Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research
Report
RR-11-02

A Temporal Extension of the Hayes/ter Horst Entailment Rules and a Detailed Comparison with W3C's N-ary Relations

Hans-Ulrich Krieger

October 2011

A Temporal Extension of the Hayes/ter Horst Entailment Rules and a Detailed Comparison with W3C's N-ary Relations

Hans-Ulrich Krieger

DFKI-RR-11-02

This work and the implementation of *HFC* started in 2007 when I worked for the EU-funded project MUSING (2006–2010; FP6 IST 27097), but has also benefitted from work conducted in the CogX project (cogx.eu; FP7 ICT 215181). Writing up this report is supported by the MONNET project (monnet-project.eu; FP7 ICT 248458).

ISSN 0946-008X

A Temporal Extension of the Hayes/ter Horst Entailment Rules and a Detailed Comparison with W3C's N-ary Relations

Hans-Ulrich Krieger
Language Technology Lab
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 and OWL. The extension requires only some lightweight forms of reasoning and is is realized by adding two further temporal arguments, thus replacing a triple by a quintuple. The approach has been implemented in the forward chaining engine *HFC*. Our decision was motivated by experiences we have gained in former projects that have dealt with the representation of changing information over time in description logic ontologies. In order to verify the superiority of the approach, we compare the quintuple scheme with a semantic-preserving encoding scheme for N-ary relations in RDF triples, as proposed by the Semantic Web Best Practices Group of the W3C. The comparison is carried out on a theoretical as well as a practical level, both in the space and the time domain when computing the deductive closure w.r.t. the triple- and quintuple-based temporal entailment rules.

1 Introduction

Representing temporally-changing information becomes increasingly important for reasoning and query services defined on top of RDF and OWL, for practical applications such as business intelligence in particular, and for the Semantic Web/Web 2.0 in general. Extending binary OWL ABox relation instances or RDF triples with further temporal arguments translates into a massive proliferation of useless "container" objects. Reasoning and querying with such representations is extremely complex, expensive, and error-prone.

In this paper, we critically compare two encoding schemes for temporally-changing information in RDF and OWL. The first one conservatively extends the RDF triple model towards a general flat quintuple representation, whereas the second approach utilizes W3C's N-ary relation proposal in RDF, as suggested by the Semantic Web Best Practices Group [3]. In order to present comparable measurements for the two approaches, we have used the rule-based forward chainer *HFC* (see section 5.1) that we have developed over the last years which is comparable to popular engines, such as Jena or OWLIM. Concerning runtime, our measurements have shown that a general tuple-based approach easily outperforms a triple-based encoding by **several orders of magnitude**, depending on the size of the ABox.

In the next section, we present proposals which are somewhat related to the problem described in this paper. After that, we investigate the memory requirements of the two proposals for simply storing a temporal fact. We then outline our approach by presenting the extended entailment rules for RDFS and the OWL Horst dialect. This section also contains a paragraph where we argue that the theoretical results from [8] do hold for our setting as well. Not only do we come up with an implemented set of entailment rules for our approach, but also with a semantic-preserving set of rules using the N-ary relation proposal of W3C in order to guarantee comparable measurements. We finally present measurements, showing that our approach easily outperforms the tuple-based approach when it comes to the materialization of implicit knowledge during temporal entailment reasoning.

2 Related Approaches

In this section, we relate our approach to already existing frameworks.

2.1 Temporal Databases

Temporal databases started somewhat delayed with the development of relational databases and logic programming. With the development and practical application of SQL, many people realized the need to add temporal information to entries in database tables [7].
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 possible.
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.

2.2 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 [6]. The usual interpretation $\mathcal{I}$ for concepts, roles, and individuals is replaced by a temporal interpretation $\Im$ that extends the denotation by a further temporal argument (usually a natural number), interpreted as a time *point*.
Alternatively, a temporal interpretation $\Im$ can also be defined as an infinite sequence $\langle \Im(i) \rangle_{i \geq 0}$ of non-temporal interpretations $\Im(i)$ (worlds, situations), sharing the same domain. For instance, $(n, \mathsf{john}^{\Im}, \mathsf{mary}^{\Im}) \in \mathsf{marriedWith}^{\Im}$ means that at time n, $(\mathsf{john}, \mathsf{mary})$ is an instance of $\mathsf{marriedWith}$. An important variant of TDLs then extends ABox formulae by adding the standard LTL modal operators. For instance, $\mathbf{F}\mathsf{Scrap}(\mathsf{mycar})$ means that there will be a time n, where my car is scrapped, and for $m \geq n$, $(m, \mathsf{mycar}^{\Im}) \in \mathsf{Scrap}^{\Im}$ 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, and 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.

2.3 Approaches Staying Inside RDF

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

1. use a meta-logical predicate;
2. reify the original relations;
3. wrap range arguments;
4. encode a perdurantist/4D view [10];
5. interpret individuals as time slices [4];

(1.), as used, e.g., in the situation calculus, requires the original relation to be reformulated as a function. However, (1.) is outside the expressive means of OWL, but can at least be encoded in RDF by reifying the atemporal fact using a new individual that is related to its temporal extent through the holds predicate. The proposals (2.)–(5.) have already been implemented in OWL. It is

worth noting that (2.)–(4.) enforce a knowledge engineer to rewrite an ontology, whereas (5.) 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, all these approaches end up in a massive proliferation of useless "container" objects. Reasoning and querying with such representations is extremely complex, expensive, and error-prone.
Approach (3.) has been proposed by the W3C Semantic Web Best Practices Group to equip binary relations with further arguments without leaving the well-known RDF model, thus can clearly be used to add temporal information as a special case.

In the following, we will compare this approach with the one we find more promising by simply adding the temporal extent directly, viz., quintuples. To the best of our knowledge, although (2.)–(5.) were only used to *store* information that changes over time, but nobody has extended the standard Hayes/ter Horst entailment rules to *reason* over time. We will do so for (3.) in order to guarantee that the measurements at the end of our paper are comparable.

3 Memory Considerations

Within this chapter, we will count how many bytes, individuals, and triples/tuples are needed to represent a relational fluent (i.e., a fact whose truth value changes over time), encoded both as a quintuple, as well as a set of triples using W3C's N-ary relation proposal.
In the following, we will restrict ourself to quaternary relations $p \subseteq D \times R \times T \times T$, where T is used to describe the starting and ending point of a fluent. Thus a quaternary diachronic relation instance $p(d, r, s, e)$ encodes a truth value for $p(d, r)$ within interval $[s, e]$.

3.1 Quintuples

A binary relation, such as worksFor between a person p of type Person and a company c of type Company becomes a quaternary relation with further temporal arguments s and e:

$$\mathsf{worksFor}(p, c) \longmapsto \mathsf{worksFor}(p, c, s, e)$$

Unfortunately, OWL and description logic (DL) in general only support unary (classes) and binary relations (properties) in order to guarantee decidability of the usual inference problems. Thus forward chainers (such as OWLIM and Jena) as well as tableaux reasoners (e.g., Racer or Pellet) are unable to handle such descriptions.

The quaternary relation instance is represented as a tuple in *HFC* (see section 5.1) by an extension of the plain N-triple format [1]:

```
p <worksFor> c s e .
```

This tuple consists of 5 elements/arguments and requires (at least) 20 ($= 5 * 4$) bytes, assuming an `int[]` representation with 4 byte integers. Using integer arrays is a common way to represent triples/tuples internally, since the external representation of URIs and XSD atoms needs to be addressed only during input and output. Overall, we obtain 1 object (the integer array) to represent the whole tuple.
This last number is very important, since it is desirable to access information directly in a semantic repository, instead of "fiddling" around with helper structures (container objects) that blow up the memory. In addition, the overall number of elements is equally important, since triple repositories usually build up large index structures to efficiently access all those triples that match a specific element at a certain position in a triple.

3.2 W3C's N-ary Relations

Wrapping the range arguments of a relation instance, i.e., grouping them in a new object, allows us to keep the original relation name, although the approach requires to rewrite the TBox of the original ontology:

$$\begin{array}{l}\mathsf{worksFor}(p, \underline{c, s, e}) \ \longmapsto\ \exists o\,.\,\mathsf{worksFor}(p, o) \wedge \\ \quad \mathsf{type}(o, \mathsf{CompanyTime}) \wedge \mathsf{company}(o, c) \wedge \mathsf{starts}(o, s) \wedge \mathsf{ends}(o, e)\end{array}$$

A new object (o), a new class (CompanyTime), and new "accessors" (company, starts, ends) need to be introduced. W3C suggests this obvious pattern to be used to encode arbitrary *N-ary relations* [3]. Instead of defining a new class for each range type of the original relation, one might alternatively define (as we do) a general class, say RangePlusTime, plus three accessors value, starts, and ends, in order to avoid a reduplication of the original class hierarchy:

```
p <worksFor> o .
o <rdf:type> <nary:RangePlusTime> .
o <nary:value> c .
o <nary:starts> s .
o <nary:ends> e .
```

Overall, 5 triples translate into 15 ($= 5 * 3$) elements or 60 ($= 5 * 12$) bytes. This approach introduces a brand-new individual `o` (a blank node) which turns out to be *problematic*, since it might lead to a *non-terminating closure computation* (cf. section 4.5).

4 Our Approach

As outlined above, we will extend the Hayes-/ter Horst-style entailment rules by a temporal dimension. Thus, in our case, we replace an 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 time period S and as a class at a different time T.

From a commonsense 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 (they do not fall apart later).
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.

4.1 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" (this is, however, not relevant for this paper).

4.2 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* (section 5.1), `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.

4.3 Extended Entailment Rules

In the following, we describe a temporal extension of the entailment rules from [2] and [9]. The rules are written in the concrete syntax of *HFC* so they slightly differ from [2] and [9] (who also use slightly different notations).
Due to space limitations, we are only able to display four extended entailment rules of the fully implemented set of 30 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.

4.3.1 rdf1

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. Note that `?s` and `?e` are *don't-care* variables not needed on the RHS.

```
?x ?p ?y ?s ?e
->
?p <rdf:type> <rdf:Property>
```

4.3.2 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.

```
?x ?p ?y ?s ?e
?p <rdfs:domain> ?dom
->
?x <rdf:type> ?dom ?s ?e
```

Next 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.

4.3.3 rdfp1a and rdfp1b

We have complemented the original rule **rdfp1** 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 ?s1 ?e1
?x ?p ?z ?s2 ?e2
->
?y <owl:sameAs> ?z
@test
IntersectionNotEmpty ?s1 ?e1 ?s2 ?e2
```

The `IntersectionNotEmpty` predicate in the test section (`@test`) guarantees that we only *identify* `?y` and `?z` on the RHS in case the temporal extent of $p(x, y)$ and $p(x, z)$ has a *non-empty intersection*:

```
IntersectionNotEmpty start1 end1 start2 end2 ≡
    local start := max(start1, start2)
    local end := min(end1, end2)
    return (start ≤ 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 (good 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 ?s1 ?e1
?x ?p ?z ?s2 ?e2
->
?x <rdf:type> <owl:Nothing> ?s ?e
@test
?y != ?z
IntersectionNotEmpty ?s1 ?e1 ?s2 ?e2
@action
?s = Max2 ?s1 ?s2
?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 $p(x, y, s_1, e_1)$ and $p(x, z, s_2, e_2)$ come with a duration, the type assignment to `?x` only holds for the intersection of the two intervals $[s_1, e_1]$ and $[s_2, e_2]$, computed by `Max2` and `Min2`.
In case the intersection is empty, we obtain a triple with duration $[s, e]$, where $e < s$. This "negative" duration indicates that bottom type assignment is not entailed by the premises. This constraint is checked by `IntersectionNotEmpty` in the last line of the definition above.

4.4 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 [2] 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 30 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 `?s` and `?e` in rule **rdf1**.

2. temporal information is transported from a LHS clause to a RHS clause; example: variables `?s` and `?e` in rule **rdfs2**.

3. temporal information is compared by the four-place predicate `Intersection-NotEmpty`, involving a $\leq$ comparison and the *min* and *max* aggregates; example: `?s1`, `?e1`, `?s2`, and `?e2` in rule **rdfp1a**.

4. temporal information on the RHS is conditioned by the input to the two aggregates `Max2` and `Min2`; example: `?s` and `?e` in rule **rdfp1b**.

The important point now is that all four rule cases do **not** produce any *new* individuals (neither atoms, URIs, nor 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 (usually blank nodes) need to be generated, bearing the potential of non-termination.

Concerning runtime, the predicate `IntersectionNotEmpty`, and the two aggregates `Min2` and `Max2` have a constant complexity, thus the original complexity results of the non-temporal 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 $p(x, y, s', t')$ can not be derived from $p(x, y, s, t)$, 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.

4.5 Extended Entailment Rules Using Triples

Due to space requirements, we will only depict a single temporal entailment rule, viz., **rdfp1b**, showing the worst case that happens when adding time, using the triple-based N-ary relation encoding:

```
?p <rdf:type> <owl:FunctionalProperty>
?p <rdf:type> <owl:DatatypeProperty>
?x ?p ?blank1
?blank1 <nary:value> ?y
?blank1 <nary:starts> ?start1
?blank1 <nary:ends> ?end1
?x ?p ?blank2
?blank2 <nary:value> ?z
?blank2 <nary:starts> ?start2
?blank2 <nary:ends> ?end2
->
?x <rdf:type> ?new
?new <rdf:type> <nary:RangePlusTime>
?new <nary:value> <owl:Nothing>
?new <nary:starts> ?start
?new <nary:ends> ?end
@test
?y != ?z
?p != <rdf:type>
IntersectionNotEmpty ?start1 ?end1 ?start2 ?end2
@action
?start = Max2 ?start1 ?start2
?end = Min2 ?end1 ?end2
?new = MakeUri <owl:Nothing> ?start ?end
```

This version is much more complex than the quintuple-based encoding shown before. The original range arguments bound to `?y` and `?z` are hidden in two nodes, together with their temporal extent. **rdfp1b** requires **10** LHS clauses to express the equivalent matching conditions (quintuple encoding: **3** clauses). It utilizes **5** RHS clauses for the representation of the entailed relational fluent (quintuple encoding: **1** clause). Finally, and very important, the rule introduce a brand-new individual (a URI) bound to `?new` in an additional RHS action that is deterministically constructed via `MakeUri` from its input arguments `<owl:Nothing>`, `?start`, and `?end`.

It is worth noting that the generation of new individuals, esp., blank nodes, bear the potential of a *non-terminating deductive closure computation*. For this reason, `?new` is not bound to a blank node, but to a URI, whose name does not change, assuming the input arguments to `MakeUri` are the same. In OWLIM, for instance, such a URI generation is not available, although RHS-only variables can be used in rules, always leading to the introduction of (brand new) blank nodes. Now, in case a rule is applied several times to the same input data, several (different) blank nodes will be generated, encoding equivalent data. In the worst case during a fixpoint computation, such blank nodes lead to an explosion of the RDF repository. The `MakeUri` action in the triple-based version of **rdfp1b**, however, guarantees that such cases will not happen here, although we opt for the quintuple-based encoding, as explained above that does not introduce new individuals at all.

5 Measurements

In order to compare the two approaches on a practical level, we need a reasoner that is able to *directly* encode arbitrary n-ary relations. Popular engines, such as RACER, Pellet, Jena or OWLIM which are geared towards binary relations/RDF triples can not be applied here. Furthermore, and very important, practical reasoning with extended relation instances need some lightweight reasoning capabilities (e.g., aggregates such as *min* and *max*) which are only available in Jena. Unfortunately, Jena is *not* able to materialize even drastically-smaller triple-based ontologies than those we have used here, even not for the original non-temporal entailment rules. As already mentioned, the experiments below were performed using *HFC*, a forward chainer we have developed over the last years.

5.1 *HFC*

Usually, bottom-up forward chaining is employed to carry out (all possible) inferences at compile time, so that querying information reduces to an indexing problem at runtime. The process of making implicit information explicit is often called *materialization* or computing the *deductive closure* of a set of ground atoms $\mathcal{A}$ w.r.t. a set R of universally-quantified implications $B \rightarrow H$ (if-then rules). Bottom-up here means that one starts from the ground atoms to which the rules are applied, contrary to top-down approaches which start with a goal (the head H) and potentially hypothesize intermediate goals that can hopefully be satisfied by ground atoms finally (Prolog's strategy). The body and the head of a rule consist of a set of clauses, interpreted *conjunctively*. In *HFC*, clause arguments are either constants or variables.

Closure computation can be characterized as the computation of the *least fixpoint* of a certain *monotonic* function over the complete lattice $\wp(\mathcal{A})$ of the set of all ground atoms $\mathcal{A}$. Forward chaining, as we used it here, can be seen as *model building* over the Herbrand interpretation of a function-free definite program (Horn logic as used in Prolog). In general, model builders are systems that try to construct a finite model for a given theory (usually, a set of first-order formulae).

In order to make forward chaining scalable, *HFC* applies several optimization techniques that are realized as a sequence of filter stages in order to avoid useless RHS instantiations. This comes as a side product of the fact that closure computation is a monotonic operation: new ground atoms are only added, nothing is deleted. Consider, for instance, a rule

$$r = (b_1\, b_2 \rightarrow H)$$

and assume that r is currently applied in iteration n of the closure computation. Due to the monotonicity argument, matching candidates M^n from $\mathcal{A}$ for the LHS variables of rule r in iteration n can be decomposed into those which are brand new at n and those which come from iteration $n-1$:

$$M^n = N \uplus M^{n-1}$$

Since bindings for the variables of individual clauses are actually tables, computing a binding for all LHS variables effectively reduces to a *natural join* $\bowtie$, known from data base theory. Given the distinction *new vs. old* already mentioned, we can compute all possible bindings for $b_1\, b_2$ from the individual bindings, given N and M^{n-1}:

$$M^n(b_1\, b_2) = N(b_1) \bowtie N(b_2) \cup N(b_1) \bowtie M^{n-1}(b_2) \cup M^{n-1}(b_1) \bowtie N(b_2)$$

This optimization massively speeds up forward chaining, since useless bindings, leading to already instantiated tuples, are no longer generated. In our case here, $M^{n-1}(b_1) \bowtie M^{n-1}(b_2)$ is **not** computed anymore, and the set of those bindings are by far the largest in size, when closure generation n increases.

Intermediate results are even memoized in case more than two tables are involved in order to avoid recomputation of already computed results. This techniques not only applies to individual clauses, but also to larger parts, so-called (LHS) clusters. *HFC* has included further optimizations, e.g.,

- bindings are shared over "similar" clause, even between different rules;
- the LHSs of rules are reordered to faster compute matching candidates;
- instances of equivalence relation on the LHS and the RHS of rules (OWL's `owl:sameAs`, `owl:equivalentClass`, and `owl:equivalentProperty`) are efficiently handled through offline rule rewriting and a union-find structure;

- the processing of individual rules can be parallelized at each fixpoint iteration step by specifying the number of processor cores;
- efficient data structures, such as open-address hash tables, integer arrays for tuples, specialized sets with strategy objects to support binding/table projection, etc., are used.

HFC has implemented several extensions that are not available in comparable systems, such as OWLIM, and that are used in the extended entailment rules:

- replacement of triples by more general tuples,
- possibility to add arbitrary tests to the LHS of a rule,
- possibility to add arbitrary actions to the RHS of a rule,
- incorporation of aggregation rules,
- incorporation of metric linear, potentially underspecified calendar time.

HFC efficiently handles ABoxes with millions of facts and provides means to work with thousands of medium-sized ABox in parallel, an important feature that we employ in the forward branching time approach, described in [5]. The memory measurements that we present in the next section are related to *HFC*, running in reasoning mode. To speed up rule execution, large sets of tuples are maintained for each rule that keep the distinction between *old vs. new*, as explained above. When used as a pure storage engine, the memory footprint of *HFC* is much smaller, e.g., less than 26GB for storing and accessing 100,000,000 triples.

5.2 Initial Numbers

The numbers below are computed against the mid-size ontology that backs up the LT-World language portal (see http://www.lt-world.org). The measurements are obtained on a 64bit Intel Core i7 (2.8 GHz), using Java 1.6. The unexpanded ABox consists of 204,959 RDF triples. Fully materialized, 548,132 triples are obtained. When setting up *HFC* with four processor cores (4 entailment rules always run in parallel, if possible), the materialization terminates in 7.7 seconds after 7 iteration steps, taking 716MB main memory.

Since temporal information is missing in the original data set, we *randomly* attach a temporal starting and ending point to every ABox relation instance, using XSD `int` atoms which we let vary between 0 and 1,000. This synthetical data, called Q1.00, is the starting point for the measurements.

From Q1.00, we produced smaller subsets (three quarters, two quarters, one quarter) of the statements, called Q0.75, Q0.50, and Q0.25. Each of the quintuple sets were then transformed into semantic-preserving sets of triples, using the N-ary relation encoding from section 3.2 (T1.00, T0.75, T0.50, T0.25). This is depicted in figure 1.

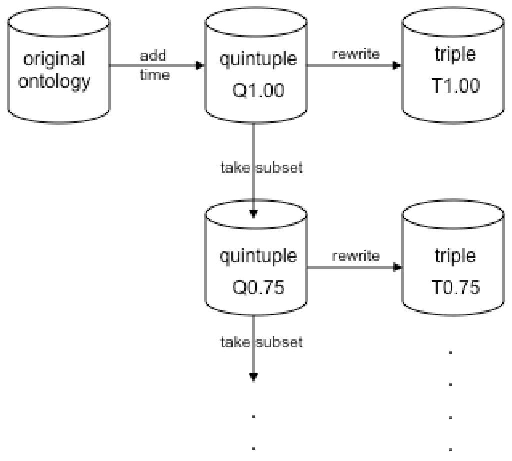

Figure 1: Scheme for constructing the quintuple- and triple-based test sets.

We also generated a second quintuple set Q1.00' from the original data with different starting/ending values. Q1.00 together with Q1.00' gave us a new set Q2.00, doubled in size.
The measurements below use the extended quintuple rules (see section 4.3) to materialize the implicit information contained in the Qx.yy data sets, whereas the triple-based N-ary rules (section 4.5) are applied to the Tx.yy data sets. Without materialization, we obtain the following "offline" numbers:

	#tuples	file size [MB]	load time [sec]
Q2.00	409,920	45.9	3.35
Q1.00	204,960	22.9	1.91
Q0.75	153,720	17.2	1.47
Q0.50	102,480	11.4	1.04
Q0.25	51,240	5.7	0.61
T1.00	1,024,795	50.8	3.89
T0.75	768,600	38.0	3.12
T0.50	512,400	25.1	2.16
T0.25	256,200	12.5	1.22

During materialization, all 30 rules of the extended entailment sets are applied over and over again to the information entailed so far, until a fixpoint is reached, i.e., until no further information is obtained. The differences between the quintuple-based and the triple-based approach are quite drastic as the following "online" numbers show:

	closure [sec]	memory [GB]	#iterations	#tuples
Q2.00	361.3	6.92	23	6,805,359
Q1.00	101.5	3.35	19	2,542,619
Q0.75	4.3	0.62	7	382,110
Q0.50	2.1	0.49	7	151,678
Q0.25	0.9	0.25	3	59,652
T1.00	——[1]	——[2]	——[3]	——[4]
T0.75	236.8	4.44	8	1,844,341
T0.50	26.8	1.49	7	748,532
T0.25	3.1	0.64	3	296,970

The table shows that some of the relational fluents in Q1.00\Q0.75, and so in T1.00\T0.75, lead to a combinatorial explosion during the closure computation from which the triple-based encoding does not recover. Even on a larger machine with 64GB main memory, we were not able to reach a fixpoint for T1.00. Interestingly, the step from Q1.00 to Q2.00 which we have expected to yield a larger combinatorics, was quite easy for the quintuple-based approach.
Clearly, the superiority of the quintuple-based approach not only comes from the smaller set of initial tuples, but also is related to the complexity of the rules from the different entailment rule sets. Both sets consist of 30 rules, but the number of LHS and RHS clauses differ by a factor of 2–3:

	#LHS clauses	#RHS clauses
quintuples	73	32
N-ary relations	143	91

In addition, 15 rules in the triple-based setting generate new individuals when the LHS match is successful (see remark at the end of section 4.5). We finally note here that the dramatic difference between the two approaches carry over to queries that are posted to a triple-/quintuple-based repository.

6 Further Remarks

We hope to have shown that a general tuple-based approach for representing temporally-changing information on the Web is far superior to triple-based approaches. We are convinced that the time now is ripe to move towards this conservative extension of the RDF data model.

It is worth noting that all triple-based approaches presented in section 2.3 are forced to introduce one (or even two) new individuals, usually blank nodes, to encode a temporal extent or other information from the range of an N-ary relation (N > 2). As explained in section 4.5, these new individuals bear the potential

[1]Closure computation stopped after 11 minutes.
[2]15 GB main memory was exceeded then.
[3]Iteration 3 was "nearly" finished.
[4]Approximately 8 million triples were computed so far.

that forward reasoning will not terminate. In case we abandon reasoning at all and only query for explicitly represented information, new individuals that are added clearly do no harm.

However, if inferencing capabilities in a triple-based setting are required, the introduction of blank nodes can often be replaced by the deterministic construction of URI names from the information that is "associated" with them, as section 4.5 has shown. This, however, requires that the reasoning engine provides means to call external functions (such as `MakeUri`).

Contrary to this, a general tuple-based approach, as presented here and implemented in *HFC*, is not plagued by these considerations and is able to directly encode the relation arguments without hiding them in a new object.

References

[1] Grant, J., Beckett, D.: RDF test cases. Tech. rep., W3C (2004), 10 February

[2] Hayes, P.: RDF semantics. Tech. rep., W3C (2004)

[3] Hayes, P., Welty, C.: Defining N-ary relations on the semantic web. Tech. rep., W3C (2006)

[4] Krieger, H.U., Kiefer, B., Declerck, T.: A framework for temporal representation and reasoning in business intelligence applications. In: AAAI 2008 Spring Symposium on *AI Meets Business Rules and Process Management*. pp. 59–70. AAAI (2008)

[5] Krieger, H.U., Kruijff, G.J.M.: Combining uncertainty and description logic rule-based reasoning in situation-aware robots. In: Proceedings of the AAAI 2011 Spring Symposium "Logical Formalizations of Commonsense Reasoning" (2011)

[6] Lutz, C., Wolter, F., Zakharyashev, M.: Temporal description logics: A survey. In: Proceedings of the 15th International Symposium on Temporal Representation and Reasoning (TIME'08). pp. 3–14 (2008)

[7] Snodgrass, R.T.: Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, San Francisco, CA (2000)

[8] ter Horst, H.J.: Combining RDF and part of OWL with rules: Semantics, decidability, complexity. In: Proceedings of the International Semantic Web Conference. pp. 668–684 (2005)

[9] ter Horst, H.J.: Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. Journal of Web Semantics 3, 79–115 (2005)

[10] Welty, C., Fikes, R.: A reusable ontology for fluents in OWL. In: Proceedings of Fourth International Conference on Formal Ontology in Information Systems (FOIS). pp. 226–236 (2006)

-11-02

ch Report