Journal 2006; 15(2): 121–142.
78. Yang X. Query for streaming information: dynamic processing and adaptive incremental maintenance of RDF stream. In Companion Proceedings of the The Web Conference 2018. pp. 843–847.
79. Calbimonte JP, Corcho O and Gray AJ. Enabling ontology-based access to streaming data sources. In The Semantic Web–ISWC 2010: 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I 9. Springer, pp. 96–111.
80. Anicic D, Fodor P, Rudolph S et al. A rule-based language for complex event processing and reasoning. In International Conference on Web Reasoning and Rule Systems. Springer, pp. 42–57.
81. Allen JF. Maintaining knowledge about temporal intervals. Communications of the ACM 1983; 26(11): 832–843.
82. Dell'Aglio D, Della Valle E, Calbimonte JP et al. RSP-QL semantics: A unifying query model to explain heterogeneity of RDF stream processing systems. International Journal on Semantic Web and Information Systems (IJSWIS) 2014; 10(4): 17–44.
83. Keskisärkkä R, Blomqvist E, Lind L et al. RSP-QL: Enabling statement-level annotations in RDF streams. In International Conference on Semantic Systems. Springer, pp. 140–155.
84. Hartig O. Foundations of RDF* and SPARQL*:(an alternative approach to statement-level metadata in RDF). In AMW 2017 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, June 7-9, 2017., volume 1912. Juan Reutter, Divesh Srivastava.
85. Pham TL, Ali MI and Mileo A. C-ASP: continuous asp-based reasoning over RDF streams. In International Conference on Logic Programming and Nonmonotonic Reasoning. Springer, pp. 45–50.
86. Calimeri F, Faber W, Gebser M et al. Asp-core-2 input language format. CoRR 2019; abs/1911.04326. URL http://arxiv.org/abs/1911.04326. 1911.04326.
87. Alur R and Henzinger TA. Real-time logics: complexity and expressiveness. Information and Computation 1993; 104(1): 35–77.
88. Eiter T and Kiesel R. Weighted LARS for quantitative stream reasoning. ECAI 2020 2020; DOI:10.3233/FAIA200160.
89. Nickles M and Mileo A. Web stream reasoning using probabilistic answer set programming. In International Conference on Web Reasoning and Rule Systems. Springer, pp. 197–205.
90. Cruz-Filipe L, Nunes I and Gaspar G. Hypothetical answers to continuous queries over data streams. Proceedings of the AAAI Conference on Artificial Intelligence 2020; 34(03): 2798–2805.
91. Artikis A, Sergot M and Paliouras G. Run-time composite event recognition. In Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems. pp. 69–80.
92. Kowalski R and Sergot M. A logic-based calculus of events. New generation computing 1986; 4: 67–95.
93. Mantenoglou P, Pitsikalis M and Artikis A. Stream reasoning with cycles. Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning 2022; 19(1): 544–553.
94. Mantenoglou P and Artikis A. Extending the range of temporal specifications of the run-time event calculus. In 31st International Symposium on Temporal Representation and Reasoning (TIME 2024). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 6–1.
95. Wang D, Hu P, Wałęga PA et al. Meteor: Practical reasoning in datalog with metric temporal operators. Proceedings of the AAAI Conference on Artificial Intelligence 2022; 36(5): 5906–5913.
96. Cabalar P, Kaminski R, Morkisch P et al. telingo= asp+ time. In International Conference on Logic Programming and Nonmonotonic Reasoning. Springer, pp. 256–269.
97. Loreti M and Castiglioni V. Spear for ultron. https://github.com/gitUltron/Ultron, 2021.
98. Margara A, Cugola G, Collavini D et al. Efficient temporal reasoning on streams of events with DOTR. European Semantic Web Conference 2018; : 384–399.
99. De Leng D and Heintz F. Dyknow: A dynamically reconfigurable stream reasoning framework as an extension to the robot operating system. In 2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR). IEEE, pp. 55–60.
100. Kharlamov E, Mailis T, Mehdi G et al. Semantic access to streaming and static data at Siemens. Journal of Web Semantics 2017; 44: 54–74.
101. Calbimonte JP. Morph-stream. https://github.com/jpcik/morph-streams, 2016.
102. Anicic D and Fodor P. Etalis. https://code.google.com/archive/p/etalis/, 2009.
103. Ziafatia P, Dastanib M, Meyerb JJ et al. Retalis language for information engineering in autonomous robot software.

Journal of Logics 2015; : 65.
104. Ren H, Anicic D and Runkler TA. The synergy of complex event processing and tiny machine learning in industrial IoT. In Proceedings of the 15th ACM International Conference on Distributed and Event-based Systems. pp. 126–135.
105. Diwold K, Mayer S, Einfalt A et al. Grid watch dog: a stream reasoning approach for lightweight scada functionality in low-voltage grids. In Proceedings of the 8th International Conference on the Internet of Things. pp. 1–8.
106. Beck H, Eiter T and Folie C. Ticker: A system for incremental asp-based stream reasoning. Theory and Practice of Logic Programming 2017; 17(5-6): 744–763.
107. Bazoobandi HR, Beck H and Urbani J. Expressive stream reasoning with laser. In International Semantic Web Conference. Springer, pp. 87–103.
108. Bazoobandi HR, Bal H, van Harmelen F et al. Handling impossible derivations during stream reasoning. In European Semantic Web Conference. Springer, pp. 3–19.
109. Artikis A, Pitsikalis M and Mantenoglou P. RTEC: Run-time event calculus. GitHub repository, 2016.
110. Lécué F, Tucker R, Bicer V et al. Predicting severity of road traffic congestion using semantic web technologies. In The Semantic Web: Trends and Challenges: 11th International Conference. pp. 611–627.
111. Lécué F and Pan J. Consistent knowledge discovery from evolving ontologies. Proceedings of the AAAI Conference on Artificial Intelligence 2015; 29(1).
112. Chen J, Lecue F, Pan JZ et al. Knowledge graph embeddings for dealing with concept drift in machine learning. Journal of Web Semantics 2021; 67: 100625.
113. Kharlamov E, Mehdi G, Savković O et al. Semantically-enhanced rule-based diagnostics for industrial internet of things: The SDRL language and case study for Siemens trains and turbines. Journal of Web Semantics 2019; 56: 11–29.
114. Doherty P, Kvarnström J, Wzorek M et al. HDRC3-a distributed hybrid deliberative/reactive architecture for unmanned aircraft systems. Handbook of unmanned aerial vehicles 2014; : 849–952.
115. Klusch M, Meshram A, Schuetze A et al. icm-hydraulic: Semantics-empowered condition monitoring of hydraulic machines. In Proceedings of the 11th international conference on semantic systems. pp. 81–88.
116. Jajaga E, Ahmedi L and Ahmedi F. Streamjess: a stream reasoning framework for water quality monitoring. International Journal of Metadata, Semantics and Ontologies 2016; 11(4): 207–220.
117. Ferrer BR, Iarovyi S, Lobov A et al. Towards processing and reasoning streams of events in knowledge-driven manufacturing execution systems. In 2015 IEEE 13th International Conference on Industrial Informatics (INDIN). IEEE, pp. 1075–1080.
118. Balaj R and Groza A. Detecting influenza epidemics based on real-time semantic analysis of twitter data. In Proc. 3rd Intern. Conf. on Modelling and Development of Intelligent Systems (MDIS 2013). pp. 30–39.
119. Lee T, Kim SH, Balduini M et al. Location-based mobile recommendations by hybrid reasoning on social media streams. In Joint International Semantic Technology Conference. Springer, pp. 261–273.
120. De Brouwer M, Bonte P, Arndt D et al. Distributed continuous home care provisioning through personalized monitoring & treatment planning. In Companion Proceedings of the Web Conference 2020. pp. 143–147.
121. Bourgais M, Giustozzi F and Vercouter L. Detecting situations with stream reasoning on health data obtained with IoT. Procedia Computer Science 2021; 192: 507–516.
122. De Brouwer M, Ongenae F, Daneels G et al. Personalized real-time monitoring of amateur cyclists on low-end devices: proof-of-concept & performance evaluation. In Companion Proceedings of the The Web Conference 2018. pp. 1833–1840.
123. De Brouwer M, Ongenae F and De Turck F. Demonstration of a stream reasoning platform on low-end devices to enable personalized real-time cycling feedback. In European Semantic Web Conference. Springer, pp. 28–32.
124. Municio E, Daneels G, De Brouwer M et al. Continuous athlete monitoring in challenging cycling environments using IoT technologies. IEEE Internet of Things Journal 2019; 6(6): 10875–10887.
125. BakhshandehAbkenar A and Loke SW. Myactivity: Cloud-hosted continuous activity recognition using ontology-based stream reasoning. In 2014 2nd IEEE international conference on mobile cloud computing, services, and engineering. IEEE, pp. 117–126.
126. Hassan T, Ramparany F and Lefebvre G. A semantic event approach to enrich device-free indoor localization data. In Proceedings of the 4th EAI International Conference on Smart Objects and Technologies for Social Good. pp. 106–111.
127. Dejonghe A, Ongenae F, Verstichel S et al. C-geosparql: streaming geosparql support on c-sparql. In Joint Proceedings of the 2nd RDF Stream Processing (RSP 2017) and the Querying the Web of Data (QuWeDa 2017) Workshops co-located with 14th ESWC 2017 (ESWC 2017). pp. 1–10.
128. Morita T, Sugawara Y, Nishimura R et al. Implementing customer reception service in robot cafe using stream reasoning and ros based on printeps. In ISWC (Posters & Demos).
129. Soylu A, Giese M, Schlatte R et al. Querying industrial stream-temporal data: An ontology-based visual approach. Journal of Ambient Intelligence and Smart Environments 2017; 9(1): 77–95.
130. Calbimonte JP. Ontology-based access to sensor data streams. PhD Thesis, Informatica, 2013.
131. Xu Y, Stojanovic N, Stojanovic L et al. An approach for more efficient energy consumption based on real-time situational awareness. In Proceedings of the 8th extended semantic web conference. pp. 270–284.
132. Ziafati P, Dastani M, Meyer JJ et al. Event-processing in autonomous robot programming. In Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems. pp. 95–102.
133. Ali MI, Ono N, Kaysar M et al. Real-time data analytics and event detection for iot-enabled communication systems. Journal of Web Semantics 2017; 42: 19–37.
134. Vanhaeverbeke J, Deprost E, Bonte P et al. Real-time estimation and monitoring of COVID-19 aerosol transmission risk in office buildings. Sensors 2023; 23(5): 2459.
135. Charbonnier L, Giustozzi F, Saunier J et al. Towards a semantic approach to detection of quality issues in manufacturing 4.0. Procedia Computer Science 2024; 246: 2439–2448.

136. Charbonnier L, Giustozzi F, Saunier J et al. Explainability of quality issues in manufacturing: a semantic based approach. In 23nd International Semantic Web Conference (ISWC 2024), volume 3828.
137. De Brouwer M, Steenwinckel B, Fang Z et al. Context-aware & privacy-preserving homecare monitoring through adaptive query derivation for iot data streams with divide. Semantic Web Journal 2023; .
138. Beck H, Bierbaumer B, Dao-Tran M et al. Rule-based stream reasoning for intelligent administration of content-centric networks. In European Conference on Logics in Artificial Intelligence. Springer, pp. 522–528.
139. Santipantakis GM, Vlachou A, Doulkeridis C et al. A stream reasoning system for maritime monitoring. In 25th International Symposium on Temporal Representation and Reasoning (TIME 2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, pp. 20–1.
140. Pitsikalis M and Artikis A. Maritime event recognition. In Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems. pp. 232–235.
141. Troupiotis-Kapeliaris A, Chatzikokolakis K, Zissis D et al. Experimental comparison of complex event processing systems in the maritime domain. In 2020 21st IEEE International Conference on Mobile Data Management (MDM). IEEE, pp. 293–298.
142. Barbieri DF, Braga D, Ceri S et al. An execution environment for c-sparql queries. In Proceedings of the 13th International Conference on Extending Database Technology. pp. 441–452.
143. Barbieri DF, Braga D, Ceri S et al. C-SPARQL: a continuous query language for RDF data streams. International Journal of Semantic Computing 2010; 4(01): 3–25.
144. Ziafati P, Elrakaiby Y, van Zee M et al. Reasoning on robot knowledge from discrete and asynchronous observations. In 2014 AAAI Spring Symposium Series.
145. Soylu A, Kharlamov E, Zheleznyakov D et al. Optiquevqs: A visual query system over ontologies for industry. Semantic Web 2018; 9(5): 627–660.
146. Kharlamov E, Kotidis Y, Mailis T et al. Towards analytics aware ontology based access to static and streaming data. In The Semantic Web–ISWC 2016: 15th International Semantic Web Conference. Springer, pp. 344–362.
147. De Brouwer M, Bonte P, Arndt D et al. Optimized continuous homecare provisioning through distributed data-driven semantic services and cross-organizational workflows. Journal of Biomedical Semantics 2024; 15(1): 9.
148. Calvaresi D and Calbimonte JP. Real-time compliant stream processing agents for physical rehabilitation. Sensors 2020; 20(3): 746.
149. Beck H, Dao-Tran M, Eiter T et al. LARS: A logic-based framework for analyzing reasoning over streams. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence. pp. 1431–1438.

A Appendix A. Key Papers

1. Della Valle E, Ceri S, Van Harmelen F et al. It's a streaming world! reasoning upon rapidly changing information. IEEE Intelligent Systems 2009; 24(6): 83–89.
2. Barbieri D, Braga D, Ceri S et al. Stream reasoning : Where we got so far. In Proceedings of the 4th Workshop on New Forms of Reasoning for the Semantic Web: Scalable & Dynamic.
3. Falzone E, Tommasini R and Della Valle E. Stream reasoning: From theory to practice. In Reasoning Web International Summer School. Springer, 2020. pp. 85–108.
4. Barbieri DF, Braga D, Ceri S et al. Querying rdf streams with c-sparql. ACM SIGMOD Record 2010; 39(1): 20–26. DOI: 10.1145/1860702.1860705.
5. Anicic D, Fodor P, Rudolph S et al. EP-SPARQL: a unified language for event processing and stream reasoning. Proceedings of the 20th international conference on World wide web 2011; DOI:10.1145/1963405.1963495.
6. Anicic D, Rudolph S, Fodor P et al. Stream reasoning and complex event processing in ETALIS. Semantic Web 2012; .
7. Balduini M, Della Valle E, Dell'Aglio D et al. Social listening of city scale events using the streaming linked data framework. In international semantic web conference. Springer, pp. 1–16.
8. Margara A, Urbani J, Van Harmelen F et al. Streaming the web: Reasoning over dynamic data. Journal of Web Semantics 2014; 25: 24–44.
9. Ye J, Dasiopoulou S, Stevenson G et al. Semantic web technologies in pervasive computing: A survey and research roadmap. Pervasive and Mobile Computing 2015; 23: 1–25.
10. Roffia L, Morandi F, Kiljander J et al. A semantic publish-subscribe architecture for the internet of things. IEEE Internet of Things Journal 2016; 3(6): 1274–1296.
11. Dell'Aglio D, Della Valle E, van Harmelen F et al. Stream reasoning: a survey and outlook: A summary of ten years of research and a vision for the next decade. Data Science 2017; 1(1-2): 59–83.
12. Beck H, Dao-Tran M and Eiter T. LARS: A logic-based framework for analytic reasoning over streams. Artificial Intelligence 2018; 261: 16–70.
13. Eiter T, Ogris P and Schekotihin K. A distributed approach to LARS stream reasoning (system paper). Theory and Practice of Logic Programming 2019; 19(5-6): 974–989.
14. Noor MHM, Salcic Z and Wang KIK. Ontology-based sensor fusion activity recognition. Journal of Ambient Intelligence and Humanized Computing 2020; 11(8): 3073–3087.
15. Bonte P, Tommasini R, Valle ED et al. RSP4J: Towards an unifying API for RDF Stream Processing. openreview.net, 2021.