

A Temporal Extension of the Hayes and ter Horst Entailment Rules for RDFS and OWL*

Hans-Ulrich Krieger

German Research Center for Artificial Intelligence (DFKI GmbH)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
krieger@dfki.de

Abstract

Temporal encoding schemes using RDF and OWL are often plagued by a massive proliferation of useless “container” objects. Reasoning and querying with such representations is extremely complex, expensive, and error-prone. We present a temporal extension of the Hayes and ter Horst entailment rules for RDFS/OWL. The extension is realized by extending RDF triples with further temporal arguments and requires only some lightweight forms of reasoning. The approach has been implemented in the forward chaining engine *HFC*.

Introduction

This paper first and foremost presents a temporal extension of the Hayes and ter Horst entailment rules (Hayes 2004; ter Horst 2005) for RDFS and OWL which is able to derive new safe facts, persisting over time. To achieve this intuitively and efficiently, we argue that the extension of relation instances with time needs to abandon the concept of an RDF triple in favor of general tuples, in our case quintuples. We show that the extension of RDFS and parts of OWL (the so-called OWL Horst dialect) only requires lightweight reasoning capabilities. The extension has been implemented in the forward chaining engine *HFC* which is comparable to OWLIM and Jena, but also supports arbitrary tuples, user-defined tests and actions, and a special kind of “aggregation” rules. More information on *HFC* can be found in (Krieger and Kruijff 2011).

Motivation

The decision of the Semantic Web/Web 2.0 community to favor RDF and OWL as standards has proved to be useful. Given this decision, we have experienced several advantages. Firstly, OWL upper ontologies have been made available that are relatively easy to interface with domain ontologies developed for projects. Secondly, tableaux-based DL reasoners can be used to check the consistency of a knowledge base on a regular basis or to query its information.

*The research described here has been financed by the European Integrated projects **CogX** (cogx.eu), **NIFTi** (nifti.eu), and **Monnet** (monnet-project.eu) under contract numbers FP7 ICT 215181, 247870, and 248458. I would like to thank the reviewers for their comments.

Copyright © 2011, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Unfortunately many projects have found that the descriptive power of OWL is too weak to formulate (rule-based) knowledge that is able to discover new and important information. This, in part, explains the popularity of forward chaining reasoners, such as OWLIM or Jena, which capture important parts of OWL together with the possibility to encode domain-specific knowledge within the *same* rule formalism.

Furthermore, and very importantly, today’s projects are more and more dealing with temporally changing information, but encounter difficulties when trying to represent this information in (existing) ontologies. At the same time, reasoning support for this kind of emerging information is also not provided in existing DL reasoners.

Summing up, these experiences have led to the insight that we actually want to *directly* extend a RDF relation instance with time and to make the original entailment rules for RDFS and OWL (Hayes 2004; ter Horst 2005) sensitive to temporal information. This means, however, that an RDF triple has to be replaced by a more general *tuple* representation. This, together with the fact that the computation of a proper temporal extent needs *aggregates* such as minimum and maximum, has resulted in the development of the aforementioned forward chainer *HFC*.

Related Approaches

We relate our approach here to already existing frameworks. For a much broader *overall* picture that is couched in terms of first-order predicate calculus and which includes incomplete and defeasible reasoning through temporal reason maintenance, we let the reader refer to (Dean and McDermott 1987).

Temporal Databases

With the development and practical application of SQL, many people realized the need to add temporal information to entries in database tables (Snodgrass 2000). Temporal databases distinguish between *valid time* (the *interval* in which a fact is true) and *transaction time* (the time when the database transaction happens). Valid time admits right-open intervals, and in principle, a left bound is also not required. Our approach to follow is much in the spirit of *valid time*, except that it comes with rules operating over tuples of the

database (ABox) in order to support RDFS- and OWL-based reasoning as well as providing domain-dependent rules.

Temporal Description Logic

Temporal aspects in description logics have been addressed in the past by various forms of *Temporal description logics* (TDLs). Very often, TDLs are constructed as a combination of a standard description logic (e.g., \mathcal{ALC}) with a standard temporal logic (e.g., LTL); see (Lutz, Wolter, and Zakharyashev 2008). The usual interpretation \mathcal{I} for concepts, roles, and individuals is replaced by a temporal interpretation \mathfrak{S} that extends the denotation by a further temporal argument (usually a natural number), interpreted as a time point. For instance, $(n, \text{john}^{\mathfrak{S}}, \text{mary}^{\mathfrak{S}}) \in \text{marriedWith}^{\mathfrak{S}}$ means that at time n , (john, mary) is an instance of the marriedWith relation. An important variant of TDLs then extends ABox formulae by adding the standard LTL modal operators. For instance, $\text{FScrap}(\text{mycar})$ means that there will be a time n , where my car is scrapped, and for $m \geq n$, $(m, \text{mycar}^{\mathfrak{S}}) \in \text{Scrap}^{\mathfrak{S}}$ is the case.

Unfortunately, we have experienced in many projects that an instant-based approach is not what people want: information extraction from natural language texts, for instance, is best couched in an interval-based approach using (potentially underspecified) calendar time or through topological temporal interval relations (Allen 1983), but not through modal operators and a hidden temporal dimension. To the best of our knowledge, we are not aware of any implemented TDL-based reasoner (for temporal ABoxes).

Annotation Properties in OWL

Many OWL ontologies define temporal concepts like `TemporalInterval`, but are *not* able to temporally annotate relation instances (e.g., represented as RDF triples or binary relations) directly with instances from these classes. Indeed, OWL defines the concept of an *annotation property*, but these properties can only be associated with classes, properties, or individuals, but not with relation instances. Thus, an ABox instance such as `(john, mary) : marriedWith` can not be equipped with temporal starting and ending values.

OWL Time

OWL Time is first and foremost a *first-order* axiomatization of time. The original document (Hobbs and Pan 2004) does *not* formulate axioms in terms of OWL, but requires the full expressive power of first-order predicate logic, including universal and existential quantification, disjunction, functions, and relations with more than two arguments. The good thing with OWL Time is that an ontology of time written in OWL can take advantage of the concepts and properties defined in OWL Time. Thus, *parts* of OWL Time can be encoded and reformulated in OWL.

OWL Time as such, however, does not give an answer to the following question (and it is not intended so): *how do we extend OWL relation instances with time*, or more general, *how do we equip “tenseless” OWL ontologies with a concept of time* (hopefully without “rewriting” the original ontology). Answers to this question are summarized below.

Approaches Staying inside RDF/OWL

Several proposals have been presented in the literature to equip (binary) relation instances with time:

1. *use further temporal arguments*
2. *use a “meta-logical” predicate*
3. *reify original relations*
4. *wrap range arguments* (W3C: N-ary relations)
5. *encode a perdurantist/4D view* (Welty and Fikes 2006)
6. *interpret individuals as time slices* (Krieger, Kiefer, and Declerck 2008)

(1.) is clearly the *silver bullet of representation*, as it is the most natural and intuitive approach, requiring the least overhead in terms of time (during reasoning/querying) and space (for the representation). (2.), as used, e.g., in the situation calculus, requires the original relation (relational fluent) to be reformulated as a function (functional fluent). However, (1.) and (2.) are outside the expressive means of OWL, a function-free two-variable variant of first-order logic.

The proposals (3.)–(6.) have already been implemented in OWL. It is worth noting that (3.)–(5.) enforce a knowledge engineer to rewrite an ontology, whereas (6.) marries arbitrary ontologies with time by introducing perdurants that possess time slices (the original individuals) onto which a temporal extent is defined. As a consequence of using RDF triples, or equivalently, by sticking to binary relation instances, (3.)–(6.) end up in a massive proliferation of useless “container” objects. Reasoning and querying with such representations is extremely complex, expensive, and error-prone.

Our Approach

As outlined above, we will extend Hayes-/ter Horst-style entailment rules by a temporal dimension. Thus, in our case, we replace a RDF *triple* by a *quintuple*, since the starting and ending time of a “temporalized” fact are encoded as separate arguments.

In a certain sense, we are still dealing with RDF triples in case we are not interested in the temporal extent of a fact or in case the temporal information is underspecified or even unspecified. So, speaking in terms of RDF, the first argument of a quintuple must come from the domain of the predicate (second argument), and the third argument is required to fall into the range.

In addition, certain RDF triples still remain triples, since we only extend information from the ABox of an ontology—we will *not* equip TBox information with a temporal extension, say, that the subtype relationship between two classes only holds for some period of time, or that a URI reference should be regarded as a property at a specific time period and as a class at a different time.

From a common sense viewpoint, we also exclude identification statements between individuals (`owl:sameAs`) to be extended by a temporal dimension—once individuals have been identified, it is assumed that they are identical for their whole lifetime. However, typing information (`rdf:type`) is usually assigned a temporal duration, due to the fact that

people often encode binary relation instances through class membership. For instance, `(car, red) : hasColor` might equally be represented as `car : Red`, whereas `Red` refers to the class of objects having color red.

What this Paper is NOT About

Several points are worth mentioning here. **Firstly**, we are not dealing with *duration time* in order to resolve expressions like *Monday* or *20 days* against valid time, when further information comes in. This needs to be handled by a richer temporal ontology and temporal arithmetic. **Secondly**, *temporal quantification*, such as in *four hours every week*, is beyond the expressive means of our approach. **Thirdly**, even though *underspecified time* is handled by our implementation through wildcards in the XSD `dateTime` format (e.g., year missing in *Over New Year's Eve, I have visited the Eiffel Tower*), we do not focus on this here. The solution requires to make certain rule tests sensitive towards the fact that time is now only partially ordered. These tests then return *true*, *false*, or *don't-know*, whereas only *true* indicates that the test succeeds, leading to the instantiation of the RHS of the rule. **Fourthly**, coalescing temporal information (i.e., building larger intervals) should be addressed in custom rules and should not be regarded as part of the RDFS/OWL rule set, since this functionality depends on the (semantic) nature of predicates. **Finally**, certain temporal inferences such as $p(\vec{x}, s, t)$ entails $p(\vec{x}, s', t')$ in case $s \leq s' \leq t' \leq t$ should *not* be handled in the below rules, since termination of the computation of the deductive closure is no longer guaranteed. Such information can only be obtained on the query level. It is worth noting that such entailments assume (as we do) that temporal intervals are convex, i.e., contain no “holes”.

Metric Linear Time

The rules below assume that the temporal measuring system is based on a one-dimensional *metric linear time*, so that we can compare starting/ending points, using operators, such as `<`, or pick out input arguments in aggregates, using `min` or `max`. We are neutral as to whether time is dense or discrete, or whether the metric uses real, rational, or natural numbers. These decisions do not change the effects of rules, since the predicates and aggregates that are used in the rules are independent of the underlying metric. In the implementation of *HFC*, long integers are used to encode milli or even nano seconds w.r.t. a fixed starting point. Alternatively, the XSD `dateTime` format can be used which provides an arbitrarily fine precision, if needed.

Extended Entailment Rules

In the following, we describe a temporal extension of the entailment rules from (Hayes 2004) and (ter Horst 2005). The rules are written in the concrete syntax of *HFC*, so they slightly differ from (Hayes 2004) and (ter Horst 2005) (who also use slightly different notations). We will not talk here about *literals*, *resources*, and *container membership properties*, thus omitting rules dealing with this information.

Due to space limitations, we are only able to display four extended entailment rules. We further note that some of

the original rules have not been extended by temporal arguments (e.g., `rdfs5`), since they only deal with TBox axiom schemes. The below notation can be seen as an extension of N-Triples with two further temporal arguments. The rules make use of further tests (`@test`) which need to be fulfilled to successfully instantiate the RHS. Rules might also be equipped with an action section (`@action`) that binds RHS-only variables to values returned by functions.

rdfl This is the only type statement that is not assigned a temporal extent, since once `?p` has been recognized as a property, it is assumed that this is always the case.

```
?s ?p ?o ?b ?e
->
?p rdf:type rdf:Property
```

rdfs2 The next rule assigns a type to a URI in domain position. The starting and ending time is taken over from the original relation instance, representing the given safe temporal information.

```
?s ?p ?o ?b ?e
?p rdfs:domain ?dom
->
?s rdf:type ?dom ?b ?e
```

Now comes the more interesting part. Up to now, RDFS rules have been extended by only moving around starting/ending information to positions in the consequent of a rule. The two OWL rules below make use of lightweight tests and aggregates.

rdfp1a and rdffp1b We have complemented the original rule `rdffp1` dealing with object properties by a new rule that also addresses datatype properties. Let us start with the assumption that the object is either a URI or a blank node, exactly what the original rule encodes in its *where* condition.

```
?p rdf:type owl:FunctionalProperty
?p rdf:type owl:ObjectProperty
?x ?p ?y ?b1 ?e1
?x ?p ?z ?b2 ?e2
->
?y owl:sameAs ?z
@test
IntervalNotEmpty ?b1 ?e1 ?b2 ?e2
```

The `IntervalNotEmpty` predicate in the test section (`@test`) guarantees that we only *identify* `?y` and `?z` if the temporal extent of $p(x, y, b_1, e_1)$ and $p(x, z, b_2, e_2)$ has a *non-empty intersection* (we use the relational notation here):

```
IntervalNotEmpty begin1 end1 begin2 end2 ≡
begin := max(begin1, begin2)
end := min(end1, end2)
return (begin ≤ end)
```

Thus a single overlapping observation leads to a *total* identification of `?y` and `?z` (at all times!), so the `sameAs` statement need not be equipped with temporal information. Even though our (my!) commonsense indicates that this is the right choice, the decision is, in principle, debatable.

If both observations, however, do talk about different *non-intersecting* times, it makes perfect sense that `?y` and `?z` need *not* be equal, even though `?p` is a *functional* property (example: `marriedWith` relation).

Let us now focus on the second rule **rdfp1b**, dealing with functional datatype properties.

```
?p rdf:type owl:FunctionalProperty
?p rdf:type owl:DatatypeProperty
?x ?p ?y ?b1 ?e1
?x ?p ?z ?b2 ?e2
->
?x rdf:type owl:Nothing ?b ?e
@test
?y != ?z
IntervalNotEmpty ?b1 ?e1 ?b2 ?e2
@action
?b = Max2 ?b1 ?b2
?e = Min2 ?e1 ?e2
```

If two non-identical atoms are defined on a property, the above rule signals a problem by assigning the bottom type `owl:Nothing` to the URI in the first place of the tuple. Since $p(x, y, b_1, e_1)$ and $p(x, z, b_2, e_2)$ come with a duration, the type assignment to `?x` only holds for the intersection of the two intervals $[b_1, e_1]$ and $[b_2, e_2]$, computed by `Max2` and `Min2`. In case the intersection is empty, we obtain a triple with duration $[b, e]$, where $e < b$. This “negative” duration indicates that bottom type assignment is not entailed by the premises. This constraint is checked by `IntervalNotEmpty` in the last line of the definition.

Complexity, Soundness, and Completeness

Hayes (2004) and ter Horst (2005) have presented a set of so-called entailment (or inference) rules for RDF/RDFS and a subset of OWL that does not fully cover OWL Lite, but implements parts of OWL DL. Given the original rules, ter Horst has shown that entailment for RDFS is decidable and NP-complete (and even in P if the RDF target graph does not contain any blank nodes). ter Horst has also proved that the incompleteness of the system presented in (Hayes 2004) can be corrected, and that the addition of OWL rules does not change the original complexity results.

The two rule sets for RDFS and OWL have been extended by temporal information, associated with an RDF triple and implemented through additional arguments. These arguments (fourth and fifth position in a quintuple) do *not* “interfere” with the arguments in first, second, and third position. Moreover, the temporal arguments are atoms (integers) which do not have an “internal structure” (unlike URIs) that needs to be considered or that is shared with other tuples in subject, predicate, or object position. By inspecting the extended rules, time can only act in four ways:

1. temporal information in a LHS clause is neither taken into account in other LHS clauses, nor on the RHS; example: variables `?b` and `?e` in rule **rdf1**.
2. temporal information is transported from a LHS clause to a RHS clause; example: variables `?b` and `?e` in rule **rdfs2**.
3. temporal information is compared through the four-place predicate `IntervalNotEmpty`, involving a \leq comparison and the *min* and *max* aggregates; example: `?b1`, `?e1`, `?b2`, and `?e2` in rule **rdfp1a**.
4. temporal information on the RHS is conditioned by the input to the two aggregates `Max2` and `Min2`; example: `?b`

and `?e` in rule **rdfp1b**.

The important point now is that all four rule cases do *not* produce any new individuals (atoms, URIs, or blank nodes). Even the two aggregates only “pick out” one of their input arguments (contrary to `SUM` in SQL, for instance). Thus the proposed extension is still function-free and the additional two arguments do not add a further theoretical complexity. In a triple-based setting (see below), this is no longer the case, since new container objects (blank nodes) need to be generated, bearing the potential of non-termination.

Concerning runtime, the predicate `IntervalNotEmpty`, and the two aggregates `Min2` and `Max2` have a constant complexity, thus the original complexity results of the “untensed” case do hold here as well. The only difference comes from the replacement of the RDF triple by a quintuple (two additional arguments).

As indicated in the beginning, the set of extended rules is *not complete* in that $p(\vec{x}, s', t')$ can not be derived from $p(\vec{x}, s, t)$, assuming $s \leq s' \leq t' \leq t$. If we would allow such rules, the computation of the deductive closure is no longer terminating. Such information, however, can (and should) be obtained through ABox queries.

As rule **rdfp1b** shows, inconsistency is expressed by assigning the bottom type `owl:Nothing` to individuals. In order to make the rule system *sound*, two additional rules must be added, addressing a combination of `owl:sameAs` and `owl:differentFrom`, as well as `owl:disjointWith` together with two `rdf:type` statements.

References

- Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26(11):832–843.
- Dean, T. L., and McDermott, D. V. 1987. Temporal data base management. *Artificial Intelligence* 32:1–55.
- Hayes, P. 2004. RDF semantics. Technical report, W3C. W3C Recommendation, <http://www.w3.org/TR/rdf-mt/>.
- Hobbs, J., and Pan, F. 2004. An ontology of time for the Semantic Web. *ACM Transactions on Asian Language Processing (TALIP)* 3(1):66–85.
- Krieger, H.-U., and Kruijff, G.-J. M. 2011. Combining uncertainty and description logic rule-based reasoning in situation-aware robots. In *AAAI 2011 Spring Symposium on “Logical Formalizations of Commonsense Reasoning”*.
- Krieger, H.-U.; Kiefer, B.; and Declerck, T. 2008. A framework for temporal representation and reasoning in business intelligence applications. In *AAAI 2008 Spring Symposium on AI Meets Business Rules and Process Management*, 59–70. AAAI.
- Lutz, C.; Wolter, F.; and Zakharyashev, M. 2008. Temporal description logics: A survey. In *Proceedings of the 15th International Symposium on Temporal Representation and Reasoning (TIME'08)*, 3–14.
- Snodgrass, R. T. 2000. *Developing Time-Oriented Database Applications in SQL*. San Francisco, CA: Morgan Kaufmann.
- ter Horst, H. J. 2005. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *Journal of Web Semantics* 3:79–115.
- Welty, C., and Fikes, R. 2006. A reusable ontology for fluents in OWL. In *Proceedings of Fourth International Conference on Formal Ontology in Information Systems (FOIS)*, 226–236.