On the interplay between validation and inference in SHACL - an investigation on the Time Ontology

Livio Robaldo [a,*] and Sotiris Batsakis [b]
[a] *HRC School of Law, Swansea University, Wales, UK*
E-mail: livio.robaldo@swansea.ac.uk
[b] *Hellenic Mediterranean University, Greece*
E-mail: sbatsakis@hmu.gr

Abstract.
This paper presents a novel SHACL-based framework for validating the Time Ontology (https://www.w3.org/TR/owl-time). The Time Ontology, currently a W3C Candidate Recommendation, is widely recognized as the "de facto" standard for representing temporal data in the Semantic Web. However, its current OWL axiomatization cannot enforce several validation constraints on temporal knowledge that can be expressed using the Time Ontology vocabulary. These constraints are instead captured by the SHACL formalization proposed in this paper. Nevertheless, we show that SHACL shapes alone are insufficient to validate even relatively simple knowledge graphs that can be encoded using this vocabulary. This limitation arises because validation must be performed on the *inferred* knowledge graph, which SHACL shapes alone cannot derive internally. To address this, our framework first computes the inferred knowledge graph using SHACL-SPARQL rules and then validates it through SHACL shapes.

In the second part of the paper, we argue that our findings extend beyond the Time Ontology and have broader implications for SHACL and knowledge graph reasoning. We therefore view our work as a call to action for the Semantic Web community to systematically investigate the interplay between validation and inference. Specifically, there is a need to study the representational requirements of different use cases to identify the minimal set of SHACL shapes and inference rules necessary for data validation in each context. These research efforts could ultimately lead to the definition of distinct *SHACL dialects*, analogous to how OWL Lite, OWL DL, and other profiles were defined for OWL. The SHACL shapes and SHACL-SPARQL rules that define the proposed framework are freely available in the GitHub repository https://github.com/liviorobaldo/TimeOntologyInSHACL, together with Java programs and detailed instructions for execution.

Keywords: Time Ontology, Encoding temporal knowledge, Validation and inference in SHACL

1. Introduction

Time is a fundamental aspect of reality and the representation of dynamic phenomena. These appear in many application domains, e.g., planning, robotics, compliance checking, real-time systems, computer aided engineering, and any other application that requires to model actions, changes, or behaviors.

There are various aspects of time that need to be taken into account when providing definitions of temporal elements [1]: time can be bounded or not, discrete or continuous, fuzzy or non-fuzzy, periodic, absolute or relative, linear or branching. In addition, temporal representations can be focused on points or intervals, which in turn can be crisp or fuzzy, bounded or unbounded, convex or non-convex, open or closed.

*Corresponding author. E-mail: livio.robaldo@swansea.ac.uk.

These aspects have been extensively studied over the past decades in the literature on Temporal Logics [2][3][4], and widely applied to tasks such as modeling the behavior and properties of state transition systems (e.g., model checking) [5]. This has led to the development of model checkers such as NuSMV [6][7], among others.

The Semantic Web also requires an explicit representation of time [8]. Its standards are grounded in the Resource Description Framework (RDF) [9], which provides a basis for representing interconnected data on the Web. However, RDF is only a *data model*, not a full logical language, as it lacks *inference rules* for deriving new triples from the asserted ones. To address this, several languages have been proposed, beginning with RDF Schema (RDFS), which introduces basic inference rules, for example to support *is-a* reasoning. The most widely used language for inference over RDF knowledge graphs is the Web Ontology Language (OWL) [10], rooted in Description Logics [11]. Among its variants, OWL 2 [12] has become the most widely adopted.

Some proposals for extending Description Logics and OWL with the expressiveness of Temporal Logics have been investigated, although they have not been incorporated into the family of Semantic Web standards. For instance, [13] and [14] provide surveys of Temporal Description Logics as a formal approach for representing time in Description Logics. Along these lines, [15] proposed an extension of OWL2-QL for ontology-based data access. Nevertheless, these and other similar initiatives have only been developed at the theoretical level, with the primary aim of addressing computational complexity and decidability issues, and they lack implementations.

More recently, the Semantic Web community has also emphasized the importance of distinguishing between *validation* and *inference* in knowledge graphs. Inference languages such as OWL and its extensions primarily focus on deriving new facts and checking logical consistency, whereas validation serves a broader purpose. Validation goes beyond detecting contradictions: it also enforces data quality, integrity, and domain-specific constraints that are not captured by purely logical reasoning. Formally, validation is the process of ensuring that the data in a knowledge graph complies with expected rules, formats, or external requirements. Its goal is to verify that each data element satisfies predefined criteria, such as schema constraints, datatypes, or permitted values, thereby ensuring that the data is accurate and well-formed according to the given specifications.

Logical consistency, by contrast, addresses a narrower concern: verifying that all facts in the knowledge graph do not contradict one another under the semantics of the chosen logical formalism. In this sense, logical consistency can be regarded as a specific type of validation, under the (reasonable) assumption that anything inconsistent is also invalid. Nevertheless, although this assumption is indeed reasonable in many use cases, it should also be noted that the explicit representation of inconsistencies, fallacies, and other abnormalities, fit to reason about them, has been identified as a critical gap in current logical frameworks for Artificial Intelligence [16]. A recent RDF-based proposal in this direction is [17], which introduces a novel deontic logic encoded in RDF and SPARQL, where inconsistencies, conflicts, violations, and other abnormalities are explicitly represented and may therefore exist within the knowledge graph. In other words, in [17], inconsistent knowledge graphs are *not* invalid.

To address the recognized need for validation in knowledge graphs, which goes beyond merely checking for inconsistencies, the Shapes Constraint Language (SHACL) was released in 2017 as a W3C Recommendation [18].

SHACL consists of two main components: SHACL Core[1] and SHACL-SPARQL[2]. SHACL Core defines a standard set of built-in constraints for validating RDF data, including predefined constraint types such as cardinality, value ranges, and datatype restrictions. SHACL-SPARQL extends SHACL Core by allowing users to define custom constraints through SPARQL queries. By leveraging SPARQL, it offers greater flexibility and expressiveness, enabling the definition of more complex validation rules.

Since its release, SHACL adoption has steadily increased in both academia and industry (see [19] for an overview). The literature includes several foundational works that propose formal semantics for SHACL (e.g., [20]), also addressing recursive SHACL shapes; these are shapes that may reference themselves, possibly through cycles involving multiple shapes, which can in principle lead to infinite loops during validation [21], [22]. These theoretical works primarily focus on SHACL Core; however, as we show below, SHACL Core alone is not sufficient to represent several validation constraints that require the additional expressivity of SHACL-SPARQL.

[1] https://www.w3.org/TR/shacl/#core-components
[2] https://www.w3.org/TR/shacl/#sparql-constraints

More recent studies have explored practical applications of SHACL in applicative scenarios with potential industrial relevance [23], [24], [25], [26]. This paper further contributes to this growing body of work, particularly to the strand concerned with the applicative uses of SHACL.

This paper focuses on the validation of the Time Ontology[3] as a case study for SHACL. Currently a W3C Candidate Recommendation Draft, the Time Ontology is widely regarded as a "de facto" standard to represent temporal knowledge in the Semantic Web. Several ontologies in different domains import the Time Ontology to model time[4].

The vocabulary of the Time Ontology provides RDF resources for representing both the quantitative and qualitative aspects of temporal instants and intervals across a variety of reference systems, including the standard Gregorian calendar, non-Gregorian calendars, Unix time, and geologic time [27]. Notably, the Time Ontology includes properties corresponding to the well-known thirteen basic Allen temporal relations [28]. However, the full version of Allen's interval algebra is not currently covered by the ontology. The complete algebra also supports *vectors* of these thirteen relations [29], which enable the expression of more complex temporal constraints. Nonetheless, the current Time Ontology vocabulary does not provide RDF resources for representing such vectors.

The Time Ontology is currently implemented in OWL 2 DL. However, its existing OWL axioms support only a limited set of consistency checks, most of which are not directly relevant for temporal reasoning. As a result, it is currently possible to encode clearly nonsensical temporal data, such as intervals that end before they start. The OWL axioms in the Time Ontology cannot detect these cases as logically inconsistent or, more generally, as *invalid*, which considerably limits the ontology's practical utility in real-world applications.

This paper presents a set of SHACL shapes to validate the RDF resources in the Time Ontology. However, we do not consider all RDF resources within the Time Ontology, but instead focus on a specific *fragment*, discussed below in Section 3: the fragment of the ontology related to the `xsd:dateTime` datatype. This is the single datatype for which SPARQL v1.1 defines operators for comparison[5].

This paper also demonstrates that SHACL shapes alone are insufficient to validate certain RDF knowledge graphs, even when only a few of the Time Ontology's RDF resources are employed. While we illustrate this point using the fragment of the Time Ontology discussed below in Section 3, it seems apparent that the implications are broader: if SHACL shapes cannot adequately validate relatively simple knowledge graphs built with the Time Ontology vocabulary, similar limitations can reasonably be expected in more complex cases.

To detect the identified invalid knowledge graphs, it is necessary to *infer* specific additional triples from the explicitly asserted ones. These triples cannot be derived internally by SHACL shapes. Once the inferred triples are available, SHACL shapes can then be used to identify invalid patterns within the *inferred* knowledge graph. In this sense, the inferred triples serve as "intermediate results" required for the final validation, which cannot be accomplished by SHACL shapes in a single step.

In principle, the inferred knowledge graph could be computed using OWL or any other formalism designed for inference over knowledge graphs, such as SWRL [30]. However, in this paper, we chose to use *SHACL rules*, as defined in the W3C Working Group Note of 8 June 2017[6].

Specifically, we used SHACL-SPARQL rules[7], which are also based on SPARQL and thus appear to be the natural choice for this task because they integrate seamlessly with SHACL-SPARQL shapes: both rely on SPARQL, with inference rules expressed as `CONSTRUCT-WHERE` queries to generate inferred triples, and shapes expressed as `SELECT-WHERE` queries to identify invalid patterns.

Using alternative logical formalisms, such as OWL or SWRL, would, in our view, hinder comprehension and make editing and debugging more cumbersome, without offering any significant benefit. In addition, as will be discussed below, it remains unclear whether OWL or other formalisms have sufficient expressivity to represent the same inference rules that we can implemented using SHACL-SPARQL.

---

[3] https://www.w3.org/TR/owl-time

[4] https://www.w3.org/2015/spatial/wiki/OWL_Time_Ontology_adoption

[5] See https://www.w3.org/TR/xmlschema-2/#dateTime. However, it is worth noting that most SPARQL implementations "unofficially" extend the coverage of the official SPARQL v1.1 operators and functions to other datatypes, such as `xsd:date` or `xsd:duration`.

[6] https://www.w3.org/TR/shacl-af , retrieved September 10, 2025

[7] https://www.w3.org/TR/shacl-af/#SPARQLRule

Similar to standard OWL reasoners such as HermiT [31], which repeatedly apply OWL axioms until no further triples can be inferred, our approach also iteratively applies SHACL-SPARQL rules until the inferred graph reaches closure, prior to validation. This iterative process is not prescribed in the aforementioned Working Group Note, nor is it implemented in existing SHACL libraries such as the TopBraid SHACL Java library v.1.3.2[8], which we used in our implementation. These libraries execute SHACL rules only once. Consequently, the software available on our GitHub repository programmatically re-executes the SHACL-SPARQL rules until no new triples are produced, and then validates the resulting knowledge graph against the SHACL shapes. We believe this two-step approach should be adopted by any software for validating knowledge graphs using SHACL. In other words, what our implementation currently achieves programmatically should, in our view, be formalized and standardized by the W3C.

The rest of the paper is organized as follows. The next section provides a brief review of the literature on representing time in the Semantic Web. Section 3 is then dedicated to the Time Ontology, focusing specifically on the fragment relevant to our proposed formalization, namely the RDF resources associated with the `xsd:dateTime` datatype. Section 4 presents the semantic characterization of the Time Ontology, as originally proposed by its proponents and initial editors, Jerry R. Hobbs and Feng Pan, and axiomatized in first-order logic in [32]. While we base our discussion on their work, we do not follow their axiomatization strictly, as we disagree with some of its technical choices. This section also explains the rationale behind our decisions and introduces the variant of [32]'s axiomatization that we adopt in this work.

Sections 5 and 7 form the core of this paper. The former demonstrates that SHACL shapes alone are insufficient to detect even some simple knowledge graphs that can be constructed with the Time Ontology vocabulary, while the latter presents the SHACL axiomatization corresponding to the first-order logic axiomatization from Section 4. Section 7 discusses possible extensions of both the proposed SHACL axiomatization and the Time Ontology vocabulary, while Section 8 provides broader reflections on the challenges and risks of using SHACL to validate RDF knowledge graphs, based on the lessons learned from our work on the Time Ontology. Finally, Section 9 summarizes the main findings and advocates the definition of distinct SHACL dialects, as mentioned in the abstract.

2. Background: representing temporal data in the Semantic Web

Time is not inherently integrated into Semantic Web standards, and maintaining compatibility with these standards requires representations that incorporate reasoning rules into existing ontologies, rather than relying on specialized reasoning software. There are two main components for representing time, as needed for application use:

a. The representation of temporal concepts such as time points and intervals.

b. The representation of dynamic properties (fluent properties) of objects and events using the above mentioned temporal concepts.

Core temporal concepts, their properties, and constraints are defined using temporal ontologies, while the application of these properties in specific domains is an orthogonal dimension. Temporal ontologies include definitions of temporal intervals and points, among others, and these definitions can be used to represent temporal properties in various ways, such as through 4D-fluents or reification.

Although the focus of this paper is on (a), specifically the representation of temporal concepts based on the definitions in the Time Ontology, the next two subsections will briefly survey past literature on representing both aspects of temporal representation in the Semantic Web, i.e., (a) and (b) above, respectively.

2.1. Temporal Ontologies

Time is a fundamental aspect of the world and temporal concepts are involved on almost all knowledge representation tasks since many properties of objects are dynamic. Thus, many temporal ontologies have been proposed in

[8] https://repo1.maven.org/maven2/org/topbraid/shacl/1.3.2

the literature [1]. Among these, the Time Ontology in OWL, also know as OWL-Time[9], is the most widely used. A recent survey on the representation and management of temporal data is provided in [33].

The Time Ontology is a temporal ontology and currently a W3C Candidate Recommendation Draft that provides definitions of temporal points, intervals, and their relationships. Having the status of a W3C candidate recommendation draft, it is widely used, but since the adoption of the Time Ontology as a standard is an ongoing work, several alternative temporal ontologies have also been proposed. Since the essence of the Semantic Web is the definition of common vocabularies, this work focuses on enhancing the Time Ontology thus contributing to the standardization effort rather than proposing yet another temporal ontology.

Other temporal ontologies include Resusable Time Ontology [34], TimeML [35], which offers a translation to DAML-OIL, the predecessor of OWL, GFO-Time [36], which is part of the upper ontology GFO, TOWL [37], which extends OWL with temporal constructs but is not compliant with standard Semantic Web tools, and TL-OWL [38], which also extends Semantic Web standards with temporal concepts, similarly to OWL-Met [39]. In PSI-ULO [40] temporal concepts are part of an upper ontology defined in PSI-Time [41] while in TimeLine ontology [42] definitions for temporal concepts for digital music are provided. Temporal RDF [43] proposes extending RDF with temporal annotations while SWRL-Time [44], CNTRO [45] and SOWL [46] define reasoning mechanisms based on Semantic Web Rule Language (SWRL) [30].

This paper proposes using SHACL-SPARQL rules as an alternative to SWRL, OWL, or other logical formalisms for the Semantic Web to enhance the current version of the Time Ontology. As noted in the introduction, SHACL-SPARQL rules were chosen because they integrate seamlessly with SHACL-SPARQL shapes. The potential use of SWRL, OWL, or other formalisms as alternatives warrants further investigation.

While SHACL has been an official W3C recommendation since 2017[10], neither SWRL nor SHACL-SPARQL rules have (yet?) achieved standard status. In other words, currently both are only *proposals* for W3C recommendations. SWRL has been a proposal since 2004[11], while SHACL-SPARQL rules have been a proposal since 2017[12].

2.2. Representation of dynamic/fluent properties

Almost all conceivable application domains involve objects with dynamic properties, also known as *fluent properties*, that change over time. The definitions of the temporal concepts involved (e.g., the temporal instant when an event occurs or the temporal interval during which a dynamic property holds) are provided by temporal ontologies. However, their application in specific domains is not straightforward, as fluent properties are not binary (e.g., they involve a subject, an object, and a temporal instant or interval). As a result, they cannot be directly represented as object or datatype properties of a class, leading to many different approaches in practice for using temporal concepts.

Integrating temporal concepts defined in temporal ontologies with representations of fluent properties in the Semantic Web can be achieved in various ways such as extending repositories with support for ternary relations [47] (standard RDF is based on triple stores), versioning [48], which is based on creating a new copy of the knowledge base when a property is modified, named graphs [49], the generic method of reification, and the 4D fluents approach proposed in [50]. Various methods are presented and compared in [51] and they have been extended to cover spacial properties of objects in [52]. The work in [51] retains compatibility with OWL/RDFS and offers integrated reasoning capabilities which is not the case of other approaches such as versioning and temporal annotated named graphs. Temporal reasoning in [51], as well as in SWRL-Time [44] and CNTRO [45], is achieved through SWRL.

In [53], the CHRONOS ED tool was proposed, offering enhanced performance compared to the work in [51]. However, this approach was somewhat ad-hoc, as it relied on specialized reasoning software that was compatible with specific ontologies, rather than using generic semantic OWL reasoners such as HermiT [54] or Pellet [55], which limited its overall applicability.

It is important to note that temporal ontologies, which provide definitions for temporal instances and intervals, can be used in the approaches discussed above by importing them into the corresponding formal frameworks. Several

[9] https://www.w3.org/TR/owl-time
[10] https://www.w3.org/TR/shacl
[11] https://www.w3.org/submissions/SWRL
[12] https://www.w3.org/TR/shacl-af

alternatives exist for defining temporal concepts, but the Time Ontology has emerged as the "de facto" standard, even though it is not yet an official W3C recommendation. As the most significant proposal in this area, it is the focus of the present work and will be described in more detail in the next section.

3. The Time Ontology

The Time Ontology is currently a W3C candidate recommendation draft[13] publicly available online and downloadable in Turtle format. Its IRI is "`http://www.w3.org/2006/time#`"; in this paper, as well as in the associated GitHub repository, we will refer to this IRI with the prefix "`time:`". The version of the Time Ontology used in this paper is the one retrieved on September 10, 2025; of course, subsequent versions of the Time Ontology could not be compatible with the implementation proposed in this paper.

While the full formal definitions of the ontology's resources (classes, individuals, and properties) are available at the Time Ontology's homepage, this section will focus on the resources that may be processed via SHACL, namely the ones associated with the `xsd:dateTime` datatype. As pointed out in the Introduction, `xsd:dateTime` is the single temporal datatype for which SPARQL 1.1 defines comparison operators[14], therefore it is the single one for which it is currently possible to assert SHACL shapes and rules. These resources are shown in Figure 1.

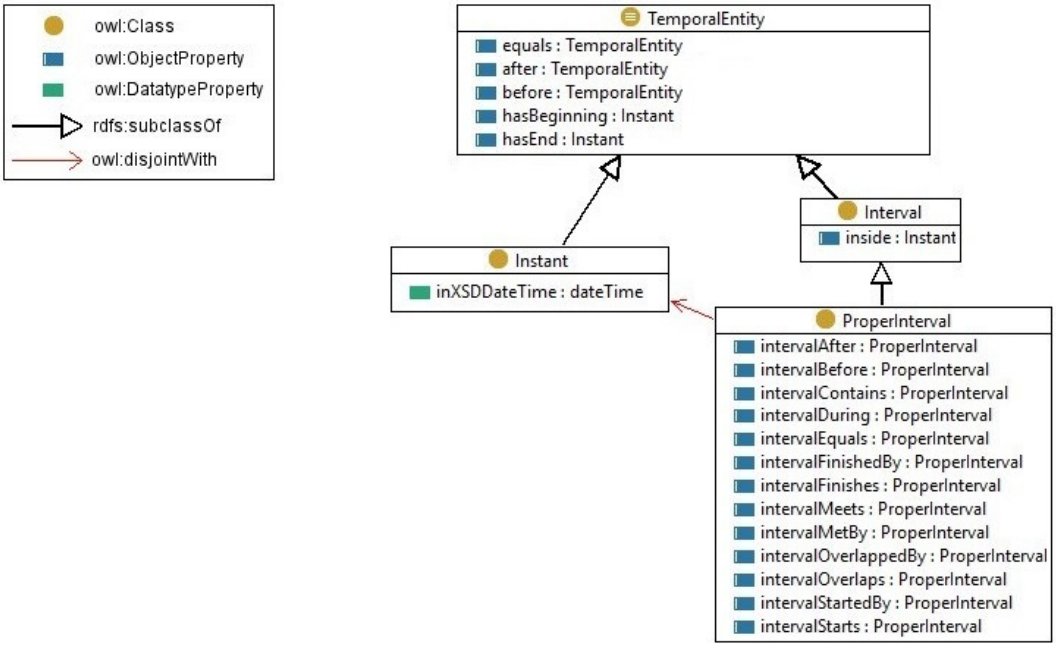

Fig. 1. Classes and properties of the Time Ontology considered in this paper. This figure is a modified version of the one available at https://www.w3.org/TR/owl-time/#topology (retrieved on September 10, 2025).

The top-level class `TemporalEntity` has two subclasses: `Interval` and `Instant`. Instances of `Interval` represent temporal entities with duration, bounded by a start and an end specified through `hasBeginning` and `hasEnd`. Instances of `Instant` denote temporal entities with *zero* duration, conceived as limiting cases of intervals where the start and end coincide. `ProperInterval`, a subclass of `Interval`, is defined as an interval whose beginning and end are distinct. `ProperInterval` and `Instant` are declared disjoint in the Time Ontology.

Regarding properties, this paper considers a single datatype property, `inXSDDateTime`, which links instances of `Instant` to values of the datatype `xsd:dateTime`. `hasBeginning` and `hasEnd` specify the start and end

[13] https://www.w3.org/TR/owl-time

[14] See https://www.w3.org/TR/2013/REC-sparql11-query-20130321/#OperatorMapping. Since the W3C specification allows the timezone in `xsd:dateTime` to be optional, the SHACL axiomatization presented below includes a shape that enforces the presence of a timezone.

points of temporal entities. `inside` links intervals to instants that occur *properly* within them; in other words, the beginning and end instants of an interval are not considered to occur "inside" the interval. `equals`, `after`, and `before` encode temporal orderings between temporal entities: `equals` connects two temporal entities that start and end at the same time; if a temporal entity comes `after` another, then the latter precedes the former; if a temporal entity comes `before` another, then the former precedes the latter.

Finally, the thirteen properties shown in Figure 1 within the `ProperInterval` box correspond to Allen's temporal relations, each linking a pair of `ProperInterval` instances. Allen's temporal algebra [28] is a cornerstone of temporal logic research. However, as noted earlier in the Introduction, the full version of Allen's algebra is defined over vectors of the basic Allen temporal relations, whereas the current Time Ontology vocabulary can represent only individual relations (i.e., single-element vectors). Extending the ontology to cover the full algebra is a possible direction for future work, but it should be noted that the constraint propagation algorithm introduced by Allen [28] for computing the closure of the algebra has exponential complexity [29]. Consequently, such an extension could result in an ontology that is impractical for real-world applications. To mitigate this issue, several tractable sub-algebras with polynomial-time reasoning have been identified (e.g., [56]). More practical extensions of the Time Ontology could therefore restrict their vocabulary to the constructs supported by these sub-algebras.

One of the thirteen basic[15] temporal relations in Allen's temporal algebra is `Equal`, represented in the Time Ontology by the property `intervalEquals`. If two intervals are related by `Equal`, then their beginnings and ends coincide. Six of the remaining relations, namely `Before`, `During`, `Meets`, `Starts`, `Finishes`, and `Overlaps`, are often regarded as the "primary" ones and correspond to the Time Ontology properties `intervalBefore`, `intervalDuring`, `intervalMeets`, `intervalStarts`, `intervalFinishes`, and `intervalOverlaps`, respectively. The other six, i.e., `After`, `Contains`, `MetBy`, `StartedBy`, `FinishedBy`, and `OverlappedBy`, are their inverse properties, and are represented in the Time Ontology by the properties `intervalAfter`, `intervalContains`, `intervalMetBy`, `intervalStartedBy`, `intervalFinishedBy`, and `intervalOverlappedBy`.

The definitions of `Equal` and the six "primary" relations are as follows:

(1)
a. `T1 Equal T2`: the beginning of `T1` coincides with the beginning of `T2` and the end of `T1` coincides with the end of `T2`.
b. `T1 Before T2`: `T1` ends earlier than `T2` begins.
c. `T1 During T2`: `T2` *properly* contains `T1`.
d. `T1 Meets T2`: the end of `T1` coincides with the beginning of `T2`.
e. `T1 Starts T2`: the beginning of `T1` coincides with the beginning of `T2`, while the end of `T1` occurs temporally before the end of `T2`.
f. `T1 Finishes T2`: the end of `T1` coincides with the end of `T2`, while the beginning of `T1` occurs temporally after the beginning of `T2`.
g. `T1 Overlaps T2`: the beginning of `T1` occurs temporally before the beginning of `T2` and the end of `T1` occurs temporally before the end of `T2`.

In addition to the RDF resources shown in Figure 1, the Time Ontology includes several RDFS and OWL axioms that define class hierarchies, property domains and ranges, and inverse relationships. The semantics of these axioms is incorporated into the axiomatization proposed in this paper, as explained in the next section. The ontology also contains various `owl:allValuesFrom`, `owl:hasValue`, and `owl:cardinality` restrictions, which are not relevant to temporal modeling and are therefore excluded from consideration in this work.

It should be observed, on the other hand, that when the Time Ontology was originally proposed, about twenty years ago, its proponents and editors, Jerry R. Hobbs and Feng Pan, postulated in [32] a formal semantics for the ontology in the form of a first-order logic axiomatization establishing a topological ordering among instants and

[15] Allen's original work [28] also defines the property `In`, which subsumes `During`, `Starts`, and `Finishes`. The Time Ontology includes a corresponding property `intervalIn`. The Time Ontology also includes an additional property, `intervalDisjoint`, which has no counterpart in Allen's original work and subsumes `intervalBefore` and `intervalAfter`. This paper does not consider these two additional super-properties and focuses solely on the thirteen basic temporal relations.

intervals. Most of these first-order logic axioms, however, were never implemented, possibly because it proved difficult to identify the specific RDFS or OWL constructs needed to capture the intended topological ordering.

By contrast, as we will show below, implementing our variant of the first-order logic axioms from [32] in SHACL is relatively straightforward. As noted earlier, we do not follow Hobbs and Pan's axiomatization strictly, as we disagree with two of its fundamental technical choices. The next section explains our reasoning and presents the variant of [32]'s axiomatization that we adopt in this work.

4. A semantic characterization of the Time Ontology's resources in Figure 1

In the previous section, the RDF resources in Figure 1 were described in plain text, along with some of the constraints imposed on them by the RDFS and OWL axioms in the official version of the Time Ontology. In this subsection, we provide a rigorous and comprehensive semantic characterization of these resources by adapting the first-order logic axiomatization originally proposed by Jerry R. Hobbs and Feng Pan in [32].

As noted at the end of the previous section, we disagree with two fundamental technical choices made in [32]. First, [32] represents infinite intervals as intervals in which either the beginning or the end (or both) is *explicitly* missing. Second, [32] does not explicitly encode or axiomatize the relation `equals`; instead, it uses the operator "$=$" to denote equality, but its meaning remains unspecified, i.e., the operator was not axiomatized in [32].

Regarding the representation of infinite intervals, [32] states that "a positively infinite interval has no end, and a negatively infinite interval has no beginning". In other words, [32] treats infinite intervals as those that never appear as the subject of `hasBeginning` (so that their beginning is interpreted as $-\infty$) and/or as the subject of `hasEnd` (so that their end is interpreted as $+\infty$).

This choice affects the formulation of several axioms in [32], such as the one defining `ProperInterval` (here, the symbol "$\neq$" from [32] is replaced by "$\neg$`equals`" to reflect our revision of the second technical choice).

$$(2)\quad \forall_{\mathtt{T}}[\,\mathtt{ProperInterval(T)} \leftrightarrow (\mathtt{Interval(T)} \wedge \forall_{\mathtt{tb,te}}[\,(\mathtt{hasBeginning(T,tb)} \wedge \mathtt{hasEnd(T,te)}) \rightarrow \neg\mathtt{equals(tb,te)}\,])\,]$$

Suppose the knowledge graph only includes the triple "`T a time:Interval`", corresponding to `Interval(T)`, in first-order logic. Since `T`'s beginning and end are missing, the axiom in (2) infers that it is an instance of `ProperInterval`. That is because the nested universal quantification holds even if the knowledge graph contains *no* individuals satisfying the universally quantified formula when substituted for the associated variables.

In Hobbs and Pan's framework, this is correct because if the beginning and end of `T` are missing, then `T` is interpreted as the interval $[-\infty, +\infty]$, which is indeed considered a proper interval.

Other axioms from [32] whose formulation is affected by this technical choice are those defining Allen's temporal relations. For example, the axiom defining `intervalStart`:

$$(3)\ \forall_{\mathtt{T1,T2}}[\,\mathtt{intervalStarts(T1,T2)} \leftrightarrow (\mathtt{ProperInterval(T1)} \wedge \mathtt{ProperInterval(T2)} \wedge \exists_{\mathtt{t2}}[\,\mathtt{hasEnd(t2,T1)} \wedge \forall_{\mathtt{t1}}[\,\mathtt{hasBeginning(t1,T1)} \leftrightarrow \mathtt{hasBeginning(t1,T2)}\,] \wedge \forall_{\mathtt{t4}}[\,\mathtt{hasEnd(t4,T2)} \rightarrow \mathtt{before(t2,t4)}\,]\,])\,]$$

This axiom stipulates that if `intervalStart` holds between `T1` and `T2`, then the end of `T1` *must* exist; therefore, it cannot be $-\infty$ or $+\infty$. The other three endpoints can instead take infinite values, but this is consistent with the definition in (3). If both beginnings are $-\infty$, the two intervals indeed start at the same time. If only one beginning is $-\infty$, the first nested universal quantification is false, as it is satisfied only if either both beginnings do not exist or both exist and coincide; this is correct, because in this case the intervals do not start simultaneously. If `T2`'s end does not exist (i.e., it is interpreted as $+\infty$), the second nested universal quantification is true. This is correct because `T1`'s end exists and therefore occurs before $+\infty$, i.e., `T2`'s end. Finally, if `T2`'s end exists, the second nested universal quantification is true only if `T1`'s end occurs before `T2`'s end, which is again correct.

We do not endorse [32]'s assumption that if an interval's endpoints are missing, they should be interpreted as $-\infty$ or $+\infty$. In our view, this assumption conflicts with the Open World Assumption, a cornerstone of RDF semantics. In RDF, if a triple does not exist, its value is simply *unknown*. Accordingly, if an interval's endpoint is missing, its

value is just unknown: it could be $-\infty$, $+\infty$, or a specific finite value. Only once the value becomes known, i.e., identified as $-\infty$, $+\infty$, or a specific finite value, we must verify whether the knowledge graph still contains valid information. Our axiomatization thus adheres to the Open World Assumption, so that all axioms from [32] affected by the assumption of treating missing endpoints as infinite endpoints have been rewritten as shown below.

Indeed, the assumption of treating missing endpoints as infinite is, more broadly, connected to another gap in Hobbs and Pan's formalization, which we examine in further detail in the next section. In [32], endpoints cannot be associated with specific values, whereas in the fragment of the Time Ontology shown in Figure 1, the datatype property `inXSDDateTime` does associate instants with specific `xsd:dateTime` values. Explicitly representing $-\infty$ or $+\infty$ would thus require special `xsd:dateTime` values that "posit instants at positive and negative infinity" (cit. from [32]), an alternative also advocated by Hobbs and Pan. However, this is not possible because every valid `xsd:dateTime` value does correspond to a specific point in time; in other words, the `xsd:dateTime` datatype does not provide "dummy" values that could be used to denote $-\infty$ and $+\infty$. Consequently, the current version of the Time Ontology is actually *unable* to represent infinite intervals, and the only viable solution appears to be *extending* the ontology's vocabulary to include additional RDF resources that explicitly represent $-\infty$ and $+\infty$. Although this is not strictly within the scope of this paper, which focuses only on the RDF resources shown in Figure 1, in Section 7 below we will propose a possible extension of the Time Ontology vocabulary for this purpose.

The following subsection presents our alternative first-order logic axiomatization for the RDF resources shown in Figure 1. In addition to adhering to the Open World Assumption, as explained above, our first-order logic axiomatization also includes axioms explicitly defining the semantics of `equals`, which in [32] is represented using the mathematical operator "=" but whose semantics is left implicit, as pointed out at the beginning of this section.

4.1. A first-order logic axiomatization for the semantics of the RDF resources in Figure 1

This section presents the first-order logic axiomatization adopted in this paper. We begin with the first-order logic axioms that are formalized as RDFS or OWL axioms in the current official version of the Time Ontology. These are listed in Table 1; for readability, the prefix "`time:`" has been omitted from the RDF resource names in the table.

1.	$\forall_{t}$ [Instant(t)→TemporalEntity(t)]	Instant rdfs:subClassOf TemporalEntity
2.	$\forall_{I}$ [Interval(I)→TemporalEntity(I)]	Interval rdfs:subClassOf TemporalEntity
3.	$\forall_{T}$ [TemporalEntity(T)→(Instant(T)∨Interval(T))]	TemporalEntity owl:unionOf (Instant Interval)
4.	$\forall_{T,t}$ [hasBeginning(T, t)→ (TemporalEntity(T)∧Instant(t))]	hasBeginning rdfs:domain TemporalEntity hasBeginning rdfs:range Instant
5.	$\forall_{T,t}$ [hasEnd(T, t)→(TemporalEntity(T)∧ Instant(t))]	hasEnd rdfs:domain TemporalEntity hasEnd rdfs:range Instant
6.	$\forall_{T,t}$ [inside(T, t)→(Interval(T)∧ Instant(t))]	inside rdfs:domain Interval inside rdfs:range Instant
7.	$\forall_{t}$ [ProperInterval(t)→ Interval(t)]	ProperInterval rdfs:subClassOf Interval
8.	$\forall_{T,t}$ [equals(T1, T2)→ (TemporalEntity(T1)∧TemporalEntity(T2))]	equals rdfs:domain TemporalEntity equals rdfs:range TemporalEntity
9.	$\forall_{T,t}$ [before(T1, T2)→ (TemporalEntity(T1)∧TemporalEntity(T2))]	before rdfs:domain TemporalEntity before rdfs:range TemporalEntity
10.	$\forall_{T,t}$ [after(T1, T2)→ (TemporalEntity(T1)∧TemporalEntity(T2))]	after rdfs:domain TemporalEntity after rdfs:range TemporalEntity
11.	$\forall_{T1,T2}$ [after(T1, T2)↔before(T2, T1)]	before owl:inverseOf after

Table 1
First-order logic axioms implemented as RDFS and OWL axioms in the Time Ontology

The axiomatization in [32] includes all axioms listed in Table 1, except for 8, 9, and 10, which specify domain and range of the three predicates `equals`, `before`, and `after`. These three predicates are intended to define a basic temporal algebra that closely mirrors those of the familiar mathematical operators "=", "<", and ">", with the only difference being that the latter apply to integers, whereas `equals`, `before`, and `after` apply to temporal entities.

As noted above, [32] does not use `equals` but instead the corresponding mathematical operator "=", without providing an explicit axiomatization of its semantics. The predicate `after` is likewise not axiomatized in [32], since axiom 11 in Table 1 allows every statement involving `after` to be rewritten as an equivalent statement involving `before`. By contrast, `before` is axiomatized: [32] defines it as anti-reflexive, anti-symmetric, and transitive. The anti-reflexivity of `before` is axiomatized in [32] as follows:

(4) $\forall_{\texttt{T1,T2}}\,[\texttt{before(T1, T2)} \rightarrow \neg\texttt{equals(T1, T2)}]$

However, in our view, this axiom only indirectly encodes the anti-reflexivity of `before`, which should instead be formalized as in axiom "1." in Table 2: a temporal entity cannot be related to *itself* via `before`. The transitivity of `before` is instead formalized in [32] as in axiom "2.", and we add a corresponding axiom to enforce the transitivity of `equals` (axiom "4."). We chose not to import the axiom from [32] enforcing `before` to be anti-symmetric, since this property can now be derived[16] from axioms "1." and "2.". Similarly, we omit an axiom asserting the reflexivity of `equals` (i.e., that every temporal entity is equal to itself), as it would only introduce unnecessary triples into the knowledge graph. Finally, we add axioms "5."–"8." to assert that the properties `before`, `hasBeginning`, `hasEnd`, and `inside` are preserved under substitution of `equals` in either of their arguments. For example, under the parallel between `equals` and `before` with the mathematical operators "=" and "<", axiom "5." states both "$\texttt{((t1<t2)} \wedge \texttt{(t2=t3))} \rightarrow \texttt{(t1<t3)}$" and "$\texttt{((t1=t2)} \wedge \texttt{(t2<t3))} \rightarrow \texttt{(t1<t3)}$". Note that (4) now follows from axioms "1.", "3.", and "5.": if both `before(T1, T2)` and `equals(T1, T2)` hold, axiom "3." yields `equals(T2, T1)`, and axiom "5." then yields `before(T1, T1)`, which contradicts axiom "1.".

1.	$\forall_{\texttt{T}}\,[\neg\texttt{before(T, T)}]$
2.	$\forall_{\texttt{T1,T2,T3}}\,[\texttt{(before(T1, T2)} \wedge \texttt{before(T2, T3))} \rightarrow \texttt{before(T1, T3)}]$
3.	$\forall_{\texttt{T1,T2}}\,[\texttt{equals(T1, T2)} \rightarrow \texttt{equals(T2, T1)}]$
4.	$\forall_{\texttt{T1,T2,T3}}\,[\texttt{(equals(T1, T2)} \wedge \texttt{equals(T2, T3))} \rightarrow \texttt{equals(T1, T3)}]$
5.	$\forall_{\texttt{T1,T2,T3}}\,[\texttt{((equals(T1, T2)} \wedge \texttt{before(T2, T3))} \vee \texttt{(before(T1, T2)} \wedge \texttt{equals(T2, T3)))} \rightarrow \texttt{before(T1, T3)}]$
6.	$\forall_{\texttt{T1,T2,T3}}\,[\texttt{((equals(T1, T2)} \wedge \texttt{hasBeginning(T2, T3))} \vee \texttt{(hasBeginning(T1, T2)} \wedge \texttt{equals(T2, T3)))} \rightarrow \texttt{hasBeginning(T1, T3)}]$
7.	$\forall_{\texttt{T1,T2,T3}}\,[\texttt{((equals(T1, T2)} \wedge \texttt{hasEnd(T2, T3))} \vee \texttt{(hasEnd(T1, T2)} \wedge \texttt{equals(T2, T3)))} \rightarrow \texttt{hasEnd(T1, T3)}]$
8.	$\forall_{\texttt{T1,T2,T3}}\,[\texttt{((equals(T1, T2)} \wedge \texttt{inside(T2, T3))} \vee \texttt{(inside(T1, T2)} \wedge \texttt{equals(T2, T3)))} \rightarrow \texttt{inside(T1, T3)}]$

Table 2

Axiomatization of `before` and `equals`. `before` is anti-reflexive while `equals` is symmetric (axioms "1." and "3."). Both are transitive (axioms "2." and "4."). All properties are preserved under substitution of `equals` in either argument (axioms "5."-"8.").

The remaining first-order logic axioms defining our semantics are shown in Table 3 and Table 4, with the latter specifying the axioms of Allen's temporal relations.

Axiom "1." in Table 3 represents the key difference between [32]'s axiomatization and the one adopted in this paper. As explained above, in our axiomatization we do *not* represent infinite endpoints as missing endpoints; instead, we require that every temporal entity always has endpoints. These endpoints can be associated with a specific `xsd:dateTime` value via the datatype property `inXSDDateTime`, or with $-\infty$ or $+\infty$ via the RDF

[16] If both `before(T1, T2)` and `before(T2, T1)` hold, transitivity would imply `before(T1, T1)` and `before(T2, T2)`, both of which contradict axiom "1."

resources discussed below in Section 7, but they may also exist in the ontology without any associated value, i.e., their value can remain unknown. In other words, axiom "1." in Table 3 does not itself add specific knowledge about the temporal entities; it merely states that every temporal entity necessarily has two endpoints, whose values could even be unknown. This contrasts with [32], where a temporal entity may lack one or both endpoints. Axiom "1." affects the formulation of subsequent axioms, which assume the existence of both endpoints.

Axioms "2."–"5." in Table 3 are imported from [32] as they are: if a temporal entity is an instant, its beginning and end are the instant itself; if a temporal entity has multiple beginnings/ends, these beginnings/ends are required to be equal. Axiom "6." is also imported from [32] as it is: every proper interval begins before it ends.

Axiom "7.", by contrast, is only slightly modified from [32]: in the original, it is defined for the class `Interval`, but since `Interval` is not disjoint from `Instant` (instants are treated in the ontology as intervals of zero duration), we decided to generalize it to the broader class `TemporalEntity`. Axiom "7." therefore states that a temporal entity cannot end before it begins. In addition, we introduce axioms "8." and "9.", which state, respectively, that for every temporal entity with beginning `tb` and end `te`, if another temporal entity `t` occurs before `tb`, then it also occurs before `te`; symmetrically, if `te` occurs before `t`, then `tb` also occurs before `t`. These two axioms are not included in [32]'s axiomatization; however, as we will show later in Subsection 6.1, they are necessary to detect certain invalid knowledge graphs that are, therefore, *not* recognized as inconsistent in [32]'s axiomatization.

Axiom "10." is our variant of the following axiom from [32]:

(5) $\forall_{T1,T2}[\,\texttt{before(T1, T2)} \rightarrow \exists_{tb,te}[\,(\texttt{hasBeginning(T2, tb)} \wedge \texttt{hasEnd(T1, te)}) \rightarrow \texttt{before(te, tb)}\,]\,]$

This axiom states that if two temporal entities are related by the property `before`, then the beginning of the first entity and the end of the second entity must exist (hence, according to the representational choice in [32], they are not infinite), and the former occurs before the latter. In our framework, however, the beginning and end of every temporal entity *always* exist, as guaranteed by axiom "1." in Table 3. Therefore, the existential quantifiers in (5) can be replaced with corresponding universal quantifiers in the antecedent, yielding axiom "10." in Table 3. Finally, the axiom in "11.", for the property `inside`, is imported from [32] as it is.

1.	$\forall_{T}[\texttt{TemporalEntity(T)} \rightarrow \exists_{tb,te}[\,\texttt{hasBeginning(T, tb)} \wedge \texttt{hasEnd(T, te)}]$
2.	$\forall_{t}[\texttt{Instant(t)} \leftrightarrow \texttt{hasBeginning(t, t)}]$
3.	$\forall_{t}[\texttt{Instant(t)} \leftrightarrow \texttt{hasEnd(t, t)}]$
4.	$\forall_{T,t1,t2}[(\texttt{TemporalEntity(T)} \wedge \texttt{hasBeginning(T, t1)} \wedge \texttt{hasBeginning(T, t2)}) \rightarrow \texttt{equals(t1, t2)}]$
5.	$\forall_{T,t1,t2}[(\texttt{TemporalEntity(T)} \wedge \texttt{hasEnd(T, t1)} \wedge \texttt{hasEnd(T, t2)}) \rightarrow \texttt{equals(t1, t2)}]$
6.	$\forall_{T,tb,te}[\,(\texttt{ProperInterval(T)} \wedge \texttt{hasBeginning(T, tb)} \wedge \texttt{hasEnd(T, te)}) \rightarrow \texttt{before(tb, te)}\,]$
7.	$\forall_{T,tb,te}[\,(\texttt{TemporalEntity(T)} \wedge \texttt{hasBeginning(T, tb)} \wedge \texttt{hasEnd(T, te)}) \rightarrow \neg\texttt{before(te, tb)}\,]$
8.	$\forall_{T,tb,te,t}[\,(\texttt{TemporalEntity(T)} \wedge \texttt{hasBeginning(T, tb)} \wedge \texttt{hasEnd(T, te)} \wedge \texttt{before(t, tb)}) \rightarrow \texttt{before(t, te)}\,]$
9.	$\forall_{T,tb,te,t}[\,(\texttt{TemporalEntity(T)} \wedge \texttt{hasBeginning(T, tb)} \wedge \texttt{hasEnd(T, te)} \wedge \texttt{before(te, t)}) \rightarrow \texttt{before(tb, t)}\,]$
10.	$\forall_{T1,T2,tb,te}[\,(\texttt{before(T1, T2)} \wedge \texttt{hasBeginning(T2, tb)} \wedge \texttt{hasEnd(T1, te)}) \rightarrow \texttt{before(te, tb)}\,]$
11.	$\forall_{T,t,tb,te}[\,(\texttt{inside(T, t)} \wedge \texttt{hasBeginning(T, tb)} \wedge \texttt{hasEnd(T, te)}) \rightarrow (\texttt{before(tb, t)} \wedge \texttt{before(t, te)})]$

Table 3

First-order logic axioms employed in this paper, adapted from [32].

Table 4 presents the final axioms of our first-order logic axiomatization; these axioms define the semantics of the property `intervalEquals` and the six "primary" Allen temporal relations. All axioms, except the one for

intervalBefore, have been modified[17] compared to their formulation in [32], based on the assumption that every temporal entity always has both endpoints, even if the *values* of these endpoints may be unknown. Additional axioms, similar to axiom "11." in Table 1, which states that before and after are inverse relations, enforce that intervalAfter is the inverse of intervalBefore, intervalMetBy is the inverse of intervalMeets, and so on; these axioms are omitted from Table 4 for brevity but are available in the GitHub repository.

intervalEquals	$\forall_{T1,T2}$ [intervalEquals(T1, T2) → (ProperInterval(T1) ∧ ProperInterval(T2) ∧ equals(T1, T2))]
intervalBefore	$\forall_{T1,T2}$ [intervalBefore(T1, T2) → (ProperInterval(T1) ∧ ProperInterval(T2) ∧ before(T1, T2))]
intervalMeets	$\forall_{T1,T2,tb,te}$ [(intervalMeets(T1, T2) ∧ hasEnd(T1, te) ∧ hasBeginning(T2, tb)) → (ProperInterval(T1) ∧ ProperInterval(T2) ∧ equals(tb, te))]
intervalOverlaps	$\forall_{T1,T2,tb1,te1,tb2,te2}$ [(intervalOverlaps(T1, T2) ∧ hasBeginning(T1, tb1) ∧ hasEnd(T1, te1) ∧ hasBeginning(T2, tb2) ∧ hasEnd(T2, te2)) → (ProperInterval(T1) ∧ ProperInterval(T2) ∧ before(tb1, tb2) ∧ before(tb2, te1) ∧ before(te1, te2))]
intervalStarts	$\forall_{T1,T2,tb1,te1,tb2,te2}$ [(intervalStarts(T1, T2) ∧ hasBeginning(T1, tb1) ∧ hasEnd(T1, te1) ∧ hasBeginning(T2, tb2) ∧ hasEnd(T2, te2)) → (ProperInterval(T1) ∧ ProperInterval(T2) ∧ equals(tb1, tb2) ∧ before(te1, te2))]
intervalDuring	$\forall_{T1,T2,tb1,te1,tb2,te2}$ [(intervalDuring(T1, T2) ∧ hasBeginning(T1, tb1) ∧ hasEnd(T1, te1) ∧ hasBeginning(T2, tb2) ∧ hasEnd(T2, te2)) → (ProperInterval(T1) ∧ ProperInterval(T2) ∧ before(tb2, tb1) ∧ before(te1, te2))]
intervalFinishes	$\forall_{T1,T2,tb1,te1,tb2,te2}$ [(intervalFinishes(T1, T2) ∧ hasBeginning(T1, tb1) ∧ hasEnd(T1, te1) ∧ hasBeginning(T2, tb2) ∧ hasEnd(T2, te2)) → (ProperInterval(T1) ∧ ProperInterval(T2) ∧ equals(te1, te2) ∧ before(tb2, tb1))]

Table 4
First-order logic axioms representing the semantics of intervalEquals and the six "primary" Allen's temporal relations

To summarize, this section has shown that the current version of the Time Ontology implements only a small subset of the original axiomatization defined in [32]. Specifically, while all axioms listed in Table 1 have been implemented as RDFS or OWL axioms, those corresponding to the axioms in Table 2, Table 3, and Table 4 remain unimplemented.

Moreover, it is not immediately clear how the unimplemented axioms from [32], or their versions in Table 2, Table 3, and Table 4, could be formalized in RDFS or OWL, if at all. This, as noted above, may explain their absence from the current official version of the Time Ontology.

In this paper, we propose formalizing all axioms from the four tables in SHACL; as will be shown below, this translation is both intuitive and straightforward.

Before presenting the SHACL counterparts of the first-order logic axioms in the four tables, however, the next section will illustrate how to associate xsd:dateTime datatypes with instances of the class Instant. Datatypes

[17] Furthermore, note that all axioms in Table 4 are expressed as implications, whereas the corresponding axioms in [32] are *bi*-implications. The reverse implications could be added to Table 4, but we chose not to do so, as they are irrelevant to the task of validating the resources in Figure 1. Section 7 below elaborates further on these issues and, in particular, explains that SHACL-SPARQL rules should be *task-oriented*: to avoid unnecessary computations, only the minimal set of rules required for the inference task at hand should be defined.

are essential for practical use, as applications rely on standard types such as integers, strings, or, in the case of the Time Ontology, `xsd:dateTime`. This is the sole datatype considered in Figure 1; it is associated with instances of `Instant` through the datatype property `inXSDDateTime`.

The next section also introduces initial SHACL shapes for validating knowledge graphs involving the `inXSDDateTime` property. Crucially, it demonstrates that SHACL shapes alone are insufficient for this task. To achieve the intended validations, most of the first-order logic axioms from the four tables will be implemented as SHACL-SPARQL rules in the subsequent sections.

5. Using SHACL to validate the Time Ontology – why SHACL shapes alone do not suffice

This section focuses on the validation of the datatype property `inXSDDateTime`, which, as noted at the end of the previous section, enables the use of the Time Ontology in practical applications, which requires working with standard datatypes such as `xsd:dateTime`, the range of `inXSDDateTime`.

Therefore, the very first SHACL shape to be imposed is the one in (6), which flags as invalid any object of `inXSDDateTime` that does not conform to the `xsd:dateTime` datatype.

(6)
```
[rdf:type sh:NodeShape;
  sh:targetSubjectsOf time:inXSDDateTime;
  sh:property[sh:path time:inXSDDateTime;
    sh:datatype xsd:dateTime;
    sh:message "Invalid datatype: xsd:dateTime is required"]].
```

Furthermore, since, as mentioned in footnote 14 above, the timezone is *optional* for `xsd:dateTime`, for practical reasons we also added the SHACL shape in (7), which requires the objects of `inXSDDateTime` to specify the timezone. (7) checks whether the timezone is unspecified by using the SHACL Core property `sh:pattern`[18].

(7)
```
[rdf:type sh:NodeShape;
  sh:targetSubjectsOf time:inXSDDateTime;
  sh:property[sh:path time:inXSDDateTime;
    sh:pattern "(Z|(\\+|-)[0-9]2:[0-9]2)$";
    sh:message "Invalid datatype: it does not specify the timezone."]].
```

Next, a SHACL shape is introduced to ensure that only a single `xsd:dateTime` value is assigned to each instant via `inXSDDateTime`. SHACL Core provides the property `sh:maxCount`[19], which allows setting an upper limit on the number of values that can be assigned to an RDF resource. This enables the implementation of the desired shape, as any additional assignments of the property will violate the constraint.

(8)
```
[rdf:type sh:NodeShape;
  sh:targetSubjectsOf time:inXSDDateTime;
  sh:property[sh:path time:inXSDDateTime;
    sh:maxCount 1;
    sh:message "Invalid Instant: multiple xsd:dateTime values are
                associated with this node."]].
```

The three SHACL shapes in (6), (7), and (8) are written in SHACL Core. As noted in the introduction, SHACL Core defines a set of built-in constraints for validating RDF data, e.g., predefined constraint types such as datatype restrictions and cardinality. However, it lacks the expressiveness required for more advanced validation checks.

One example illustrating the expressivity limitations of SHACL Core is the validation of instants that are associated with specific `xsd:dateTime` values and are connected via the properties `equals`, `before`, or `after`. For `equals`, the two `xsd:dateTime` values must be identical; for `before` and `after`, they must respect the corresponding temporal order. For instance, the following triples are invalid:

[18] https://www.w3.org/TR/shacl/#PatternConstraintComponent
[19] https://www.w3.org/TR/shacl/#MaxCountConstraintComponent

(9)
```
:i1 time:inXSDDateTime "2024-01-01T00:00:00Z"^^xsd:dateTime.
:i1 time:equals :i2.
:i2 time:inXSDDateTime "2025-01-01T00:00:00Z"^^xsd:dateTime.
```

It is not possible in SHACL Core to verify that the `xsd:dateTime` values associated with two nodes connected by `equals` are identical, nor to check the temporal ordering required by `before` or `after`. Such cross-node value comparisons fall outside the expressive power of SHACL Core.

In contrast, SHACL-SPARQL can naturally handle such cases. The two SHACL-SPARQL shapes that validate pairs of instants connected by either `equals` or `before` and each associated with an `xsd:dateTime` value via the property `inXSDDateTime`, are shown in (10). A similar shape can also be defined for `after`

(10)
```
[rdf:type sh:NodeShape;
  sh:targetSubjectsOf time:inXSDDateTime;
  sh:sparql[sh:prefixes ...;
    sh:select """SELECT $this ?dt1 ?dt2
                 WHERE{$this time:equals ?i2. FILTER($this!=?i2).
                   $this time:inXSDDateTime ?dt1. ?i2 time:inXSDDateTime ?dt2.
                   FILTER(?dt1!=?dt2)}""";
    sh:message "Invalid Instant {$this}: this instant is declared equal
                to another instant, but the two instants have different
                values, {?dt1} and {?dt2}."]].

[rdf:type sh:NodeShape;
  sh:targetSubjectsOf time:inXSDDateTime;
  sh:sparql[sh:prefixes ...;
    sh:select """SELECT $this ?dt1 ?dt2
                 WHERE{$this time:before ?i2. FILTER($this!=?i2).
                   $this time:inXSDDateTime ?dt1. ?i2 time:inXSDDateTime ?dt2.
                   FILTER(?dt1>=?dt2)}""";
    sh:message "Invalid Instant {$this}: this instant is declared to occur
                before another instant, but it occurs at {?dt1} while
                the other occurs at {?dt2}."]].
```

SHACL-SPARQL shapes incorporate SPARQL queries using the `SELECT-WHERE` structure. Consequently, these shapes tend to be more verbose than SHACL Core shapes. To enhance readability and maintain focus, we will specify only the four key components of SHACL-SPARQL shapes throughout this paper: `sh:target*`, `SELECT`, `WHERE`, and `sh:message`. The shapes in the GitHub repository, being executable, are instead provided in full. For example, the shape in (10) will be represented in a more compact form as:

(11)
```
sh:targetSubjectsOf time:inXSDDateTime;
SELECT $this ?dt1 ?dt2
WHERE{$this time:equals ?i2. FILTER($this!=?i2).
      $this time:inXSDDateTime ?dt1. ?i2 time:inXSDDateTime ?dt2.
      FILTER(?dt1!=?dt2)}""";
sh:message "Invalid Instant {$this}: this instant is declared equal to another
     instant, but the two instants have different values, {?dt1} and {?dt2}."

sh:targetSubjectsOf time:inXSDDateTime;
SELECT $this ?dt1 ?dt2
WHERE{$this time:before ?i2. FILTER($this!=?i2).
      $this time:inXSDDateTime ?dt1. ?i2 time:inXSDDateTime ?dt2.
      FILTER(?dt1>=?dt2)}""";
sh:message "Invalid Instant {$this}: this instant is declared to occur before
     another instant, but it occurs at {?dt1} while the other occurs at {?dt2}."
```

Let us now consider more complex knowledge graphs that also include the properties `hasBeginning` and `hasEnd`. These properties have `TemporalEntity` as their domain and `Instant` as their range; therefore, any RDF resource occurring as their object can be inferred to be an instance of `Instant`. Moreover, since `Instant` is a subclass of `TemporalEntity`, instances of `Instant` may themselves occur as subjects of `hasBeginning` and `hasEnd`. Thus, we can obtain *paths* of `hasBeginning` and `hasEnd` properties of *arbitrary length*, that is, *chains* of instants connected by these two properties. If two of these instants are associated with an `xsd:dateTime` value, these values must, of course, be identical.

An example of such a chain of instants is shown in Figure 1. The knowledge graph in the figure should be detected as invalid because `te1`, `te2`, `te3`, and `te4` are all inferred to be instances of `Instant`; consequently, `te1` and `te4` can only be associated with the same `xsd:dateTime` value, yet in Figure 1 they are not.

Fig. 2. Invalid temporal entities, inferred as instants, connected by a path of `hasBeginning` and `hasEnd` properties.

This validation check can be enforced by the SHACL-SPARQL shape in (12), which is capable, more generally, of detecting paths involving an arbitrary number of `hasBeginning` and `hasEnd` properties and ending with an instant associated with an `xsd:dateTime` value. SPARQL 1.1 provides nine operators for defining *regular expressions* over properties, known as "SPARQL 1.1 Property Paths"[20]. Thus, an arbitrary path of `hasBeginning` and `hasEnd` properties ending with an instant associated with an `xsd:dateTime` value can be represented by the regular expression `(time:hasBeginning|time:hasEnd)+/time:inXSDDateTime`.

(12)
```
sh:targetSubjectsOf time:inXSDDateTime;
SELECT $this ?dt1 ?dt2
WHERE{$this (time:hasBeginning|time:hasEnd)+/time:inXSDDateTime ?dt2.
      $this time:inXSDDateTime ?dt1. FILTER(?dt1!=?dt2)}
sh:message "Invalid Instant {$this}: this instant is declared equal to another
     instant, but the two instants have different values: {?dt1} and {?dt2}."
```

Nevertheless, it was indeed quite straightforward to write a SHACL-SPARQL shape that invalidates knowledge graphs such as the one in Figure 2, i.e., graphs that contain only arbitrary paths of `hasBeginning` and `hasEnd` properties *oriented in the same direction*. This ensures that each temporal entity along the path is an instance of `Instant`, since it occurs as the object of either `hasBeginning` or `hasEnd`.

Conversely, let us now consider the pattern in Figure 3, which depicts a knowledge graph containing an arbitrary number of temporal entities connected by a "zig-zag" path of `hasEnd` properties. By "zig-zag" we mean that the `hasEnd` properties alternate in direction, i.e., they can be traversed both forward and backward along the path.

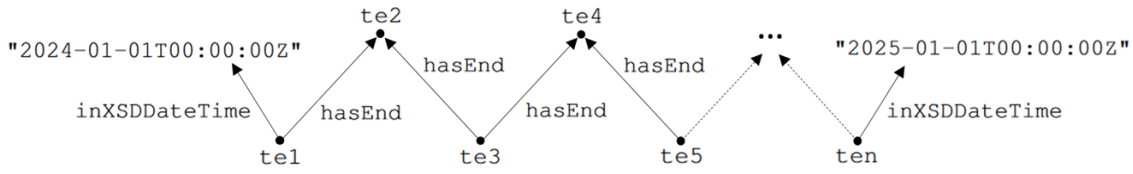

Fig. 3. Temporal entities connected by a "zig-zag" path of `hasEnd` properties. The knowledge graph is invalid only if all entities along the path are instances of `Instant`.

[20]https://www.w3.org/TR/sparql11-property-paths

Unlike the previous case, in Figure 3 it is unknown whether all temporal entities along the path are instances of `Instant`. In particular, we cannot determine whether `te3` and `te5` are instants, since they occur as the *subject* of `hasEnd` rather than as the *object*, and the *domain* of `hasEnd` is the more general class `TemporalEntity`.

However, if all temporal entities in the path, including `te3` and `te5`, are indeed instances of `Instant`, either because this is explicitly asserted or because it can be inferred (for example, from their occurrence as objects of other `hasBeginning` or `hasEnd` properties), then the knowledge graph is invalid. In this case, all these instants must share the same `xsd:dateTime` value, yet in Figure 3 they do not.

It is not possible to define a SHACL-SPARQL shape that detects this invalid pattern for an arbitrary number of nodes using SPARQL 1.1 Property Path operators, even though one such operator ("^") allows traversal in the reverse direction. SPARQL 1.1 Property Paths only permit the composition of regular expressions over the *properties* in the path but do not allow posing additional constraints on the *nodes* of the path, i.e., the RDF resources connected by those properties, which, in the pattern under scrutiny, must all be verified as instances of `Instant`.

To implement the required validation check, the expressivity of the SPARQL 1.1 Property Path operators would need to be extended to allow constraints to be imposed on the RDF resources traversed by the property path. Such an extension could be considered for future versions of SPARQL.

With the current SPARQL 1.1 recommendation, instead, a practical solution for validating the pattern in Figure 3 is to first add inference rules ensuring that, if all temporal entities in the path are instants, they are all inferred to be equal. This can be achieved by adding SHACL-SPARQL rules that implement axioms "3." and "5." in Table 3. Once these inferences are applied, the first SHACL shape in (11) can detect that the knowledge graph is invalid, since `te1` and `ten` would be asserted as equal while being associated with two distinct `xsd:dateTime` values.

This approach is applied in the following sections, where we implement the axioms in the four tables from the previous section as SHACL-SPARQL shapes and SHACL-SPARQL rules. Executing the shapes on the knowledge graphs inferred through the rules yields the intended validation.

In addition, it is worth noting that even when a SHACL-SPARQL shape can be written directly, as in the case of Figure 2, doing so is not always desirable. Regular expressions constructed using SPARQL 1.1 Property Paths operators can quickly become long and complex, making SHACL shapes difficult to read, debug, and maintain, which may hinder the practical adoption of the SHACL standard. For example, the regular expression in (12), although used to validate a relatively simple pattern in knowledge graphs, is already somewhat complex as it involves three properties combined with three SPARQL 1.1 Property Paths operators.

Decoupling validation into two sequential steps (first inference, then validation using *simple* SHACL shapes) is therefore not only necessary in *some* cases but also advantageous in *many* others. This approach results in clearer and more modular validations, improves maintainability, and thus facilitates the rapid development and adaptation of SHACL shapes for evolving knowledge graphs.

6. Implementing the first-order logic axioms from Section 4 in SHACL

The previous section demonstrated that SHACL-SPARQL shapes alone are insufficient to validate arbitrary property paths when additional constraints on the RDF resources along the path must also be enforced. This limitation was illustrated using a "zig-zag" pattern of `hasEnd` properties: if all RDF resources along the pattern are instances of the class `Instant`, they must all be associated with the same `xsd:dateTime` value. However, it is not possible to validate such patterns because the SPARQL 1.1 Property Path operators only define regular expressions over the *properties* in the path, while they do not allow constraining the *nodes* of the path, i.e., the RDF resources connected by those properties. As a result, these nodes remain unconstrained. Thus, in the investigated "zig-zag" pattern, it cannot be verified whether they are all instances of the class `Instant`.

To address the expressivity limitations of SHACL shapes, the validation of knowledge graphs is *decoupled* into two sequential[21] steps: (1) inference and (2) validation.

[21] A similar recent approach is presented in [57], which proposes an algorithm that translates SHACL recursive constraints and OWL 2 QL inferences into stand-alone, non-recursive SHACL. However, this approach does not appear to scale to other inference profiles, such as additional OWL profiles or SHACL-SPARQL rules. In contrast, we advocate a *modular* approach in which inference and validation are clearly separated.

In this paper, SHACL-SPARQL rules are employed for the inference step. However, in general, the inferences could be performed using any other reasoning language for knowledge graphs, e.g., OWL. In other words, while SHACL-SPARQL rules provide a convenient means to represent the first-order logic axioms presented above in Subsection 4.1, we do not claim that they should *always* be used. In some cases, OWL may provide a simpler yet effective solution, while in other cases, even SHACL-SPARQL rules may prove insufficient, necessitating the use of alternative, more expressive reasoning languages.

Let us now illustrate how the semantic characterization presented in Subsection 4.1 is implemented as SHACL-SPARQL rules and SHACL shapes.

It is evident that the properties `equals`, `before`, and `after` define a basic temporal algebra on the class `Instant`. This algebra is straightforward to understand because the semantics of these three properties closely correspond to those of the well-known mathematical operators "=", "<", and ">", the only difference being that the latter operate on integers, while `equals`, `before`, and `after` operate on instants.

Indeed, this basic temporal algebra can be defined solely in terms of the properties `equals` and `before`, because, thanks to axiom "11." in Table 1, every triple involving the property `after` can be transformed into a corresponding triple involving `before`. Without axiom "11.", it would be necessary to define, for `after`, axioms analogous to those for `equals` and `before`.

In light of this, the first SHACL-SPARQL rule presented in this paper is shown in (13). SHACL-SPARQL rules are similar to SHACL-SPARQL shapes, but instead of using `SELECT`, they employ `CONSTRUCT` to generate triples that are added to the knowledge graph, rather than producing error messages. The rule in (13) identifies pairs of RDF resources, `$this` and `?te`, connected by `after`, and adds a triple linking `?te` to `$this` via `before`.

Note that (13) implements only the *implication* from `after` to `before` in axiom "11." of Table 1, not the full *bi-implication*. The reverse implication is unnecessary, since, as explained above, the basic temporal algebra relies only on the properties `equals` and `before`.

(13)
```
sh:targetSubjectsOf time:after;
CONSTRUCT{?te time:before $this}
WHERE{$this time:after ?te}
```

All first-order logic axioms in Table 1, with the exception of axiom "3.", can be readily converted into SHACL-SPARQL rules that capture the inferences implied by RDF Schema. For instance, the rules implementing the `rdfs:domain` and `rdfs:range` properties are shown below. Rules for the other RDF Schema properties are implemented similarly and are available in the GitHub repository.

(14)
```
sh:targetSubjectsOf rdfs:domain;
CONSTRUCT{?s rdf:type ?c}
WHERE{$this rdfs:domain ?c. ?s $this ?o}

sh:targetSubjectsOf rdfs:range;
CONSTRUCT{?o rdf:type ?c}
WHERE{$this rdfs:range ?c. ?s $this ?o}
```

The SHACL-SPARQL rules implementing the properties of RDF Schema are, of course, not specific to the Time Ontology and can be reused in any ontology that employs the `rdfs` prefix. Similarly, one could re-implement any OWL property; for example, (13) could be expressed as a general SHACL rule on the property `owl:inverseOf`. However, while we accept re-implementing as SHACL-SPARQL rules the few RDFS properties that enable inferences, in the formalization presented in this paper we deliberately avoid defining SHACL-SPARQL rules over OWL properties, so as to maintain a clear separation between SHACL and OWL.

Concerning axiom "3." in Table 1, we decided not to implement it because it only adds information that is irrelevant for validating the resources in Figure 1. The axiom asserts that a temporal entity can only be an instant or an interval (or both). Such an assertion would only be invalid if it contradicted a statement that the temporal entity is *not* an instant or an interval; for example, if it were an instance of a class disjoint with them. However, with the four classes in Figure 1, i.e., `TemporalEntity` and its subclasses, it is not possible to make such a negated assertion.

Let us now illustrate how the axioms in Table 2 have been implemented in SHACL. The first axiom is the only one that entails a negated literal. Consequently, this axiom concerns logical consistency, as it infers a triple that *does not* hold; if the triple is instead present in the knowledge graph, a logical inconsistency must be reported.

As noted in the Introduction, logical consistency can be regarded as a form of validation under the assumption that anything inconsistent is also invalid. This assumption is reasonable in many use cases, although not in all (see, e.g., [17], where inconsistencies can be explicitly represented and thus are not considered invalid).

Accordingly, axiom "1." in Table 2 is implemented as a SHACL-SPARQL shape, as follows:

(15)
```
sh:targetSubjectsOf time:before;
SELECT $this
WHERE{$this time:before $this}
sh:message "Invalid triple `{$this} time:before {$this}':
                                              time:before is anti-reflexive."
```

All other first-order logic axioms in Table 2 are implemented as SHACL-SPARQL rules. For example, axiom "2.", which establishes the transitivity of `before`, corresponds to the rule in (16). Axioms "3." and "4.", which capture the symmetry and transitivity of `equals`, are implemented in a similar manner. These are omitted here for brevity, but the reader can find them in the GitHub repository.

(16)
```
sh:targetSubjectsOf time:before;
CONSTRUCT{$this time:before ?te2}
WHERE{$this time:before ?te1. ?te1 time:before ?te2}
```

The remaining four axioms in Table 2, which enforce the preservation of the properties `before`, `hasBeginning`, `hasEnd`, and `inside` under substitution of `equals` in either argument, can be implemented with a single SHACL-SPARQL rule, as follows:

(17)
```
sh:targetSubjectsOf time:equals;
CONSTRUCT{?T1 ?P ?T3}
WHERE{VALUES ?P {time:before time:hasBeginning time:hasEnd time:inside}
      {$this time:equals ?T2. ?T2 ?P ?T3. BIND($this AS ?T1)}UNION
      {$this time:equals ?T3. ?T1 ?P $this. BIND($this AS ?T2)}}
```

The SPARQL 1.1 `VALUES` clause is used to range over the four properties with the variable `?P`. The `UNION` clause allows representing the disjunction in the antecedent; however, since the variable `?this` occurs in different arguments of `?P` in each disjunct, it must be bound to a separate variable using the SPARQL 1.1 `BIND` operator.

We now illustrate how the axioms in Table 3 have been implemented in SHACL. Table 3 also includes an axiom that entails a negated literal, namely axiom "7.". This axiom states that the end of a temporal entity cannot occur before its beginning. In terms of the Time Ontology vocabulary, this means not only that the triple corresponding to `before(te, tb)` cannot be asserted in the knowledge graph, but also that `te`, the temporal entity's end, cannot be associated with an `xsd:dateTime` value that is lower than the `xsd:dateTime` value of its beginning `tb`. This latter constraint is enforced by the following SHACL-SPARQL shape:

(18)
```
sh:targetClass time:TemporalEntity;
SELECT $this
WHERE{$this time:hasBeginning ?tb. $this time:hasEnd ?te.
      ?tb time:inXSDDateTime ?dtb. ?te time:inXSDDateTime ?dte.
      FILTER(?dte<?dtb)}
sh:message "Invalid temporal entity {$this}: it ends before it begins."
```

On the other hand, there is no need to define an additional SHACL shape to check whether the triple corresponding to `before(te, tb)` occurs in the graph. If such a triple were present, axiom "8." would derive `before(te, te)`, while axiom "9." would derive `before(tb, tb)`, both of which would contradict axiom "1." in Table 2.

All other first-order logic axioms in Table 3 have been implemented as SHACL-SPARQL rules. Axiom "1." differs from the other remaining axioms in Table 3 in that it involves existential quantifiers, which appear in the *consequent* of the implication. To translate existential quantifiers occurring in the consequent of an implication, SHACL-SPARQL, like other well-known logic programming languages whose syntax does not include existential quantifiers, such as Prolog, typically requires *Skolemization* of these quantifiers into new, explicit individuals of the domain. In SHACL-SPARQL, these are represented as new anonymous RDF resources, also known as "blank nodes", which can be created in the `WHERE` clause using the function `BNode`.

Nevertheless, it is clear that creating two new nodes would be redundant when the beginning and end of a temporal entity already exist in the knowledge graph. Similarly, if the temporal entity is an instance of the class `Instant`, there is no need to create new blank nodes, since axioms "2." and "3." in Table 3 stipulate that the beginning and end of an instant are the instant itself.

This is actually a common situation when working with knowledge graphs: it is frequently necessary to check whether certain triples already exist in the knowledge graph and to create new blank nodes only if they do not. SPARQL supports this conditional logic through a combination of: the `OPTIONAL` clause, which attempts to match existing triples without failing the query if they are absent; the `IF` function, which enables if-else branches in the `WHERE` clause; and the `EXISTS` and `BOUND` predicates, which test whether certain triples exist and whether a variable is bound, respectively.

Using these operators, the rule corresponding to axiom "1." in Table 3 can be encoded in SHACL-SPARQL as shown below. Note that this rule also incorporates axioms "2." and "3.", specifically the implications that if `t` is an instant, then its beginning and end are `t`.

(19)
```
sh:targetClass time:TemporalEntity;
CONSTRUCT{$this time:hasBeginning ?tb. $this time:hasEnd ?te.
          ?tb a time:Instant. ?te a time:Instant}
WHERE{OPTIONAL{$this time:hasBeginning ?tbopt}
      BIND(IF(EXISTS{$this a time:Instant},$this,
           IF(BOUND(?tbopt),?tbopt,BNode())) AS ?tb)
      OPTIONAL{$this time:hasEnd ?teopt}
      BIND(IF(EXISTS{$this a time:Instant},$this,
           IF(BOUND(?teopt),?teopt,BNode())) AS ?te)}
```

In (19), the `OPTIONAL` clauses attempt to match the beginning and end of `$this`. If these triples exist, the corresponding variables `?tbopt` and `?teopt` are bound to them; otherwise, they remain unbound. The first `BIND` statements implement a three-branch conditional logic using `IF`, `EXISTS`, and `BOUND`. Specifically, for `?tb`, the query first checks whether `$this` is an instance of `Instant` using `EXISTS`; if so, `?tb` is bound directly to `$this`. If not, it checks whether `?tbopt` was bound by the preceding `OPTIONAL` clause; if it was, `?tb` is bound to that existing beginning. If neither condition holds, a new blank node is created via `BNode` and bound to `?tb`. The same logic applies to `?te`. In all cases, the `CONSTRUCT` clause asserts `?tb` and `?te` as the beginning and end of `$this` (note that if `?tb` and `?te` already exist, the corresponding triples are simply reasserted) and as instances of `Instant`. As it will be explained in Section 8 below, asserting them as instances of `Instant` ensures that the SHACL-SPARQL rule in (19) does not loop infinitely.

All remaining axioms in Table 3, as well as those in Table 4, do not differ significantly from the axioms already discussed. Their translation into SHACL-SPARQL rules is therefore implemented in a similar way. For example, the SHACL-SPARQL rule corresponding to axiom "11." in Table 3 is the following:

(20)
```
sh:targetSubjectOf time:inside;
CONSTRUCT{?tb time:before ?t. ?t time:before ?te}
WHERE{$this time:inside ?t. $this time:hasBeginning ?tb. $this time:hasEnd ?te}
```

The axiom associated with `intervalDuring` in Table 4 is instead implemented as follows:

```
(21) sh:targetSubjectOf time:intervalDuring;
     CONSTRUCT{$this a time:ProperInterval. ?T2 a time:ProperInterval.
               ?tb2 time:before ?tb1. ?te1 time:before ?te2}
     WHERE{$this time:intervalDuring ?T2. $this time:hasBeginning ?tb1.
           $this time:hasEnd ?te1. ?T2 time:hasBeginning ?tb2. ?T2 time:hasEnd ?te2}
```

The SHACL-SPARQL rules for all other axioms can be found in the GitHub repository associated with this paper.

6.1. Examples

This section has shown how the semantic characterization axiomatized in first-order logic in Subsection 4.1 can be implemented in SHACL. Before proceeding, it is useful to present some examples of the execution of SHACL-SPARQL rules followed by SHACL shapes, to enhance understanding.

We begin with the "zig-zag" pattern shown in Figure 3, which illustrates that SHACL shapes alone are insufficient to capture the semantics of the target first-order logic axiomatization. While Figure 3 depicts an *abstract* "zig-zag" pattern with an *arbitrary* number of temporal entities `te1...ten`, we focus here on a knowledge graph with a *finite* number of nodes, specifically, seven temporal entities `te1...te7`, encoded in RDF as follows:

```
(22) :te1 time:inXSDDateTime "2026-01-01T00:00:00Z"^^xsd:dateTime.
     :te1 time:hasEnd :te2. :te3 time:hasEnd :te2. :te3 a time:Instant.
     :te3 time:hasEnd :te4. :te5 time:hasEnd :te4. :te5 a time:Instant.
     :te5 time:hasEnd :te6. :te7 time:hasEnd :te6.
     :te7 time:inXSDDateTime "2025-01-01T00:00:00Z"^^xsd:dateTime.
```

The temporal entities `te1`, `te2`, `te4`, `te6`, and `te7` are all inferred as instances of `Instant` by the SHACL-SPARQL rule defined on `rdfs:domain` and `rdfs:range` shown above in (14). All these temporal entities occur either as the subject of `inXSDDateTime` or as the object of `hasEnd`. The two remaining temporal entities, `te3` and `te5`, are instead *explicitly* asserted as instances of `Instant` in (22). Since `te1`, ..., `te7` are all instants, the SHACL-SPARQL rule shown above in (19) infers that their beginnings and ends are themselves. In other words, the rule shown in (19) above adds the following triples to the knowledge graph:

```
(23) :te1 time:hasBeginning :te1. :te1 time:hasEnd :te1.
     :te2 time:hasBeginning :te2. :te2 time:hasEnd :te2.
     ...
     :te7 time:hasBeginning :te7. :te7 time:hasEnd :te7.
```

Next, the SHACL-SPARQL rule implementing axiom "5." in Table 3, not shown in the paper but available on GitHub, infers that the instants are pairwise equal, i.e., it adds the following triples to the knowledge graph:

```
(24) :te1 time:equals :te2. :te3 time:equals :te2.
     :te3 time:equals :te4. :te5 time:equals :te4.
     :te5 time:equals :te6. :te7 time:equals :te6.
```

Then, since `equals` is a symmetric and transitive relation, as enforced by the SHACL-SPARQL rules corresponding to axioms "3." and "4." in Table 2, all seven instants are inferred to be equal to one another. Finally, since `te1` is associated with 1st January 2026 while `te7` is associated with 1st January 2025 in (22), the first SHACL shape in (11) detects that the knowledge graph is invalid.

The second example of an invalid knowledge graph is shown in Figure 4. In the figure, `i2` and `i3` are, respectively, the beginning and the end of the temporal entity `te1`. However, it is not specified whether `te1` is an instant or a proper interval. All that can be inferred, given axiom "7." in Table 3, is that `i3` cannot occur before `i2`: either the two instants are equal, or `i2` occurs before `i3`.

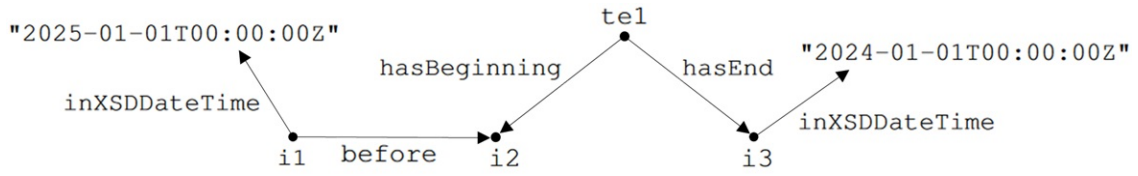

Fig. 4. Invalid knowledge graph.

The SHACL shape associated with axiom "7." in Table 3 is unable to invalidate the knowledge graph, since only `i3` is associated with an `xsd:dateTime` value. Nevertheless, the graph is indeed invalid because `i2` occurs after another instant, `i1`, which is associated with an `xsd:dateTime` value greater than the one associated with `i3`.

Once again, the SHACL-SPARQL rules make it possible to compute an inferred graph that is subsequently invalidated by the SHACL shapes. Specifically, axiom "8." in Table 3 derives that, since `i1` occurs before `i2`, it must also occur before `i3`. The second SHACL shape presented above in (11) then identifies the knowledge graph in Figure 4 as invalid, since the `xsd:dateTime` values associated with the two instants contradict the inferred `before` property, which is directed from `i1` to `i3`.

A similar, though more complex, example of an invalid knowledge graph, which nonetheless does not involve `xsd:dateTime` values, is shown in Figure 5.

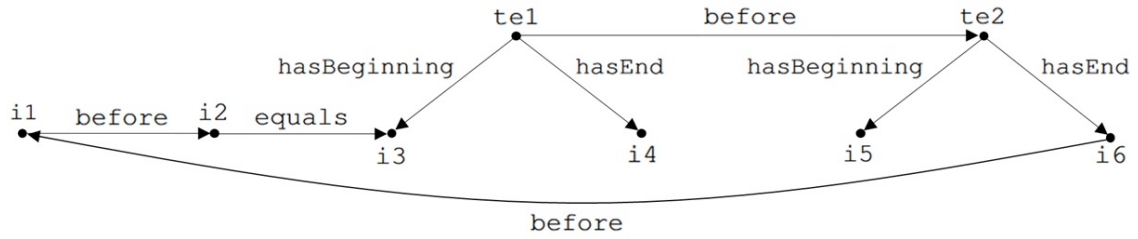

Fig. 5. Invalid knowledge graph.

Again, it is unknown whether the temporal entities `te1` and `te2` are instants rather than temporal intervals. However, since `te1` occurs before `te2`, in light of axiom "10." in Table 3, it is inferred that `te1`'s end instant occurs before `te2`'s start instant. On the other hand, the SHACL-SPARQL rule in (17), enforcing the preservation of `before` under the substitution of `equals` in either of its arguments, infers that `i1` occurs before `i3`. The following triples are therefore added to the knowledge graph in Figure 5.

(25) `:i4 time:before :i5. :i1 time:before :i3.`

The SHACL-SPARQL rules corresponding to axioms "8." and "9." in Table 3 then infer that `i4` occurs before `i6`, `i5` occurs before `i1`, `i1` occurs before `i4`, and `i3` occurs before `i5`. Thus, a cyclic path of `before` properties is inferred among `i1`, `i4`, and `i5`. From this, the SHACL-SPARQL rule in (16), which enforces the transitivity of `before`, infers that `i1`, `i4`, and `i5` occur before themselves; this is detected as invalid by the SHACL shape in (15), which enforces the anti-reflexivity of `before`.

Note that the original first-order logic axiomatization in [32] is incapable of detecting the knowledge graph in Figure 5 as invalid (or, more precisely, inconsistent), even though it should, since the graph, as noted above, does not associate any instant with specific `xsd:dateTime` values and thus falls within the intended scope of [32]'s axiomatization. This limitation arises because [32] does not include axioms corresponding to axioms "8." and "9." in Table 3, which we added specifically to propagate the `before` assertions from the temporal entities' beginnings to their ends, and vice versa. As will also be explained in the next section, when dealing with abstract temporal entities, i.e., entities for which it is unknown whether they are instants or proper intervals, it is not sufficient to verify that their ends do not occur before their beginnings, as stated in axiom "7." in Table 3 (taken from [32]) prescribes; it

is also necessary to check whether a path of `before`, and possibly `equals`, properties exists from their ends to their beginnings, since such a path would create a cycle, as illustrated in Figure 5. Axioms "8." and "9." have been introduced precisely for this purpose, namely to propagate assertions from the beginnings to the ends and vice versa, without directly linking these beginnings to the ends.

The final example of an invalid knowledge graph is shown in Figure 6. The graph in the figure involves only properties representing Allen's temporal relations.

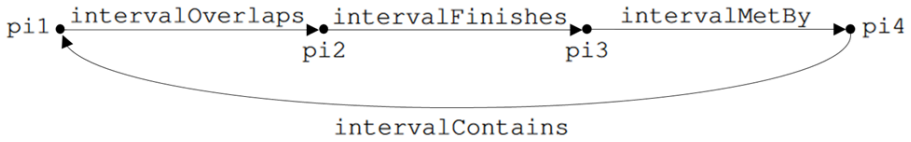

Fig. 6. Invalid knowledge graph involving properties representing Allen's temporal relations.

First, the SHACL-SPARQL rule in (19) creates two instants for each of the four proper intervals involved, one corresponding to the beginning and one to the end of the interval. The triples in (27) are added to the knowledge graph. `_:b11`, `_:b12`, etc., are the blank nodes created by the rule in (19); to improve readability, their indices have been adjusted only to correspond to the associated proper intervals. The rule also asserts these nodes as instances of `Instant` and, when re-applied to them, infers that their beginnings and ends are themselves; however, the additional triples resulting from this re-application are omitted from (27).

(26)
```
:pi1 time:hasBeginning _:b11. :pi1 time:hasEnd _:b12.
:pi2 time:hasBeginning _:b21. :p12 time:hasEnd _:b22.
:pi3 time:hasBeginning _:b31. :p13 time:hasEnd _:b32.
:pi4 time:hasBeginning _:b41. :p14 time:hasEnd _:b42.
```

The property `intervalMetBy`, from `pi3` to `pi4`, and the property `intervalContains`, from `pi4` to `pi1`, are respectively converted into their inverse properties: `intervalMeets`, from `pi4` to `pi3`, and `intervalDuring`, from `pi1` to `pi4`.

Now, the SHACL-SPARQL rules corresponding to the axioms in Table 4 can be applied. The rule for `intervalOverlaps` infers that `_:b11` occurs before `_:b21`, that `_:b21` occurs before `_:b12`, and that `_:b12` occurs before `_:b22`. The rule for `intervalFinishes` infers that `_:b22` is equal to `_:b32` and that `_:b31` occurs before `_:b21`. The rule for `intervalMeets` infers that `_:b42` is equal to `_:b31`. Finally, the rule for `intervalDuring` infers that `_:b41` occurs before `_:b11` and that `_:b12` occurs before `_:b42`. These triples are listed below, with each line showing the triples inferred by one of the four rules.

(27)
```
_:b11 time:before _:b21. _:b21 time:before _:b12. _:b12 time:before _:b22.
_:b22 time:equals _:b32. _:b31 time:before _:b21
_:b42 time:equals _:b31.
_:b41 time:before _:b11. _:b12 time:before _:b42.
```

The triples in (27) form a cycle, which can be described as follows (the mathematical symbols "=" and "<" are used here in place of `equals` and `before`):

(28) `_:b12 < _:b42 = _:b31 < _:b21 < _:b12`

Then, the SHACL-SPARQL rules enforcing anti-reflexivity, transitivity, and the preservation of `before` under substitution of `equals` in either argument are once again able to infer that the knowledge graph is invalid.

7. Extending the proposed SHACL axiomatization for the Time Ontology

The previous sections presented a SHACL axiomatization for validating the RDF resources shown in Figure 1, adapted from the original first-order logic axiomatization in [32].

The rationale for this validation is straightforward: as outlined at the beginning of Section 6, the properties `equals`, `before`, and `after` define a basic temporal algebra over the class `Instant`, which resemble closely the semantics of the mathematical operators "=", "<", and ">" (with ">" being redundant, as it can be expressed as the inverse of "<"). The RDF triples involving the resources in Figure 1 are thus translated into constraints within this basic temporal algebra through SHACL-SPARQL rules.

Once all `before` (<) and `equals` (=) relations have been identified, the basic temporal algebra must satisfy the following requirements:

- When two instants are connected by `before` or `equals` and are associated with `xsd:dateTime` values, the ordering of these values must be consistent with the temporal ordering denoted by the property. This is enforced via the SHACL shapes in (11).
- The property `before` is anti-reflexive. This is enforced by the SHACL shape in (15).

As explained above, the remaining RDF resources in Figure 1 impose only constraints within this basic temporal algebra. For instances, for the classes `Instant` and `ProperInterval`, the following holds:

- `Instant`: the beginning of the instance must be equal (=) to its end.
- `ProperInterval`: the beginning of the instance must precede (<) its end.

Concerning the thirteen properties representing Allen's temporal relations, they impose additional "<" and "=" constraints on the endpoints of the two proper intervals they relate, as specified in Table 4. The same holds for the property `inside`, which imposes additional "<" constraints between its object instant and the beginning and end of the temporal entity serving as its subject.

The class `TemporalEntity`, on the other hand, poses some challenges, as it subsumes both `Instant` and `ProperInterval`. Consequently, if an RDF resource is known only to belong to `TemporalEntity`, it is not possible to infer constraints between its beginning and end using `before` (<) or `equals` (=). Introducing a new property, such as `beforeOREquals`, corresponding to "$\leqslant$", would address this issue; it could be axiomatized via SHACL-SPARQL rules as transitive and, when both arguments are equal, also reflexive and symmetric.

However, we chose not to extend the Time Ontology vocabulary with such a property. Instead, we added axioms "8." and "9." in Table 3 to propagate `before` assertions across endpoints. In terms of "$\leqslant$", these axioms capture the implications $((t1 \leqslant t2) \wedge (t2 < t3)) \rightarrow (t1 < t3)$ and $((t1 < t2) \wedge (t2 \leqslant t3)) \rightarrow (t1 < t3)$. As a result, they enable the detection of invalid knowledge graphs, such as the one in Figure 5 above. It is worth noting that these graphs are *not* flagged as inconsistent by [32], since that axiomatization includes only axiom "7." in Table 3 but lacks mechanisms for propagating temporal constraints between endpoints, analogous to axioms "8." and "9.".

Therefore, compared to [32], our SHACL formalization indeed provides a more comprehensive and effective validation of the `before`, `equals`, and `inside` properties, the classes `TemporalEntity`, `Instant`, and `ProperInterval`, and the thirteen properties representing Allen's temporal relations.

Building on this foundation, although our focus in this paper has been limited to validating the RDF resources in Figure 1, additional rules could be introduced to enable further inferences. For example, two SHACL-SPARQL rules could be added to implement the following first-order logic axioms:

(29) $\forall_{T,t1,t2}[(\texttt{equals}(tb, te) \wedge \texttt{hasBeginning}(T, tb) \wedge \texttt{hasEnd}(T, te)) \rightarrow \texttt{Instant}(T)]$
$\forall_{T,t1,t2}[(\texttt{before}(tb, te) \wedge \texttt{hasBeginning}(T, tb) \wedge \texttt{hasEnd}(T, te)) \rightarrow \texttt{ProperInterval}(T)]$

These two axioms were not included in the axiomatization presented in the previous sections because they are not relevant to the validation task. As explained above, to validate the RDF resources in Figure 1, only the constraints on the basic temporal algebra for `equals` (=) and `before` (<) are needed. The two axioms in (29) do not extract any such constraints; rather, they implement complementary inferences: if the knowledge graph specifies that the

beginning and end of a temporal entity are connected by `equals` rather than `before`, these axioms infer the category of the temporal entity, i.e., `Instant` rather than `ProperInterval`.

These two rules may therefore be useful in applications that require categorizing temporal entities; otherwise, their inclusion merely enriches the knowledge graph for the sake of completeness, without providing any practical benefit in the present context. For this reason, they were not included in the previous sections, where the primary focus was on validating the knowledge graphs constructed from the resources shown in Figure 1.

A more concrete example comes from the RDF-based framework recently proposed in [58] for reasoning about obligations, permissions, and other deontic statements. Incorporating the Time Ontology into the framework described in [58] is left for future work, with the aim of enabling inferences such as determining that, in (30.a-b), John violated the prohibition on entering the park, *but only from 4pm to 5pm*.

(30) a. It is prohibited to enter the park from 3pm until 5pm.

b. John was in the park from 4pm until 6pm.

To draw this inference, we need to introduce an additional SHACL-SPARQL rule that, given two proper intervals that overlap, *creates* a new instance of `ProperInterval` representing the interval shared by the two overlapping intervals (unless this proper interval already exists). This rule could be:

(31)
```
sh:targetSubjectsOf time:intervalOverlaps;
CONSTRUCT{?pi3 a time:ProperInterval.
          ?pi3 time:hasBeginning ?b. ?pi3 time:hasEnd ?e.
          ?pi3 time:intervalFinishes $this. ?pi3 time:intervalStarts ?pi2}
WHERE{$this time:intervalOverlaps ?pi2.
      ?pi2 time:hasBeginning ?b. $this time:hasEnd ?e.
      OPTIONAL{?p time:hasBeginning ?b. ?p time:hasEnd ?e}
      BIND(IF(BOUND(?p), ?p, BNODE()) AS ?pi3)}
```

Given the two proper intervals `[3pm, 5pm]` and `[4pm, 6pm]`, the rule in (31) infers and generates the *new* proper interval `[4pm, 5pm]`, which is the interval to be associated with the violation. Naturally, adding the rule in (31) to the axiomatization from the previous section would also be irrelevant for the purpose of validating the RDF resources in Figure 1; this rule is required only in the proposed extension of the framework in [58].

The discussions made so far in this section suggest that SHACL-SPARQL rules should be *task-oriented*. In other words, we should not attempt to include every possible rule to derive all conceivable knowledge. Striving for exhaustiveness would only increase computational costs and make updates and debugging more difficult, since a larger number of rules would need to be maintained and monitored. Instead, we should focus on the specific task at hand, such as validating the semantics ascribed to a given set of RDF resources, as done in this paper, rather than inferring additional intervals as in the example in (30); we should define only the minimal number of SHACL-SPARQL rules necessary for that task.

Adding further rules is not the only possible way to broaden the scope of the axiomatization presented above. Another possible direction is to extend the *vocabulary* of the Time Ontology with new RDF resources.

For example, one could introduce RDF resources capable of representing *vectors* of Allen's temporal relations and complement them with inference rules that implement Allen's propagation algorithm. However, as noted in Section 3, this may not always be advisable, since Allen's propagation algorithm for the full temporal algebra has exponential complexity. A more pragmatic approach would therefore be to restrict the extension to a *tractable* subalgebra, such as the one proposed in [56], where reasoning can be performed in polynomial time. However, this direction requires substantial further research, which we may address in future work.

Another possible extension of the Time Ontology's vocabulary, which we have actually advocated above, concerns the representation of infinite intervals. As previously discussed, [32] model infinite intervals as those in which one or both endpoints are *explicitly* omitted. In that approach, if the beginning of an interval is missing, it is assumed to be $-\infty$; if the end is missing, it is assumed to be $+\infty$. We chose not to adopt this approach because, as explained above, we consider it difficult to reconcile with the Open World Assumption, which is central to RDF semantics. In our axiomatization, every temporal entity is instead required to have both endpoints, which may take the values

$-\infty$, $+\infty$, or a specific `xsd:dateTime` value via the property `inXSDDateTime`. An endpoint may also lack a value, in which case this value is interpreted as "unknown", in accordance with the Open World Assumption.

Since $-\infty$ and $+\infty$ cannot be represented using the `xsd:dateTime` datatype, we introduce two new classes, `minusInfinite` and `plusInfinite`, to represent instants with these values. In other words, instants that are instances of these classes are intended to denote instants whose values are $-\infty$ and $+\infty$, respectively.

Additionally, SHACL shapes are introduced to invalidate knowledge graphs that contain instants with values $-\infty$ or $+\infty$. In particular, a knowledge graph is considered invalid if:

(32)
a. An instant belongs to both classes `minusInfinite` and `plusInfinite`, or to only one of them while also being associated with an `xsd:dateTime` value. This enforces that an instant cannot simultaneously have the values $-\infty$ and $+\infty$, nor can it have either of these while also being associated with a finite `xsd:dateTime` value.

b. An instant belongs to the class `minusInfinite` and also occurs as the object of `hasEnd`, or it belongs to the class `plusInfinite` and also occurs as the object of `hasBeginning`. This enforces that temporal entities cannot begin at $+\infty$ or end at $-\infty$.

c. An instant belongs to either the class `minusInfinite` or `plusInfinite` and is connected via the property `equals` to another instant that either belongs to the opposite class or is associated with an `xsd:dateTime` value. This enforces that an instant cannot be $-\infty$ or $+\infty$ and, at the same time, be equal to another instant associated with the infinite value of the opposite sign or with a finite `xsd:dateTime` value. A similar constraint applies to the property `before`: no instant may occur before $-\infty$ or after $+\infty$, including other instants that are themselves associated with these values.

The shapes implementing (32.a) are shown in (33); the `UNION` clause captures both options described in (32.a).

(33)
```
sh:targetClass time:minusInfinite;
SELECT $this
WHERE{{$this a time:plusInfinite}UNION{$this time:inXSDDateTime ?dt}}
sh:message "Invalid Instant {$this}: this instant's value is both -infinite
            and either +infinite or a finite xsd:dateTime value."

sh:targetClass time:plusInfinite;
SELECT $this
WHERE{{$this a time:minusInfinite}UNION{$this time:inXSDDateTime ?dt}}
sh:message "Invalid Instant {$this}: this instant's value is both +infinite
            and either -infinite or a finite xsd:dateTime value."
```

(32.b) is implemented by the following two SHACL shapes; the `FILTER` clause ensures that the case where an infinite instant both begins and ends at itself, as enforced by axioms "2." and "3." in Table 3, is not reported as invalid by the shapes, since these triples are considered acceptable.

(34)
```
sh:targetClass time:minusInfinite;
SELECT $this ?T
WHERE{?T time:hasEnd $this. FILTER((?T!=$this)&&!EXISTS{?T time:equals $this})}
sh:message "Invalid TemporalEntity {?T}: it ends at -infinite."

sh:targetClass time:plusInfinite;
SELECT $this ?T
WHERE{?T time:hasBeginning $this.
      FILTER((?T!=$this)&&!EXISTS{?T time:equals $this})}
sh:message "Invalid TemporalEntity {?T}: it begins at +infinite."
```

Finally, (32.c) is implemented by the following two SHACL shapes; as in the SHACL shape in (33), the `UNION` clauses allow to account for all possible cases.

(35)
```
sh:targetSubjectsOf time:equals;
SELECT $this ?t
WHERE{$this time:equals ?t.
      {$this a time:minusInfinite
       {?t a time:plusInfinite}UNION{?t time:inXSDDateTime ?dt}}UNION
      {$this a time:plusInfinite
       {?t a time:minusInfinite}UNION{?t time:inXSDDateTime ?dt}}}
sh:message "Invalid Instant {$this}: it is associated with an infinite
            value but is also equal to another instant with a value
            incompatible with this infinite value."

sh:targetSubjectsOf time:before;
SELECT $this
WHERE{{$this a time:plusInfinite}UNION
      {$this time:before ?t. ?t a time:minusInfinite}}
sh:message "Invalid Instant $this: it either is +infinite and another
            instant occurs after it, or it occurs before another instant
            associated with -infinite."
```

The classes `minusInfinite` and `plusInfinite`, together with the shapes in (33), (34), and (35), extend the basic temporal algebra defined in the previous sections for the properties `equals` ($=$) and `before` ($<$). While the algebra in the previous sections encompassed only finite `xsd:dateTime` values, these two new classes also allow it to include the values $-\infty$ and $+\infty$. The new shapes then check for and invalidate the following patterns involving instants at $-\infty$ and $+\infty$ (assuming "dt" is some specific `xsd:dateTime` value, while "$\star$" represents "any value", i.e., either $-\infty$, $+\infty$, or dt):

- $-\infty = +\infty$
- $-\infty = \mathtt{dt}$
- $+\infty = \mathtt{dt}$
- $[\star, -\infty]$
- $[+\infty, \star]$
- $+\infty < *$
- $* < -\infty$

An example of invalid knowledge graph is the following:

(36)
```
:T time:hasBeginning :tb. :tb a time:plusInfinite. :tb time:before :t.
```

These triples are invalid for two reasons. First, the instant `tb`, whose value is $+\infty$, is specified as the beginning of the temporal entity `T`. As correctly detected by the second SHACL shape in (34), this is not allowed: a temporal entity cannot begin at $+\infty$ (in symbols: "$[+\infty, \star]$" is not allowed). Second, the triples in (36) indicate that `tb` occurs before the temporal entity `t`. As correctly detected by the second SHACL shape in (35), this is also invalid, because $+\infty$ cannot occur before any other instant (in symbols: "$+\infty < *$" is not allowed).

8. Beyond the Time Ontology: challenges and risks of using SHACL-SPARQL rules

This paper has demonstrated the high expressivity of SHACL, and in particular of SPARQL, which is embedded within SHACL. This expressivity enables the implementation, within the Time Ontology, of inferences that are significantly more advanced than those currently supported in the ontology's official OWL-encoded version.

However, the expressive power of SPARQL also introduces potential challenges and risks. Although the SHACL-SPARQL rules proposed above for the Time Ontology are not affected by these issues, as will be explained below, this final section discusses them more generally to caution readers about the potential pitfalls of using SHACL-SPARQL rules on RDF ontologies.

Specifically, the use of SHACL-SPARQL rules involves two main risks: (1) the potential for infinite loops, and (2) the possibility of non-deterministic outcomes. The following subsections examine these issues in detail and explain why, in the set of SHACL-SPARQL rules presented above, they either do not occur or have no practical effect.

8.1. SHACL-SPARQL rules may generate infinite loops

As explained at the end of the Introduction, in this paper inference is performed in the same way as standard OWL reasoners, such as HermiT, where OWL axioms are repeatedly applied until no further triples can be inferred.

On a *fixed* set R of RDF resources, of which P⊆R are properties, the maximum (finite) number of possible triples that can be created is R×P×R. Once all such triples among the R resources have been generated, no further triples can be added, ensuring that the process terminates.

However, SHACL-SPARQL rules can also *create* new individuals in the CONSTRUCT clause, thereby *expanding* the set of resources R. As a result, re-executing these rules may indeed lead to an infinite loop.

A simple illustrative example of a SPARQL rule in CONSTRUCT-WHERE form that triggers an infinite loop via repeated execution is the following:

(37)
```
CONSTRUCT{?f a :Man. ?m :friend-of ?f}
WHERE{?m a :Man. BIND(BNode() AS ?f)}
```

Consider an initial knowledge graph containing a single man. The WHERE clause in (37) creates a new anonymous individual, binds it to the variable ?f, and asserts that it is both a man and a friend of the first man. Therefore, after the first execution of the rule, the knowledge graph contains two men. Re-executing the rule adds two more men as friends of the two existing ones, then four, then eight, and so on, resulting in exponential growth that ultimately generates an infinite number of men.

Of course, more complex patterns of SHACL-SPARQL rules that generate infinite loops are possible. For instance, we could have a set of n rules, where the first rule produces a new individual that matches the WHERE clause of the second rule, the second rule produces a new individual matching the WHERE clause of the third rule, and so on, until the last rule, which then produces a new individual matching the WHERE clause of the first rule.

It is therefore evident that rules creating new anonymous individuals must be carefully checked. It is the responsibility of the knowledge engineer to determine, when such rules are included in the axiomatization, whether they might generate infinite loops and, if so, to include conditions in their WHERE clauses to "break" the loop.

In the SHACL axiomatization of the Time Ontology described above, only a single SHACL-SPARQL rule generates new anonymous individuals. This is the rule in (19) above, which is associated with the first-order logic axiom "1." in Table 3. For the reader's convenience, both the rule and the first-order logic axiom are repeated here:

(38) $\forall_{\mathtt{T}}[\mathtt{TemporalEntity(T)} \rightarrow \exists_{\mathtt{tb,te}}[\mathtt{hasBeginning(T,tb)} \wedge \mathtt{hasEnd(T,te)}]$

```
sh:targetClass time:TemporalEntity;
CONSTRUCT{$this time:hasBeginning ?tb. $this time:hasEnd ?te.
          ?tb a time:Instant. ?te a time:Instant}
WHERE{OPTIONAL{$this time:hasBeginning ?tbopt}
      BIND(IF(EXISTS{$this a time:Instant},$this,
           IF(BOUND(?tbopt),?tbopt,BNode())) AS ?tb)
      OPTIONAL{$this time:hasEnd ?teopt}
      BIND(IF(EXISTS{$this a time:Instant},$this,
           IF(BOUND(?teopt),?teopt,BNode())) AS ?te)}
```

If the first-order logic axiom in (38) were implemented in SHACL-SPARQL exactly as written, the rule would result in an infinite loop: for each temporal entity T, two new temporal entities, tb and te, would be generated. These new entities would re-enter the rule, producing two additional temporal entities, then four, then eight, and so on, as in the previous example in (37).

To avoid this infinite loop, we observed that: (1) the two new individuals tb and te must be instances of Instant, since they are asserted in the CONSTRUCT clause as objects of hasBeginning and hasEnd; and (2) the beginning and the end of an instant are the instant itself, as stipulated by axioms "2." and "3." in Table 3. Based on these observations, we were able to expand the rule as shown in (38) without altering the intended semantics. The version in (38) does not generate infinite loops, because all newly created anonymous individuals are asserted as instances of Instant; therefore, when they re-enter the same rule, they satisfy the first branch of the IF condition, and thus no additional individuals are generated.

Nevertheless, the mechanism used in (38) to prevent infinite loops is, of course, an ad-hoc and non-generalizable solution. It was easy to implement due to the specific semantics we wish to ascribe to the Time Ontology, and the fact that (38) is the only rule in the axiomatization that creates anonymous individuals.

For other RDF ontologies or alternative semantics, the SHACL axiomatization may involve a larger number of SHACL-SPARQL rules that create anonymous individuals. In such cases, the knowledge engineer must demonstrate that these rules, when executed together, never result in infinite loops. When the interaction patterns become more complex, this may require supplementing the axiomatization with formal proofs to guarantee termination.

8.2. SHACL-SPARQL rules may lead to non-deterministic outcomes

SHACL-SPARQL rules behave like a standard rule-production system. However, not all sets of SHACL-SPARQL rules behave as a *monotonic* rule-production system, in which adding new information does not invalidate previous conclusions and, consequently, the set of inferred triples does not depend on the execution order of the rules.

In fact, the SPARQL vocabulary contains non-deterministic operators that vary their outcome based on the presence of certain triples in the knowledge graph, which may be produced by other rules. Examples of such operators include `EXISTS`, `IF`, and `OPTIONAL`. These operators can lead to non-deterministic behavior, depending on whether the rules that generate the triples they test are executed before or after them. A simple example is the following pair of SHACL-SPARQL rules:

(39)
```
CONSTRUCT{?m a :Lonely}
WHERE{NOT EXISTS{?m :friend-of ?f}}

CONSTRUCT{?m1 :friend-of ?m2}
WHERE{?m1 a :Man. ?m2 a :Man. FILTER(?m1!=?m2)}
```

The `WHERE` clause of the first rule in (39) is satisfied by every individual who has no friends; the rule infers that each such individual is an instance of the class `Lonely`. The second rule in (39), on the other hand, searches the knowledge graph for pairs of men and, for each pair, asserts that the two men are friends.

It is easy to see that the two rules in (39) may produce different outcomes depending on the order in which they are executed. Consider, for example, a knowledge graph containing two men: `John` and `Jack`. If the first rule in (39) is executed first, both `John` and `Jack` are inferred to be lonely men. If the second rule is executed first, `John` and `Jack` are inferred to be friends, which prevents the first rule from applying; in other words, if the second rule is executed first, `John` and `Jack` are *not* inferred to be lonely men.

The non-deterministic behavior of the two rules in (39) is due to the `NOT EXISTS` clause, which is the SPARQL operator used to implement *negation-as-failure*. As is well known, negation-as-failure is a non-monotonic operator; therefore, if the triples specified within the `NOT EXISTS` clause do not exist in the knowledge graph, i.e., if they are *unknown*, the clause evaluates to true.

Other SPARQL operators can produce similar effects, thereby emulating negation-as-failure. For instance, the `IF` operator, used to create if-else conditions in the `WHERE` clause, yields one outcome if certain conditions are met and another outcome otherwise. However, these conditions might be entailed by other rules, which may be executed before or after the rule containing the `IF` clause. The same considerations apply to the `OPTIONAL` operator, which binds variables to certain RDF resources when the corresponding triples exist; again, these triples may be entailed by other rules, which may be executed before or after the rule containing the `OPTIONAL` clause.

As with the problem of infinite loops, it is again the responsibility of the knowledge engineer to verify either that the set of SHACL-SPARQL rules is monotonic or that, in cases where multiple outcomes are produced, the intended semantics is preserved in all of them. Such verification again requires an ad-hoc formal analysis of the specific axiomatization, the results of which are not always generalizable.

To address non-determinism, the SHACL-SPARQL vocabulary provides the property `sh:order`, which enables prioritization of rules. By defining a specific execution order, non-deterministic behavior can be avoided. This strategy has been applied, for example, in [59].

In the axiomatization proposed above for the Time Ontology, it was not necessary to prioritize the rules using `sh:order`, because only a single SHACL-SPARQL rule includes non-deterministic operators. Although this rule

may indeed produce different outcomes in a few cases, none of these outcomes affects the intended semantics or, more generally, the task of validating the Time Ontology resources shown in Figure 1.

This rule, again, is the one in (38). As explained in the previous subsection, it may or may not create new instants and assign them as the beginning and end of a temporal entity, depending on whether the temporal entity is itself an instant or already specifies its beginning or end. However, the classification of a temporal entity as an instant, or the specification of its beginning or end, may be entailed by other rules; consequently, depending on whether these rules are executed before or after (38), the latter may or may not create new anonymous individuals.

In particular, the proposed axiomatization includes three rules that may interfere with (38) in this way. These are the rules corresponding to axioms "4.", "5.", and "6." in Table 1, which infer a temporal entity to be an instance of `Instant` if the entity occurs as the object of the properties `hasBeginning`, `hasEnd`, or `inside`.

An example of a knowledge graph that may yield two different inferred graphs, depending on the execution order of the aforementioned rules, is the following:

(40)
```
:T time:hasBeginning :tb. :tb a time:TemporalEntity.
```

Since `tb` occurs as the object of `hasBeginning`, and the range of `hasBeginning` is the class `Instant`, if the second rule in (14) above, defined on `rdfs:range`, executes first, `tb` is inferred as an instance of the class `Instant`. Therefore, when (38) executes on `tb`, it is set as both the beginning and the end of itself. With this rule execution order, the inferred knowledge graph is then the following:

(41)
```
:T time:hasBeginning :tb. :tb a time:TemporalEntity.
:tb a time:Instant. :tb time:hasBeginning :tb. :tb time:hasEnd :tb.
```

By contrast, if (38) executes on `tb` before (14), two new anonymous individuals are created and then inferred as instances of `Instant`, as well as as the beginning and end of `tb`. Then, `tb` is also inferred as instance of `Instant` by the second rule in (14) and its beginning and end sets to `tb` itself. Finally, the SHACL-SPARQL rules corresponding to axioms "4." and "5." in Table 3 set `tb` equal to the two anonymous individuals, which are then inferred to be equal to each other. The inferred knowledge graph is then the following:

(42)
```
:T time:hasBeginning :tb. :tb a time:TemporalEntity.
:tb time:hasBeginning :_b0. :tb time:hasEnd :_b1.
:_b0 a time:Instant. :_b1 a time:Instant. :tb a time:Instant.
:tb time:hasBeginning :tb. :tb time:hasEnd :tb.
:tb time:equals :_b0. :tb time:equals :_b1. :_b0 time:equals :_b1.
```

Since all instants are inferred to be equal, it is easy to see that the SHACL shapes presented in the previous sections yield the same results on both (41) and (42). According to the basic temporal algebra implemented by these shapes on the properties `equals` (=) and `before` (<), a knowledge graph is invalidated if and only if `tb`, and any other instant equal to it, are assigned different `xsd:dateTime` values.

Again, it was relatively straightforward to explain that the potential non-deterministic behavior of the proposed axiomatization does not affect the validation of the Time Ontology resources shown in Figure 1: only a single, and rather simple, rule had to be examined, the same that could potentially engender infinite loops. This rule is the only one that both creates new anonymous individuals and utilizes non-deterministic SPARQL operators.

Nevertheless, it is clear that investigating more complex axiomatizations, which include multiple rules that create anonymous individuals and/or employ non-deterministic SPARQL operators such as `EXISTS`, `IF`, or `OPTIONAL`, requires considerably more care and should ideally be accompanied by detailed formal proofs.

9. Conclusions and future works

This paper introduced a novel SHACL axiomatization for the Time Ontology. Effective time management is crucial in both academia and industry, as it underpins a wide range of real-world applications. The Time Ontology is widely recognized as the "de facto" standard for representing temporal data in the Semantic Web. However, its

current OWL-based version primarily serves as a terminological vocabulary, providing standardized symbols to label instants, intervals, and other temporal concepts. While harmonizing and standardizing these symbols is essential for interoperability and data sharing, the ontology itself offers only limited inferencing capabilities, which can lead to inconsistencies such as intervals that end before they start.

The SHACL axiomatization we proposed is inspired by and built upon the first-order logic axiomatization defined by Jerry R. Hobbs and Feng Pan, the original proponents of the Time Ontology, in [32]. Although their axiomatization was introduced about twenty years ago, it has never been fully implemented in OWL or other Semantic Web formalisms such as SWRL, perhaps due to the limited expressivity of these formalisms to capture its complexity. By contrast, using SHACL, the implementation of these axioms becomes relatively straightforward.

However, rather than adopting Hobbs and Pan's axiomatization verbatim, we defined a variant that retains most of the original axioms while introducing several key extensions. Three main reasons underpin this choice:

(43) a. Hobbs and Pan represent infinite intervals as intervals that *explicitly* lack one or both endpoints. As discussed in Section 7 and earlier, we consider this approach highly incompatible with the Open World Assumption, a fundamental principle of RDF semantics. As an alternative, we propose extending the Time Ontology's vocabulary by introducing two additional classes to represent the values $-\infty$ and $+\infty$, along with SHACL shapes to validate knowledge graphs containing these values.

b. Hobbs and Pan's axiomatization does not employ the `equals` property, which is part of the Time Ontology vocabulary, nor does it allow associating instants with explicit finite `xsd:dateTime` values. We introduce SHACL shapes and SHACL-SPARQL rules to address these gaps.

c. We identified a few patterns of invalid knowledge graphs that Hobbs and Pan's axiomatization does not flag as inconsistent (e.g., Figure 5). Our axiomatization introduces two additional axioms to detect and mark these patterns as invalid.

The extended first-order logic axiomatization incorporating (43.a–c) constitutes a further contribution of this paper. Overall, we believe our work opens new opportunities for temporal data validation and AI-driven reasoning over temporal knowledge in the Semantic Web.

In addition, our research journey on defining the SHACL axiomatization of the Time Ontology helped us identify broader insights into SHACL itself, particularly regarding the interplay between validation and inference. In other words, the SHACL axiomatization of the Time Ontology that we developed has actually served as a *case study* for SHACL, illustrating how the W3C standard can be applied effectively.

First, we demonstrated that SHACL shapes alone are insufficient to detect certain invalid knowledge graphs that can be constructed using the Time Ontology vocabulary. A notable example is the "zig-zag" pattern shown in Figure 3, which is particularly illustrative, as it can arise using only the RDF class `Instant` and the properties `inXSDDateTime` and `hasEnd`. Therefore, simply put, SHACL shapes cannot fully validate knowledge graphs built from even a single class and two properties of the Time Ontology vocabulary. This limitation stems from the fact that SPARQL 1.1 Property Paths, the only SHACL construct for examining subgraphs of arbitrary length, can define regular expressions over properties but cannot simultaneously impose constraints on the RDF nodes connected by these properties. Although this empirical finding pertains specifically to the Time Ontology, its implications appear broader: if SHACL shapes cannot adequately validate relatively simple knowledge graphs built with this vocabulary, similar limitations can reasonably be expected in more complex scenarios.

To overcome this limitation, we argue that validation should be performed on the fully inferred knowledge graph, i.e., the graph obtained by iteratively applying inference rules until no new triples are derived. This two-step procedure is not prescribed by the W3C Working Group Note (08 June 2017)[22], which, in fact, even appears to suggest the reverse sequence: first validate the graph, then apply the rules[23].

More broadly, our work highlights the general need to balance inference and constraint validation in SHACL. One possible solution is the definition of SHACL dialects or profiles, similar to the profiles defined for OWL (OWL

[22] https://www.w3.org/TR/shacl-af

[23] See https://www.w3.org/TR/shacl-af/#rules-examples; note, however, that this section is non-normative (personal communication with Holger Knublauch).

Lite, OWL DL, OWL2 EL, etc.)[24]. Defining different profiles would allow practitioners to select an appropriate level of expressivity while managing computational complexity. The current W3C SHACL recommendation already mentions support for different *entailment regimes*[25], but only the SPARQL 1.1 entailment regimes[26] are listed, and their support is even indicated as optional.

Defining additional profiles or entailment regimes that accept only specific patterns of SHACL-SPARQL rules poses a significant research challenge and demands substantial effort from the Semantic Web research community.

As discussed in the previous section, two main risks must be considered when iterating SHACL-SPARQL rules until no further triples are inferred: SHACL-SPARQL rules can produce infinite loops when creating new anonymous individuals or non-deterministic outcomes when using SPARQL operators such as `EXISTS`, `IF`, and `OPTIONAL`. To ensure termination and either deterministic or harmless non-deterministic outcomes, different profiles or entailment regimes could restrict the creation of new individuals and the use of these non-deterministic operators, allowing them only in patterns that avoid infinite loops and harmful non-determinism.

However, it should be noted that this paper does not claim that SHACL-SPARQL rules must *always* be used for reasoning over knowledge graphs. In some cases, OWL or other reasoning languages provide simpler or more cost-effective solutions, while in others, even SHACL-SPARQL may lack sufficient expressivity. For example, implementing Allen's propagation algorithm to realize the full version of Allen's temporal algebra requires *cycles* over intervals [28], which SHACL-SPARQL cannot handle. In such cases, SHACL-X[27], which integrates SHACL with JavaScript, appears to provide the necessary expressive power.

More generally, the choice of the inference language, whether OWL, SHACL-SPARQL, SHACL-X, or a dialect of these, should be guided by the expressivity required for a given use case, while avoiding unnecessary complexity. Defining dialects and entailment regimes for SHACL, similar to what has been done with OWL profiles over the past decades, could enable targeted implementations and specialized reasoners optimized for each regime.

In sum, this paper not only proposes a SHACL axiomatization for the Time Ontology but also contributes to a broader understanding of the interplay between inference and validation in the Semantic Web, paving the way for more systematic exploration and practical tooling in this area.

Acknowledgments

We sincerely thank Maxime Jakubowski, Jose Emilio Labra Gayo, Francesco Compagno, and three anonymous reviewers for their insightful feedback and valuable suggestions.

References

[1] V. Ermolayev, S. Batsakis, N. Keberle, O. Tatarintseva and A. Grigoris, Ontologies of Time: Review and Trends., *International Journal of Computer Science & Applications* **11**(3) (2014).

[2] Y. Shoham, Temporal logics in AI: Semantical and ontological considerations, *Artificial intelligence* **33**(1) (1987).

[3] D. Gabbay, I. Hodkinson and M. Reynolds, *Temporal Logic (Vol. 1): Mathematical Foundations and Computational Aspects*, Oxford University Press, Inc., USA, 1994.

[4] D. Gabbay, M. Reynolds and M. Finger, *Temporal Logic (Vol. 2): Mathematical Foundations and Computational Aspects*, Oxford University Press, United Kingdom, 2000.

[5] K. Rozier, Linear temporal logic symbolic model checking, *Computer Science Review* **5**(2) (2011).

[6] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella, Nusmv 2: An opensource tool for symbolic model checking, in: *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings 14*, Springer, 2002, pp. 359–364.

[7] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri and S. Tonetta, The nuXmv symbolic model checker, in: *Computer Aided Verification: 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings 26*, Springer, 2014, pp. 334–342.

[24] https://www.w3.org/2007/OWL/wiki/Profile_Explanations, https://www.w3.org/TR/owl2-profiles

[25] https://www.w3.org/TR/shacl/#shacl-rdfs

[26] https://www.w3.org/TR/sparql11-entailment

[27] https://github.com/SHACL-X/shacl-x

[8] G. Antoniou and F. Van Harmelen, *A Semantic Web primer*, MIT press, 2004.
[9] F. Manola, E. Miller, B. McBride et al., RDF primer, *W3C recommendation* **10**(1–107) (2004), 6.
[10] D. McGuinness and F. Van Harmelen, OWL web ontology language overview, *W3C recommendation* (2004).
[11] F. Baader, I. Horrocks, C. Lutz and U. Sattler, *Introduction to description logic*, Cambridge University Press, 2017.
[12] P. Hitzler, M. Krötzsch, B. Parsia, P.F. Patel-Schneider, S. Rudolph et al., OWL 2 web ontology language primer, *W3C recommendation* **27**(1) (2009), 123.
[13] C. Lutz, F. Wolter and M. Zakharyaschev, Temporal Description Logics: a survey, in: *15th International Symposium on Temporal Representation and Reasoning*, IEEE, 2008.
[14] A. Artale and E. Franconi, A survey of temporal extensions of description logics, *Annals of Mathematics and Artificial Intelligence* **30** (2000), 171–210.
[15] A. Artale, R. Kontchakov, F. Wolter and M. Zakharyaschev, Temporal Description Logic for Ontology-Based Data Access, in: *Proc. of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.
[16] A. Steen and C. Benzmuller, What are Non-classical Logics and Why Do We Need Them? An Extended Interview with Dov Gabbay and Leon van der Torre, *Künstliche Intelligenz* **To appear** (2024).
[17] L. Robaldo and G. Pozzato, Handling irresolvable conflicts in the Semantic Web: an RDF-based conflict-tolerant version of the Deontic Traditional Scheme, *The Journal of Logic and Computation (to appear)* (2025).
[18] H. Knublauch and D. Kontokostas, Shapes Constraint Language (SHACL), *W3C recommendation* (2017).
[19] P. Pareti and G. Konstantinidis, A review of SHACL: From data validation to schema reasoning for RDF graphs, *Reasoning Web International Summer School* (2021).
[20] B. Bogaerts, M. Jakubowski and J. Van den Bussche, SHACL: A Description Logic in Disguise, in: *Logic Programming and Nonmonotonic Reasoning*, G. Gottlob, D. Inclezan and M. Maratea, eds, Springer International Publishing, 2022.
[21] J. Corman, J. Reutter and O. Savković, Semantics and Validation of Recursive SHACL, Springer-Verlag, Berlin, Heidelberg, 2018.
[22] M. Andresel, J. Corman, M. Ortiz, J. Reutter, O. Savkovic and M. Simkus, Stable Model Semantics for Recursive SHACL, in: *Proc. of The Web Conference 2020*, Association for Computing Machinery, 2020.
[23] U. Şimşek, K. Angele, E. Kärle, O. Panasiuk and D. Fensel, Domain-Specific Customization of Schema.org Based on SHACL, in: *Proc. of 19th International Semantic Web Conference (ISWC)*, Springer-Verlag, Berlin, Heidelberg, 2020.
[24] L. Robaldo, Towards compliance checking in reified I/O logic via SHACL, in: *Proc. of 18th International Conference for Artificial Intelligence and Law (ICAIL 2021)*, J. Maranhão and A.Z. Wyner, eds, ACM, 2021.
[25] J. Anim, L. Robaldo and A. Wyner, A SHACL-Based Approach for Enhancing Automated Compliance Checking with RDF Data, *Information* **15**(12) (2024).
[26] N. Ferranti, J. de Souza, S. Ahmetaj and A. Polleres, Formalizing and Validating Wikidata's Property Constraints using SHACL and SPARQL, *Semantic Web journal* **to appear** (2024).
[27] S. Cox, Time ontology extended for non-Gregorian calendar applications, *Semantic Web* **7**(2) (2016).
[28] J. Allen, Towards a General Theory of Action and Time, *Artificial Intelligence* **23**(2) (1984).
[29] M. Vilain, H. Kautz and P. van Beek, Constraint Propagation Algorithms for Temporal Reasoning: A Revised Report, in: *Readings in qualitative reasoning about physical systems*, Morgan Kaufmann Publishers Inc., 1990, pp. 373–381.
[30] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean et al., SWRL: A semantic web rule language combining OWL and RuleML, *W3C Member submission* **21**(79) (2004), 1–31.
[31] B. Glimm, I. Horrocks, B. Motik, G. Stoilos and Z. Wang, HermiT: An OWL 2 Reasoner, *Journal of Automated Reasoning* **53**(3) (2014).
[32] J.R. Hobbs and F. Pan, An ontology of time for the semantic web, *ACM Transactions on Asian Language Information Processing* **3**(1) (2004).
[33] D. Wu, H. Wang and A. Tansel, A survey for managing temporal data in RDF, *Information Systems* (2024).
[34] R. Fikes and Q. Zhou, A reusable time ontology, in: *AAAI-2002 Workshop on Ontologies and the Semantic Web*, Citeseer, 2002.
[35] J. Pustejovsky, R. Ingria, R. Sauri, J.M. Castaño, J. Littman, R.J. Gaizauskas, A. Setzer, G. Katz and I. Mani, The Specification Language TimeML., 2005.
[36] R. Baumann, F. Loebe and H. Herre, Ontology of time in GFO, in: *Formal Ontology in Information Systems*, IOS Press, 2012, pp. 293–306.
[37] V. Milea, F. Frasincar and U. Kaymak, tOWL: a temporal web ontology language, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **42**(1) (2011), 268–281.
[38] S.-K. Kim, M.-Y. Song, C. Kim, S.-J. Yea, H.C. Jang and K.-C. Lee, Temporal ontology language for representing and reasoning interval-based temporal knowledge, in: *The Semantic Web: 3rd Asian Semantic Web Conference, ASWC 2008, Bangkok, Thailand, December 8-11, 2008. Proceedings. 3*, Springer, 2008, pp. 31–45.
[39] N. Keberle, Y. Litvinenko, Y. Gordeyev and V. Ermolayev, Ontology evolution analysis with OWL-MeT, in: *Proceedings of the International Workshop on Ontology Dynamics (IWOD-07)*, 2007, pp. 1–12.
[40] V. Ermolayev, N. Keberle and W.-E. Matzke, An upper level ontological model for engineering design performance domain, in: *Conceptual Modeling-ER 2008: 27th International Conference on Conceptual Modeling, Barcelona, Spain, October 20-24, 2008. Proceedings 27*, Springer, 2008, pp. 98–113.
[41] V. Ermolayev, N. Keberle and W.-E. Matzke, An ontology of environments, events, and happenings, in: *2008 32nd Annual IEEE International Computer Software and Applications Conference*, IEEE, 2008, pp. 539–546.
[42] Y. Raimond, S.A. Abdallah, M.B. Sandler and F. Giasson, The Music Ontology., in: *ISMIR*, Vol. 2007, Vienna, Austria, 2007, p. 8th.
[43] C. Gutierrez, C. Hurtado and A. Vaisman, Temporal rdf, in: *The Semantic Web: Research and Applications: Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29–June 1, 2005. Proceedings 2*, Springer, 2005, pp. 93–107.

[44] M.J. O'Connor and A.K. Das, A method for representing and querying temporal information in OWL, in: *International joint conference on biomedical engineering systems and technologies*, Springer, 2010, pp. 97–110.
[45] C. Tao, W.-Q. Wei, H.R. Solbrig, G. Savova and C.G. Chute, CNTRO: a semantic web ontology for temporal relation inferencing in clinical narratives, in: *AMIA annual symposium proceedings*, Vol. 2010, American Medical Informatics Association, 2010, p. 787.
[46] S. Batsakis and E.G. Petrakis, SOWL: a framework for handling spatio-temporal information in OWL 2.0, in: *Rule-Based Reasoning, Programming, and Applications: 5th International Symposium, RuleML 2011–Europe, Barcelona, Spain, July 19-21, 2011. Proceedings 5*, Springer, 2011, pp. 242–249.
[47] H.-U. Krieger, A General Methodology for Equipping Ontologies with Time., in: *LREC*, 2010.
[48] M. Klein, D. Fensel, A. Kiryakov and D. Ognyanov, Ontology versioning and change detection on the web, in: *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web: 13th International Conference, EKAW 2002 Sigüenza, Spain, October 1–4, 2002 Proceedings 13*, Springer, 2002, pp. 197–212.
[49] J. Tappolet and A. Bernstein, Applied temporal RDF: Efficient temporal querying of RDF data with SPARQL, in: *The Semantic Web: Research and Applications: 6th European Semantic Web Conference, ESWC 2009 Heraklion, Crete, Greece, May 31–June 4, 2009 Proceedings 6*, Springer, 2009, pp. 308–322.
[50] C. Welty, R. Fikes and S. Makarios, A reusable ontology for fluents in OWL, in: *FOIS*, Vol. 150, 2006, pp. 226–236.
[51] S. Batsakis, E.G. Petrakis, I. Tachmazidis and G. Antoniou, Temporal representation and reasoning in OWL 2, *Semantic Web* **8**(6) (2017), 981–1000.
[52] S. Batsakis, I. Tachmazidis and G. Antoniou, Representing time and space for the semantic web, *International Journal on Artificial Intelligence Tools* **26**(03) (2017), 1750015.
[53] A. Preventis, E.G. Petrakis and S. Batsakis, Chronos Ed: A tool for handling temporal ontologies in Protege, *International Journal on Artificial Intelligence Tools* **23**(04) (2014), 1460018.
[54] B. Glimm, I. Horrocks, B. Motik, G. Stoilos and Z. Wang, HermiT: an OWL 2 reasoner, *Journal of automated reasoning* **53** (2014), 245–269.
[55] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur and Y. Katz, Pellet: A practical owl-dl reasoner, *Journal of Web Semantics* **5**(2) (2007), 51–53.
[56] B. Nebel and H. Bürckert, Reasoning about temporal relations: a maximal tractable subclass of Allen's interval algebra, *Journal of the ACM (JACM)* **42**(1) (1995).
[57] S. Ahmetaj, M. Ortiz, A. Oudshoorn and M. Simkus, Reconciling SHACL and Ontologies: Semantics and Validation via Rewriting, in: *ECAI 2023 - 26th European Conference on Artificial Intelligence*, K. Gal, A. Nowé, G. Nalepa, R. Fairstein and R. Radulescu, eds, Frontiers in Artificial Intelligence and Applications, Vol. 372, IOS Press, 2023.
[58] L. Robaldo, S. Batsakis, R. Calegari, F. Calimeri, M. Fujita, G. Governatori, M. Morelli, F. Pacenza, G. Pisano, K. Satoh, I. Tachmazidis and J. Zangari, Compliance checking on first-order knowledge with conflicting and compensatory norms - a comparison among currently available technologies, *Artificial Intelligence and Law* **32** (2023).
[59] L. Robaldo, F. Pacenza, J. Zangari, R. Calegari, F. Calimeri and G. Siragusa, Efficient compliance checking of RDF data, *Journal of Logic and Computation* **33**(8) (2023).