

# **An Ontology of Temporal Concepts for the Semantic Web and Natural Language**

**Ph.D. Dissertation Proposal**

**Feng Pan**

Computer Science Department and Information Sciences Institute  
University of Southern California  
pan@isi.edu

# Contents

<b>1</b>	<b>INTRODUCTION.....</b>	<b>3</b>
1.1	Time for the Semantic Web.....	3
1.2	Time for Natural Language .....	3
1.3	Applications of OWL-Time: the Current Usage .....	5
<b>2</b>	<b>RELATED WORK.....</b>	<b>6</b>
2.1	Ontology of Temporal Concepts .....	6
2.2	Temporal Ontology for Natural Language .....	7
2.3	Other Temporal Representation and Reasoning Formalisms .....	8
<b>3</b>	<b>OWL-TIME BASICS .....</b>	<b>8</b>
3.1	Topological Temporal Relations .....	9
3.2	Linking Time and Events .....	11
3.3	Measuring Durations .....	12
3.3.1	Temporal Units.....	12
3.3.2	Concatenation and <i>Hath</i> .....	13
3.3.3	The Structure of Temporal Units .....	14
3.3.4	Duration Description .....	14
3.4	Clock and Calendar .....	16
3.4.1	Time Zones .....	16
3.4.2	Time Zone Resource in OWL.....	16
3.4.3	Clock and Calendar Units .....	17
3.4.4	Calendar-Clock Description .....	19
<b>4</b>	<b>TEMPORAL AGGREGATES .....</b>	<b>22</b>
4.1	Temporal Aggregates in FOL.....	22
4.1.1	Temporal Sequences.....	22
4.1.2	Temporal Sequences and Their Elements.....	24
4.1.3	<i>Everynthp</i> .....	25
4.2	Embedding iCalendar Recurrence Sets in OWL-Time.....	26
4.2.1	Reify Recurrence Rules .....	26
4.2.2	Map Recurrence Sets .....	27
4.3	Natural Language Examples in FOL .....	29
4.4	Temporal Aggregates in OWL .....	31
4.4.1	Temporal Sequences and Their Members.....	31
4.4.2	Temporal Aggregate Description.....	33
4.5	Two Natural Language Examples in OWL .....	35
<b>5</b>	<b>TEMPORAL ARITHMETIC.....</b>	<b>38</b>
5.1	Desired Properties for Temporal Arithmetic Computation .....	39
5.2	Meaning of “Day Lost (DL)” .....	40
5.3	Temporal Arithmetic Rules .....	40
5.3.1	Adding Durations to Dates .....	41
5.3.2	Subtracting Durations from Dates .....	42
5.3.3	Computing Duration between Two Dates.....	44
<b>6</b>	<b>EXTRACTING TYPICAL EVENT DURATIONS FROM NEWS ARTICLES.....</b>	<b>47</b>
6.1	Annotation Guidelines and Events Classes .....	48
6.1.1	Annotation Instructions .....	48
6.1.2	Analysis .....	49
6.1.3	Event Classes.....	49

6.2	Inter-Annotator Agreement .....	52
6.2.1	What Should Count as Agreement?.....	52
6.2.2	Expected Agreement.....	54
6.2.3	Experiments.....	56
6.3	Towards Learning Coarse-Grained Event Durations.....	58
<b>7</b>	<b>PROPOSED FUTURE WORK .....</b>	<b>59</b>
<b>8</b>	<b>THESIS CONTRIBUTIONS AND TIMELINE .....</b>	<b>60</b>
8.1	Contributions .....	60
8.2	Timeline .....	61
<b>9</b>	<b>REFERENCES.....</b>	<b>61</b>

# 1 Introduction

As an essential dimension of our information space, *time* plays a very important role in every aspect of our lives. A specification of temporal information is necessarily required for a large group of applications, including the Semantic Web, natural language understanding, and so on.

## 1.1 Time for the Semantic Web

The vision of the Semantic Web is to create a “meaningful” environment for programs and software agents to roam from page to page to carry out sophisticated tasks for users (Berners-Lee et al., 2001), and the specification of temporal information is necessarily required for bringing the Semantic Web into reality. For example, some program or software agent does a Web search for its user to find a place to buy a book needed before next Tuesday. In order to decide whether or not to use an online bookstore that promises delivery within five business days, the specification of temporal information is needed to represent durations in its service description (i.e., “delivery within five business days”), represent calendar dates and temporal relations for user’s constraints and preferences (i.e., “needed before next Tuesday”), do temporal arithmetic on calendar dates and durations (i.e., adding five business days to the current date, assuming shipping out immediately), and compare calendar dates to see whether it can arrive before next Tuesday.

In response to this need, such an *ontology of temporal concepts*, OWL-Time (formerly DAML-Time) (Hobbs and Pan, 2004), has been developed for describing the temporal content of Web pages and the temporal properties of Web services (McIlraith et al., 2001). It has been informed by temporal ontologies developed at a number of sites, and is intended to capture the essential features of all of them and make them easily available to a large group of Web developers and users. OWL-Time is represented in both first-order logic (FOL) and the OWL Web Ontology Language (McGuinness and Harmelen, 2003).

The reason for choosing FOL to represent our ontology is that it is a flexible, well-understood, and computationally tractable approach to the representation of knowledge that satisfies many of the requirements raised for a meaning representation language. Specifically, it provides a sound computational basis for the verifiability, inference, and expressiveness requirements (Jurafsky and Martin, 2000).

Endorsed by W3C, the OWL Web Ontology Language is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics (McGuinness and Harmelen, 2003).

## 1.2 Time for Natural Language

Since most of the information on the Web is in natural language, OWL-Time is also an ontology for natural language and can be used for temporal reasoning and to increase the

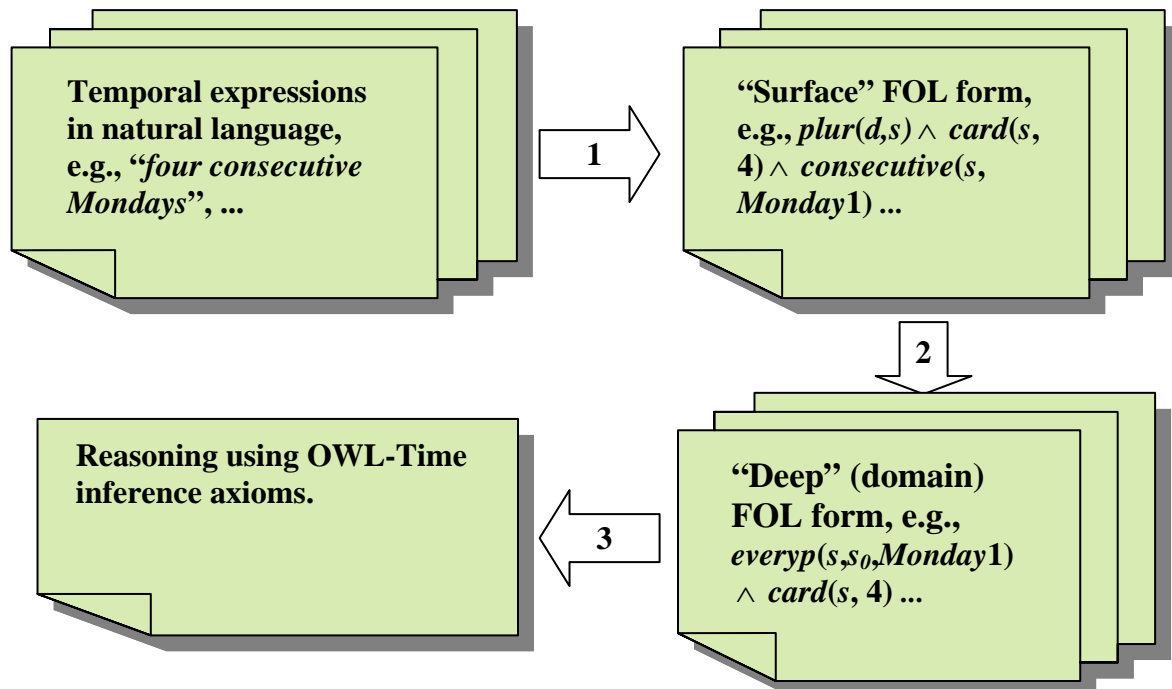


Figure 1.1 Pipeline of Temporal Information Processing and Reasoning in NL Text

temporal awareness for different natural language applications, such as question answering, information retrieval, and summarization. For example, a question answering system may have in its knowledge base the following sentences about Bill Clinton:

*On February 1, 2005, he (Bill Clinton) was picked by UN Secretary-General Kofi Annan to head the United Nations earthquake and tsunami relief and reconstruction effort. Five days later, he and Bush (George H. W. Bush) both appeared on the Super Bowl XXXIX pre-game show on Fox in support of their bipartisan effort to raise money for relief of the disaster through the USA Freedom Corps, an action which Bush described as “transcending politics.” Thirteen days later, they both traveled to the affected areas to see how the relief efforts are going.*

In order to answer questions like “when did Clinton and Bush both travel to affected areas in 2005?”, the specification of temporal information is needed to represent the calendar dates (e.g., “February 1, 2005”) and durations (e.g., “thirteen days”), and do temporal reasoning on the date and the durations to get the answer date by applying temporal arithmetic rules (for this example, simply add 5 days and then 13 days to February 1, and get the answer date of February 19, 2005).

FOL representation and reasoning for natural language has already been applied successfully to question answering (Moldovan et al., 2002) at LCC<sup>1</sup>, where questions and answers are first translated into first-order logical forms. A resolution-based module then proves that the question logically follows from the answer using a set of axioms that are automatically extracted from the WordNet glosses.

<sup>1</sup> <http://www.languagecomputer.com/>

For temporal information processing and reasoning in natural language text, our 3-step pipeline looks a little bit different (see Figure 1.1). Step 1 uses LFToolkit<sup>2</sup> to translate original temporal expressions in natural language to a “surface” FOL form where the predicate names are taken directly from the expressions. Step 2 uses domain mapping rules to map the “surface” FOL form to the “deep” or domain FOL form where the predicate names are from the OWL-Time vocabulary. For example, the mapping rule used in the pipeline example is:

$$(\forall s, p) \textit{consecutive}(s, p) \equiv (\exists s_0) \textit{everyyp}(s, s_0, p)$$

In this pipeline example, a given natural language expression (“*four consecutive Mondays*”) is first translated into its “surface” FOL form ( $\textit{plur}(d,s) \wedge \textit{card}(s, 4) \wedge \textit{consecutive}(s, \textit{Monday}1)$ ). The above mapping rule is then used to translate this “surface” FOL form to the OWL-Time domain FOL form ( $\textit{everyyp}(s, s_0, \textit{Monday}1) \wedge \textit{card}(s, 4)$ )<sup>3</sup>.

Finally, in Step 3 with all the propositions in the OWL-Time vocabulary, inferences can be made using OWL-Time inference axioms by any FOL theorem provers.

Moreover, OWL-Time can also be used to support and interpret useful annotation of temporal information in natural language text (Hobbs and Pustejovsky, 2003), e.g., TimeML (Pustejovsky et al., 2002), to provide a temporal grounding for texts that will facilitate reasoning over events, their orderings, and their relative granularity.

### 1.3 Applications of OWL-Time: the Current Usage

OWL-Time can be applied to a very wide range of applications. Whenever you need a program or software agent to process and/or reason about temporal information, such a time ontology would be needed. In fact, OWL-Time has already been used in applications from all sorts of different fields besides the Semantic Web and natural language, from ubiquitous and pervasive computing, information integration, video event representation, to geographic information science, and so on.

OWL-Time has already been used in/for

- OWL-S, an OWL-based Web service ontology which supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form (OWL-S Coalition, 2004). In the current version (i.e., OWL-S 1.1 release<sup>4</sup>), OWL-Time is used in its upper ontology for services in process.owl<sup>5</sup> and its Congo.com (a fictitious B2C site) example<sup>6</sup> (Pan and Hobbs, 2004).
- Question Answering systems to handle questions that ask about temporal information. OWL-Time is used as one of the resources for identifying expected

<sup>2</sup> <http://www.isi.edu/~nrathod/wne/LFToolkit/>

<sup>3</sup> Please see Section 4.5 for more details on this example.

<sup>4</sup> <http://www.daml.org/services/owl-s/1.1/>

<sup>5</sup> <http://www.daml.org/services/owl-s/1.1/Process.owl>

<sup>6</sup> <http://www.daml.org/services/owl-s/1.1/examples.html>

answer types, as well as supplying rules (axioms) for temporal inferences (Harabagiu and Bejan, 2005).

- Information integration for scientific data on the Grid to map temporal attributes from different sources (Tuchinda et al., 2004).
- Data integration to resolve temporal semantic conflicts between data sources and receivers due to data semantics changes over time, e.g., historical stock price database, (Zhu et al., 2004).
- Improving portlet (i.e., applications within a portal in much the same way as servlets within a Web server) interoperability by defining events for the portal ontology (Diaz et al., 2005).
- Video event representation for the development of a formal language for describing an ontology of video events (Nevatia et al., 2004).
- Ubiquitous and pervasive computing and its use for building a smart meeting room system for its standard ontology SOUPA and the broker-centric agent architecture CoBrA (Chen et al., 2003, 2004).
- Semantic Tuple Spaces (sTuples) to provide temporal concepts for creating ontologies in sTuples, a infrastructure for pervasive computing, where Tuple Space can be viewed as a logically shared memory, where producers add tuples to the space, while consumers read or extract tuples from the space using a search template (Khushraj et al., 2004).
- Geographic Information Science (GIScience) to provide a basis for representing geographic information, e.g., “time regions” and “space-time regions” (Agarwal, 2005).
- Ontology architecture for semantic Geo services for the Olympic Games 2008 in Beijing as its time ontology component (Weissenberg and Gartmann, 2003).

## 2 Related Work

### 2.1 Ontology of Temporal Concepts

There already exist ontologies of temporal concepts from both commercial (e.g., time ontology in Cyc (Lenat, 1995)) and academic sources (e.g., a reusable time ontology (Zhou and Fikes, 2002); time ontology in SUMO (Suggested Upper Merged Ontology) (Niles and Pease, 2001); a catalog of temporal theories (Hayes, 1995)).

Both Cyc and SUMO are large upper ontologies encoding common sense knowledge, and hence more focus on abstract and philosophical concepts that are general enough to address a broad range of domain areas. Each domain (including time) is not

intended to be complete, and they don't aim at Web or natural language applications either. Moreover, OWL-Time is designed to be cleanly separated from other ontologies one might build, where the interface between them will involve the minimum of elements, and the only ontology assumed is arithmetic. As defined in a large upper ontology, the time ontology in Cyc and SUMO, however, is less separated from other domains in the upper ontology (e.g., event ontology), and makes more assumptions about higher level concepts and their properties.

As a commercial institution, Cyc has only released a small part of its entire ontology to the public. For example, for time ontology only the vocabulary is publicly available, not the axioms that define the concepts, their properties and relations which are very important for doing temporal inference and reasoning. The Cyc ontology has not been subject to extensive peer review either.

SUMO is written in SUO-KIF (Pease, 2004) which was derived from KIF (Genesereth, 1992) to support the definition of SUMO. Special translation or a theorem prover would be needed in order to interact with ontologies written in other languages. The time ontology in SUMO is relatively less expressive. For example, it cannot express multiple-layered or conditional temporal aggregates.

(Zhou and Fikes, 2002) proposed a reusable time ontology written in KIF for large-scale knowledge system applications. Our treatments on topological temporal relations are pretty similar, for example, we both treat instant and interval as independent primitives. However, we have different treatment for many other domains (e.g., calendar and clock information and measures of durations), and OWL-Time includes a richer set of axioms and concepts and also broader coverage of the domains (e.g., time zone, temporal aggregates, and temporal arithmetic).

As a survey of several structures that time can be taken to have, a catalog of temporal theories (Hayes, 1994) gives a coherent overview of different theories and synthesizes them as much as possible. Like OWL-Time, what it's concerned with is the actual structure of time, instead of how facts persist through time, or how states of knowledge are sensitive to the changes that time can produce, which is the major concern for the research on temporal logic as described next.

## 2.2 Temporal Ontology for Natural Language

According to Mark Steedman (1997, 2002), in general there are two approaches to temporal ontology for natural language. The first one is the descriptive approach which is usually attributed to Vendler (1967) and has then been refined (Dowty 1979, 1982; Verkuyl, 1989) and further extended (Moens and Steedman, 1988; Smith, 1991). The work in this framework most concerns the descriptive properties of tense and aspect in natural language, for example, how to disambiguate the four aspectual categories (i.e., achievements, activities, accomplishments, and states), which is relevant to our work on extracting typical event durations from news articles (see Section 6 for details).

The logical and computational approach (McDermott, 1982; van Benthem, 1983, 1995; Allen, 1984; Galton, 1984; Allen and Ferguson, 1997) is the other approach that tries to formalize this quite complex ontology. OWL-Time falls into this category with the focus on representing and reasoning about temporal concepts and expressions, instead of tense and aspect in natural language.

Besides FOL, or first-order predicate calculus (FOPC), which is very popular in the logical and computational approach, special temporal operators were used in other temporal logic, for example, tense logic (Prior, 1957), which, as other modal logics, has a richer expressive power in its domain (e.g., time domain), but it needs special theorem proving systems (maybe inefficient/not practical). Most of the tense logics can be translated into FOL, and usually they are translated into FOL for (efficient/practical) theorem proving or automated deduction.

## 2.3 Other Temporal Representation and Reasoning Formalisms

Constraint-based formalisms for temporal reasoning have been developed for representing and reasoning, ranging from less expressive problems (e.g., Simple Temporal Problem (STP) (Dechter et al., 1991)) to more expressive ones (e.g., Disjunctive Temporal Problem (DTP)). Efficient algorithms have been developed for solving STPs and DTPs to get exact solutions (Tsamardinos and Pollack, 2003) or approximate solutions (Moffitt and Pollack, 2005). Planning and scheduling tasks are their most concerned applications.

Although DTP is more expressive than STP and Temporal Constraint Satisfaction Problem (TCSP), it can only represent duration constraints (plus simple relations as before/after/at given time points, which can be converted to their representation), and there are no calendar or clock concepts involved. It cannot represent temporal aggregates or any non-temporal properties either.

TCSP-based framework has also been applied to temporal reasoning for natural language (Han and Lavie, 2004), where not only the durations but also the calendar and granularity are modeled. However, it's not clear whether temporal aggregates can be represented in their framework. A special-purpose reasoner, which is a conventional TCSP solver with the aid of the calendar model, has to be used to draw inferences on given temporal expressions.

A formal framework for the definition and representation of time granularities was proposed in (Bettini et al., 2000) especially for temporal database applications. This framework has also been extended to TCSP (Bettini et al., 2002) for more general temporal reasoning tasks.

## 3 OWL-Time Basics

The material in this section is taken from (Hobbs, 2002; Hobbs and Pan, 2004). The full definitions of OWL-Time can be found in (Hobbs and Pan, 2004). Here we only present those parts that are essential for our treatment of temporal aggregates and temporal arithmetic later, including a vocabulary for expressing facts about topological relations among instants and intervals (Section 3.1), together with information about durations (Section 3.3), and about clock and calendar information (Section 3.4). The abstract characterization of the concepts and relations are expressed in both FOL and OWL. A time zone resource in OWL developed for the entire world will also be described.

In an extension of the time ontology, it is also allowed for temporal predicates to apply directly to events (Pan and Hobbs, 2004), should the user wish, but here we restrict our treatment to temporal entities.

### 3.1 Topological Temporal Relations

There are two subclasses of TemporalEntity: Instant and Interval.

$$(\forall T)[TemporalEntity(T) \equiv Interval(T) \vee Instant(T)]$$

This can be expressed in OWL as:

```
<owl:Class rdf:ID="Instant">
  <rdfs:subClassOf rdf:resource="#TemporalEntity"/>
</owl:Class>

<owl:Class rdf:ID="Interval">
  <rdfs:subClassOf rdf:resource="#TemporalEntity"/>
</owl:Class>

<owl:Class rdf:ID="TemporalEntity">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Instant" />
    <owl:Class rdf:about="#Interval" />
  </owl:unionOf>
</owl:Class>
```

Intervals are, intuitively, things with extent and instants are, intuitively, point-like in that they have no interior points.

The predicates “begins” and “ends” are relations between instants and temporal entities.

$$begins(t,T) \supset Instant(t) \wedge TemporalEntity(T)$$

$$ends(t,T) \supset Instant(t) \wedge TemporalEntity(T)$$

“begins”, for example, can be specified in OWL as:

```
<owl:ObjectProperty rdf:ID="begins">
  <rdfs:domain rdf:resource="#TemporalEntity" />
  <rdfs:range rdf:resource="#Instant" />
</owl:ObjectProperty>
```

The predicate “inside” is a relation between an instant and an interval.

$$inside(t,T) \supset Instant(t) \wedge Interval(T)$$

This concept of “inside” is not intended to include beginnings and ends of intervals.

It will be useful in characterizing clock and calendar terms to have a relation between instants and intervals that says that the instant is inside or the beginning of the interval.

$$(\forall t, T)[\text{beginsOrIn}(t, T) \equiv [\text{begins}(t, T) \vee \text{inside}(t, T)]]$$

A proper interval can be defined as one whose start and end are not identical.

$$\begin{aligned} (\forall T)\text{ProperInterval}(T) \\ \equiv \text{Interval}(T) \wedge (\forall t_1, t_2)[\text{begins}(t_1, T) \wedge \text{ends}(t_2, T) \supset t_1 \neq t_2] \end{aligned}$$

There is a “before” relation on temporal entities, which gives directionality to time. If temporal entity  $T_1$  is before temporal entity  $T_2$ , then the end of  $T_1$  is before the start of  $T_2$ . Thus, “before” can be considered to be basic to instants and derived for intervals.

$$\begin{aligned} (\forall T_1, T_2)[\text{before}(T_1, T_2) \\ \equiv (\exists t_1, t_2)[\text{ends}(t_1, T_1) \wedge \text{begins}(t_2, T_2) \wedge \text{before}(t_1, t_2)]] \end{aligned}$$

The “before” relation is anti-reflexive, anti-symmetric, and transitive.

$$\text{before}(T_1, T_2) \supset T_1 \neq T_2$$

$$\text{before}(T_1, T_2) \supset \neg \text{before}(T_2, T_1)$$

$$\text{before}(T_1, T_2) \wedge \text{before}(T_2, T_3) \supset \text{before}(T_1, T_3)$$

If an instant is inside a proper interval, then the beginning of the interval is before the instant, which is before the end of the interval. This is the principal property of inside.

$$\begin{aligned} \text{inside}(t, T) \wedge \text{begins}(t_1, T) \wedge \text{ends}(t_2, T) \wedge \text{ProperInterval}(T) \\ \supset \text{before}(t_1, t) \wedge \text{before}(t, t_2) \end{aligned}$$

The relation “after” is defined in terms of “before”.

$$\text{after}(T_1, T_2) \equiv \text{before}(T_2, T_1)$$

The relations between intervals defined in Allen’s temporal interval calculus (Allen, 1984; Allen and Ferguson, 1997) can be defined in a straightforward fashion in terms of “before” and identity on the beginning and end points.

OWL-Time includes axioms defining the interval relations “intEquals”, “intBefore”, “intMeets”, “intOverlaps”, “intStarts”, “intDuring”, “intFinishes”, and their reverse interval relations: “intAfter”, “intMetBy”, “intOverlappedBy”, “intStartedBy”, “intContains”, “intFinishedBy”. For example, the definition of “intMeets” is:

$$\begin{aligned} \text{intMeets}(T_1, T_2) \\ \equiv [\text{ProperInterval}(T_1) \wedge \text{ProperInterval}(T_2) \\ \wedge (\exists t)[\text{ends}(t, T_1) \wedge \text{begins}(t, T_2)]] \end{aligned}$$

It will be useful below to have a single predicate for intervals intersecting in at most an instant.

$$\begin{aligned} & \text{nonoverlap}(T_1, T_2) \\ & \equiv [\text{intBefore}(T_1, T_2) \vee \text{intAfter}(T_1, T_2) \vee \text{intMeets}(T_1, T_2) \vee \text{intMetBy}(T_1, T_2)] \end{aligned}$$

This could have been defined as easily in terms of before relations on the beginnings and ends of the intervals.

“intMeets”, for example, can be specified in OWL as:

```
<owl:ObjectProperty rdf:ID="intMeets">
  <rdfs:subPropertyOf rdf:resource="#nonoverlap" />
  <rdfs:domain rdf:resource="#ProperInterval" />
  <rdfs:range rdf:resource="#ProperInterval" />
</owl:ObjectProperty>
```

## 3.2 Linking Time and Events

The time ontology links to other things in the world through four predicates: “atTime”, “during”, “holds”, and “timeSpan”. It is assumed that another ontology provides for the description of events, either a general ontology of event structure abstractly conceived, or specific, domain-dependent ontologies for specific domains.

The predicate “atTime” relates an event to an instant, and is intended to say that the event holds, obtains, or is taking place at that time.

$$\text{atTime}(e, t) \supset \text{Instant}(t)$$

The predicate “during” relates an event to an interval, and is intended to say that the event holds, obtains, or is taking place during that interval.

$$\text{during}(e, T) \supset \text{Interval}(T)$$

Whether a particular process is viewed as instantaneous or as occurring over an interval is a granularity decision that may vary according to the context of use, and is assumed to be provided by the event ontology.

The predicate “timeSpan” relates eventualities to instants or intervals (or temporal sequences of instants and intervals). For contiguous states and processes, it tells the entire instant or interval for which the state or process obtains or takes place.

$$\text{timeSpan}(T, e) \supset \text{TemporalEntity}(T) \vee \text{tseq}(T)^7$$

$$\text{timeSpan}(T, e) \wedge \text{Interval}(T) \supset \text{during}(e, T)$$

$$\text{timeSpan}(t, e) \wedge \text{Instant}(t) \supset \text{atTime}(e, t)$$

<sup>7</sup> *tseq*(*T*): *T* is a temporal sequence. Please see Section 4 for details.

Whether the event obtains at the start and end points of its time span is a matter for the event ontology to specify. The silence here on this issue is the reason “timeSpan” is not defined in terms of necessary and sufficient conditions.

In an extension of the time ontology, it is also allowed for temporal predicates to apply directly to events (Pan and Hobbs, 2004), should the user wish. Thus, “begins( $t, e$ )” says that the instant  $t$  begins the interval that is the time span of event  $e$ .

### 3.3 Measuring Durations

#### 3.3.1 Temporal Units

This development assumes that ordinary arithmetic is available.

There are at least two approaches that can be taken toward measuring intervals. The first is to consider units of time as functions from Intervals to Reals. Owing to infinite intervals, the range must also include Infinity.

$$\text{minutes: Intervals} \rightarrow \text{Reals} \cup \{\text{Infinity}\}$$

$$\text{minutes}([5:14, 5:17]) = 3$$

The other approach is to consider temporal units to constitute a set of entities, call it TemporalUnits, and have a single function *duration* mapping Intervals  $\times$  TemporalUnits into the Reals.

$$\text{duration: Intervals} \times \text{TemporalUnits} \rightarrow \text{Reals} \cup \{\text{Infinity}\}$$

$$\text{duration}([5:14, 5:17], *Minute*) = 3$$

The two approaches are interdefinable:

$$\text{seconds}(T) = \text{duration}(T, *Second*)$$

$$\text{minutes}(T) = \text{duration}(T, *Minute*)$$

$$\text{hours}(T) = \text{duration}(T, *Hour*)$$

$$\text{days}(T) = \text{duration}(T, *Day*)$$

$$\text{weeks}(T) = \text{duration}(T, *Week*)$$

$$\text{months}(T) = \text{duration}(T, *Month*)$$

$$\text{years}(T) = \text{duration}(T, *Year*)$$

Ordinarily, the first is more convenient for stating specific facts about particular units; the second is more convenient for stating general facts about all units.

The arithmetic relations among the various units are as follows:

$$\text{seconds}(T) = 60 * \text{minutes}(T)$$

$$\text{minutes}(T) = 60 * \text{hours}(T)$$

$$\text{hours}(T) = 24 * \text{days}(T)$$

$$\text{days}(T) = 7 * \text{weeks}(T)$$

$$\text{months}(T) = 12 * \text{years}(T)$$

The relation between days and months (and, to a lesser extent, years) are specified as part of the ontology of clock and calendar, below. On their own, however, month and year are legitimate temporal units.

### 3.3.2 Concatenation and *Hath*

The multiplicative relations above don't tell the whole story of the relations among temporal units. Temporal units are *composed of* smaller temporal units. A larger temporal unit is a concatenation of smaller temporal units. A general relation of concatenation between an interval and a set of smaller intervals will be defined first. A predicate *Hath* that specifies the number of smaller unit intervals that concatenate to a larger interval will then be introduced.

*Concatenation*: A proper interval  $x$  is a concatenation of a set  $S$  of proper intervals if and only if  $S$  covers all of  $x$ , and all members of  $S$  are subintervals of  $x$  and are mutually disjoint. (The third conjunct on the right side of  $\equiv$  is because “beginsOrIn” only covers instants that begin or are inside  $x$ .)

$$\begin{aligned} \text{concatenation}(x,S) & \\ & \equiv \text{ProperInterval}(x) \\ & \wedge (\forall z)[\text{beginsOrIn}(z,x) \supset (\exists y)[\text{member}(y,S) \wedge \text{beginsOrIn}(z,y)]] \\ & \wedge (\forall z)[\text{ends}(z,x) \supset (\exists y)[\text{member}(y,S) \wedge \text{ends}(z,y)]] \\ & \wedge (\forall y)[\text{member}(y,S) \\ & \quad \supset [\text{intStarts}(y,x) \vee \text{intDuring}(y,x) \vee \text{intFinishes}(y,x) \\ & \quad \quad \vee \text{intEquals}(y,x)]] \\ & \wedge (\forall y_1, y_2)[\text{member}(y_1, S) \wedge \text{member}(y_2, S) \\ & \quad \supset [y_1 = y_2 \vee \text{nonoverlap}(y_1, y_2)]] \end{aligned}$$

*Hath*: The basic predicate used here for expressing the composition of larger intervals out of smaller clock and calendar intervals is *Hath*, from statements like “30 days hath September” and “60 minutes hath an hour.” Its structure is

$$\text{Hath}(N,u,x)$$

meaning “ $N$  proper intervals of duration one unit  $u$  hath the proper interval  $x$ .” That is, if “ $\text{Hath}(N,u,x)$ ” holds, then  $x$  is the concatenation of  $N$  unit intervals where the unit is  $u$ . For example, if  $x$  is some month of September then “ $\text{Hath}(30, * \text{Day} *, x)$ ” would be true. *Hath* is defined as follows:

$$\begin{aligned} \text{Hath}(N,u,x) & \equiv (\exists S)[\text{card}(S) = N \wedge (\forall z)[\text{member}(z,S) \supset \text{duration}(z,u) = 1] \\ & \quad \wedge \text{concatenation}(x,S)] \end{aligned}$$

That is,  $x$  is the concatenation of a set  $S$  of  $N$  proper intervals of duration one unit  $u$ .

The type constraints on its arguments can be proved as a theorem:  $N$  is an integer (assuming that is the constraint on the value of “card”),  $u$  is a temporal unit, and  $x$  is a proper interval:

$$Hath(N,u,x) \supset integer(N) \wedge TemporalUnit(u) \wedge ProperInterval(x)$$

This treatment of *concatenation* will work for scalar phenomena in general. This treatment of *Hath* will work for measurable quantities in general.

### 3.3.3 The Structure of Temporal Units

It is now possible to define predicates true of intervals that are 1 temporal unit long. For example, *week* is a predicate true of intervals whose duration is one week.

$$second(T) \equiv seconds(T) = 1$$

$$minute(T) \equiv minutes(T) = 1$$

$$hour(T) \equiv hours(T) = 1$$

$$day(T) \equiv days(T) = 1$$

$$week(T) \equiv weeks(T) = 1$$

$$month(T) \equiv months(T) = 1$$

$$year(T) \equiv years(T) = 1$$

It is now in a position to state the relations between successive temporal units.

$$minute(T) \supset Hath(60,*Second*,T)$$

$$hour(T) \supset Hath(60,*Minute*,T)$$

$$day(T) \supset Hath(24,*Hour*,T)$$

$$week(T) \supset Hath(7,*Day*,T)$$

$$year(T) \supset Hath(12,*Month*,T)$$

### 3.3.4 Duration Description

The duration of an interval (or temporal sequence) can have many different descriptions. An interval can be 1 day 2 hours, or 26 hours, or 1560 minutes, and so on. It is useful to be able to talk about these descriptions in a convenient way as independent objects, and to talk about their equivalences. We do this first in terms of a predicate called “durationOf” that takes eight arguments, one for a temporal thing, and one each for years, months, weeks, days, hours, minutes, and seconds. Then we will define a specific kind of individual called a “duration description”, together with a number of relations relating the duration description to the values of each of the eight arguments. Thereby we convert the 8-ary predicate “durationOf” into eight binary relations that are more convenient for description logic-based markup languages, such as OWL. Here is the definition of the duration description, as well as the binary relations like “yearsOf”:

$$(\forall T,y,m,w,d,h,n,s)[durationOf(T,y,m,w,d,h,n,s)$$

$$\begin{aligned} \equiv (\exists D)[&durationDescriptionOf(D,T) \wedge DurationDescription(D) \\ &\wedge yearsOf(D, y) \wedge monthsOf(D, m) \wedge weeksOf(D, w) \\ &\wedge daysOf(D, d) \wedge hoursOf(D, h) \wedge minutesOf(D, n) \\ &\wedge secondsOf(D, s)] \end{aligned}$$

It can be expressed in OWL as:

```
<owl:Class rdf:ID="DurationDescription">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#years" />
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:Restriction>
  </rdfs:subClassOf>

  ...

  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#seconds" />
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

There are two ways to specify the duration description of a temporal thing. The relation “durationDescriptionOf” uses “DurationDescription” as its range, while “durationDescriptionDataType” uses the XML Schema datatype “duration”<sup>8</sup> as its range:

```
<owl:ObjectProperty rdf:ID="durationDescriptionOf">
  <rdfs:domain rdf:resource="#TemporalEntity" />
  <rdfs:range rdf:resource="#DurationDescription" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="durationDescriptionDataType">
  <rdfs:domain rdf:resource="#TemporalEntity" />
  <rdfs:range rdf:resource="&xsd;duration" />
</owl:DatatypeProperty>
```

Using the XML Schema datatype duration is simpler and more standard. But it can’t specify weeks, which can be specified by using DurationDescription. Using DurationDescription also makes it easier to extract values from any field for the later use. The user has the freedom to choose either of these two properties/relations to specify a duration description for a temporal thing.

<sup>8</sup> <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#duration>

## 3.4 Clock and Calendar

### 3.4.1 Time Zones

What hour of the day an instant is in is relative to the time zone. This is also true of minutes, since there are regions in the world, e.g., central Australia, where the hours are not aligned with UTC (Coordinated Universal Time)<sup>9</sup> hours, but are, e.g., offset half an hour. Seconds are not relative to the time zone.

Days, weeks, months and years are also relative to the time zone, since, e.g., 2005 began in the Eastern Standard time zone three hours before it began in the Pacific Standard time zone. Thus, predications about all clock and calendar intervals except seconds are relative to a time zone.

We have been referring to time zones, but in fact it is more convenient to work in terms of what might be called the "time standard" that is used in a time zone. That is, it is better to work with the Pacific Standard Time (PST) as a legal entity than with the PST zone as a geographical region. A time standard is a way of computing the time, relative to a world-wide system of computing time. For each time standard, there is a zone, or geographical region, and a time of the year in which it is used for describing local times. Where and when a time standard is used have to be axiomatized, and this involves interrelating a time ontology and a geographical ontology. These relations can be quite complex. Only the entities like PST and EDT, the time standards, are part of the time ontology.

If we were to conflate time zones (i.e., geographical regions) and time standards, it would likely result in problems in several situations. For example, the Eastern Standard zone and the Eastern Daylight zone are not identical, since most of Indiana until recently was on Eastern Standard time all year. The state of Arizona and the Navajo Indian Reservation, two overlapping geopolitical regions, have different time standards during the daylight saving times -- one is Pacific and the other is Mountain.

Time standards that seem equivalent, like Eastern Standard and Central Daylight, should be thought of as separate entities. Whereas they function the same in the time ontology, they do not function the same in the ontology that articulates time and geography. For example, it would be false to say those parts of Indiana shifted in April from Eastern Standard to Central Daylight time.

### 3.4.2 Time Zone Resource in OWL

A time zone resource<sup>10</sup> in OWL for not only the US but also the entire world has been developed, including three parts: the time ontology file<sup>11</sup>, the US time zone instance file<sup>12</sup>, and the world time zone instance file<sup>13</sup>.

The time zone ontology links a preliminary geographic ontology with a time ontology. It defines the vocabulary about regions, political regions (countries, states,

---

<sup>9</sup> <http://aa.usno.navy.mil/faq/docs/UT.html>

<sup>10</sup> <http://www.isi.edu/~pan/timezonehomepage.html>

<sup>11</sup> <http://www.isi.edu/~pan/damlttime/timezone-ont.owl>

<sup>12</sup> <http://www.isi.edu/~pan/damlttime/timezone-us.owl>

<sup>13</sup> <http://www.isi.edu/~pan/damlttime/timezone-world.owl>

counties, reservations, and cities), time zones, daylight saving policies, and the relationships between these concepts. Its instances also link to other existing data on the Web, such as US states instances developed by Terry Payne<sup>14</sup>, FIPS 55 county instances<sup>15</sup>, and ISO country instances<sup>16</sup>.

It can handle all the usual time zone and daylight savings cases. For example, Los Angeles uses PST, the time offset from UTC is -8 hours, and it observed daylight savings from April 3 to October 30 in 2005. But it handles unusual cases as well. For example, in Idaho the northern part is in the Pacific zone, the southern part in the Mountain. The city of West Wendover, Nevada is in the Mountain time zone, while the rest of Nevada is in the Pacific. For the details, see the documentation<sup>17</sup>, which includes an outline of the ontology and the anticipated use.

### 3.4.3 Clock and Calendar Units

A day as a calendar interval begins at and includes midnight, and goes until, but does not include, the next midnight. This contrasts with a day as a duration which is any interval that is 24 hours in length.

Including the beginning but not the end of a calendar interval in the interval may strike some as arbitrary. But we get a cleaner treatment if, for example, all times of the form 12:xx am, including 12:00 am, are part of the same hour and day, and all times of the form 10:15:xx, including 10:15:00, are part of the same minute. Clock intervals are described with the predicate “clockInt”:

$$\text{clockInt}(y,n,u,x)$$

This expression says that  $y$  is the  $n$ th clock interval of type  $u$  in  $x$ . For example, the proposition “clockInt(10:03,3, \*Minute\*, [10:00,11:00])” holds. Here  $u$  is a member of the set of clock units, that is, one of \*Second\*, \*Minute\*, or \*Hour\*. The larger interval  $x$  may not line up exactly with clock intervals. In this case we take  $y$  to be the  $n$ th complete clock interval of type  $u$  in  $x$ . In addition, there is a calendar unit function with similar structure:

$$\text{calInt}(y,n,u,x)$$

This says that  $y$  is the  $n$ th calendar interval of type  $u$  in  $x$ . For example, the proposition “calInt(12Mar2002,12, \*Day\*, Mar2002)” holds. Here  $u$  is one of the calendar units \*Day\*, \*Week\*, \*Month\*, and \*Year\*.

A distinction is made above between clocks and calendars because they differ in how they number their unit intervals. The first minute of an hour is labeled with 0; for example, the first minute of the hour [10:00,11:00] is 10:00. The first day of a month is labeled 1; the first day of March is March 1. We number minutes for the number just completed; we number days for the day we are working on. Thus, if the larger unit has  $N$  smaller units, the argument  $n$  in “clockInt” runs from 0 to  $N-1$ ; whereas, in “calInt”,  $n$  runs from 1 to  $N$ . To state properties true of both clock and calendar intervals, we can use the predicate “calInt” and relate the two notions with the axiom

---

<sup>14</sup> <http://www.daml.ri.cmu.edu/ont/USRegionState.daml>

<sup>15</sup> <http://www.daml.org/2003/02/fips55/>

<sup>16</sup> <http://www.daml.org/2001/09/countries/iso>

<sup>17</sup> <http://www.isi.edu/~pan/damlttime/time-zone documentation.txt>

$$callInt(y,n,u,x) \equiv clockInt(y,n-1,u,x)$$

In “ $callInt(y,n,u,x)$ ” and “ $clockInt(y,n,u,x)$ ”,  $y$  is not an arbitrary interval; it has to be a calendar-clock interval which is a subclass of a proper interval:

$$CalendarClockInterval(T) \supset ProperInterval(T)$$

The names of months can be defined in terms of the predicate “ $callInt$ ”. For example, July is the seventh month of a year.

$$July(m,y) \equiv callInt(m,7,*Month*,y) \wedge (\exists n,t) [callInt(y,n,*Year*,t)]$$

The top-level time interval (for modern applications) is  $CE(z)$ , which is the Common Era in time zone  $z$ . Thus, the year 2005 in the Eastern Standard Time Zone is the  $y$  such that “ $callInt(y, 2005, *Year*, CE(*EST*))$ ”.

A week is any seven consecutive days. A calendar week, by contrast, according to a commonly adopted convention, starts at midnight, Saturday night, and goes to the next midnight, Saturday night. That is, weeks start with Sunday. (By contrast, the ISO 8061 standard week starts with Monday, and this is also accommodated in the time ontology.) There are 52 weeks in a year, but there are not usually 52 calendar weeks in a year. Weeks are independent of months and years. However, we can still talk about the  $n$ th week in some larger period of time, e.g., the third week of the month or the fifth week of the semester. To say a time interval is a calendar-week, we say

$$callInt(y,n,*Week*,x)$$

As it happens, the  $n$  and  $x$  arguments will often be irrelevant when we only want to say that some period is a calendar week, and not say which.

The day of the week is a calendar interval of type  $*Day*$ . The  $n$ th day-of-the-week in a week is the  $n$ th day in that interval.

$$dayofweek(y,n,x) \equiv callInt(y, n, *Day*,x) \wedge (\exists n_1,x_1) callInt(x,n_1,*Week*, x_1)$$

The days of the week have special names in English, such as Sunday, Monday, and so on. For example, Monday is defined as follows:

$$dayofweek(y,2,x) \equiv Monday(y,x)$$

This says that  $y$  is the Monday of week  $x$ . The ISO 8061 standard week is related to the traditional week as follows:

$$0 < n < 7 \supset [isodayofweek(y,n,x) \equiv dayofweek(y,n+1,x)]$$

$$isodayofweek(y,7,x) \equiv Sunday(y,x)$$

Holidays can also be specified in this ontology. To say that July 4 is a holiday one could write

$$(\forall d,m,y)[callInt(d,4,*Day*,m) \wedge July(m,y) \supset holiday(d)]$$

Holidays like Easter can be defined in terms of this ontology coupled with an ontology of the phases of the moon.

Standard notation for date lists the year, month, day, and time zone. It is useful to define a predication for this.

$$\begin{aligned}
& \text{dateOf}(t,y,m,d,z) \\
& \equiv (\exists d_1,m_1,y_1,e) [\text{beginsOrIn}(t,d_1) \\
& \quad \wedge \text{callInt}(d_1,d,*\text{Day}*,m_1) \wedge \text{callInt}(m_1,m,*\text{Month}*,y_1) \\
& \quad \wedge \text{callInt}(y_1,y,*\text{Year}*,e) \wedge \text{CE}(z) = e]
\end{aligned}$$

Dates of intervals can be defined similarly.

### 3.4.4 Calendar-Clock Description

To express “callInt(y,n,u,x)” and “clockInt(y,n,u,x)” directly in OWL is inconvenient since  $x$  is itself a clock or calendar interval that requires description. So a calendar-clock description is defined in OWL for specifying both calendar and clock information for a calendar-clock interval.

A calendar-clock description has the following properties/fields: unitType, year, month, week, day, dayOfWeek, dayOfYear, hour, minute, second, and time zone. The property “unitType” specifies the temporal unit type of the calendar-clock description, and its domain is TemporalUnit:

```

<owl:Class rdf:ID="TemporalUnit">
  <owl:oneOf rdf:parseType="Collection">
    <TemporalUnit rdf:about="#unitSecond" />
    <TemporalUnit rdf:about="#unitMinute" />
    <TemporalUnit rdf:about="#unitHour" />
    <TemporalUnit rdf:about="#unitDay" />
    <TemporalUnit rdf:about="#unitWeek" />
    <TemporalUnit rdf:about="#unitMonth" />
    <TemporalUnit rdf:about="#unitYear" />
  </owl:oneOf>
</owl:Class>

```

For example, the temporal unit type of 10:30 is minute (unitMinute), and the temporal unit type of March 20, 2003 is day (unitDay). The unit type is required. With a given temporal unit type, all the fields/properties for smaller units will be ignored. For instance, if the temporal unit type is day (unitDay), the values of the field/property hour, minute, and second, if present, will be ignored.

Since calendar-clock description is for describing calendar-clock intervals, a property, called “calendarClockDescriptionOf” with “CalendarClockDescription” as the range, for calendar-clock intervals is defined.

To express “callInt(12Mar2002,12,\*Day\*,Mar2002)”, for example, using calendar-clock description, we need an instance of “CalendarClockDescription” that has values only for unitType (unitDay), year (2002), month (3), and day (12). “clockInt(10:03,3,\*Minute\*,[10:00, 11:00))” can be expressed similarly.

“CalendarClockDescription” and “calendarClockDescriptionOf” are defined in OWL as:

```

<owl:Class rdf:ID="CalendarClockDescription">

```

```

<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#unitType" />
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#year" />
    <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1
  </owl:Restriction>
</rdfs:subClassOf>

...

<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#second" />
    <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#timeZone" />
    <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="calendarClockDescriptionOf">
  <rdfs:domain rdf:resource="#CalendarClockInterval" />
  <rdfs:range rdf:resource="#CalendarClockDescription" />
</owl:ObjectProperty>

```

In order to specify that an instant thing is in a calendar-clock interval, an “inCalendarClock” property/relation is defined similarly to “calendarClockDescriptionOf” as follows:

```

<owl:ObjectProperty rdf:ID="inCalendarClock">
  <rdfs:domain rdf:resource="#Instant" />
  <rdfs:range rdf:resource="#CalendarClockDescription" />
</owl:ObjectProperty>

```

With this “inCalendarClock” relation, we can say that an instant thing is at a specific calendar-clock time. For example, the beginning of a meeting, which is an instant, is at 6:00pm which is actually in a calendar-clock interval of [6:00:00, 6:01:00).

Two simpler relations, “calendarClockDescriptionDatatype” and “inCalendarClockDatatype” are also defined in OWL. Similar to durations, the only difference between these two relations and the above “calendarClockDescriptionOf” and

“inCalendarClock” relations is their ranges: these two simpler relations use the XML Schema datatype “dateTime”<sup>18</sup> as their ranges, while the above uses “CalendarClock-Description”:

```

<owl:DatatypeProperty rdf:ID="inCalendarClockDataType">
  <rdfs:domain rdf:resource="#Instant" />
  <rdfs:range rdf:resource="&xsd;dateTime" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="calendarClockDescriptionDataType">
  <rdfs:domain rdf:resource="#CalendarClockInterval" />
  <rdfs:range rdf:resource="&xsd;dateTime" />
</owl:DatatypeProperty>

```

To illustrate more clearly the difference between using CalendarClockDescription and using the XML datatype dateTime, let’s look at a concrete example: an instant, called "instantExample", at 10:30am EST on 01/01/2005 can be expressed using both inCalendarClockDataType and inCalendarClock in OWL as:

```

<time:Instant rdf:ID="instantExample">
  <time:inCalendarClock rdf:resource="instantExampleDescription" />
  <time:inCalendarClockDataType rdf:datatype="&xsd;dateTime">
    2005-01-01T10:30:00-5:00</time:inCalendarClockDataType>
</time:Instant>

<time:CalendarClockDescription rdf:ID="instantExampleDescription">
  <time:unitType rdf:resource="&time;unitMinute" />
  <time:year rdf:datatype="&xsd;gYear">2005</time:year>
  <time:month rdf:datatype="&xsd;gMonth">1</time:month>
  <time:week rdf:datatype="&xsd;nonNegativeInteger">1</time:week>
  <time:day rdf:datatype="&xsd;gDay">1</time:day>
  <time:dayOfWeekField rdf:datatype="&xsd;nonNegativeInteger">6
</time: dayOfWeekField>
  <time:dayOfYearField rdf:datatype="&xsd;nonNegativeInteger">1
</time: dayOfYearField>
  <time:hour rdf:datatype="&xsd;nonNegativeInteger">10</time:hour>
  <time:minute rdf:datatype="&xsd;nonNegativeInteger">30
</time:minute>
  <time:timeZone rdf:resource="&tz-us;EST" />
</time:CalendarClockDescription>

```

We can see from this example that it’s much simpler to use the XML Schema datatype, “dateTime”. However, the advantage of using “CalendarClockDescription” is that it can express more information than “dateTime”, such as "week", "day of week" and "day of year", so in the above example, we can also know that 01/01/2005 is Saturday, on the first day of the year, and in the first week of the year. The namespace “tz-us” points to our US time zone data<sup>19</sup>. Moreover, each field of “CalendarClockDescription” is separate so that it's easier to extract the value of some fields for the later use and easier to reason about.

<sup>18</sup> <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#dateTime>

<sup>19</sup> <http://www.isi.edu/~pan/damlttime/timezone-us.owl>

## 4 Temporal Aggregates

Temporal aggregates are collections/aggregates of temporal entities. Natural language texts involve many expressions of temporal aggregates, such as “every Tuesday”, “every 3rd Monday in 2001”, “4 consecutive Sundays”, “3 weekdays after today”, “the 4<sup>th</sup> of 6 days of voting”, and so on. Thus it is crucial to have a good ontology of temporal aggregates to represent these expressions.

In Section 4.1 we describe our general approach to temporal aggregates in FOL that is intended to handle any collection of temporal entities, regardless of how they are described. An important special case is temporal aggregates consisting of clock and calendar temporal entities, like calendar months. iCalendar (Dawson and Stenerson, 1998) is a popular framework for describing these, and in Section 4.2 we show how iCalendar can be embedded in OWL-Time. In Section 4.3 we present several examples of natural language expressions for temporal aggregates and how they would be represented in FOL. Some can be described in both iCalendar and OWL-Time, while others can only be represented in OWL-Time. We also demonstrate how we can translate from a natural language sentence to our representation. The temporal aggregates ontology represented in OWL is described in detail in Section 4.4. Two natural language examples will be taken from Section 4.3, and are expressed in OWL in Section 4.5.

### 4.1 Temporal Aggregates in FOL

In this section, the notation of set theory is assumed. Sets and elements of sets will be ordinary individuals, and relations such as "member" will be relations between such individuals. In particular, we will use the relation "member" between an element of a set and the set. We will use the notation " $\{x\}$ " for the singleton set containing the element  $x$ . We will use the function “union” to refer to the union operation between two sets. The function "card" will map a set into its cardinality.

In addition, for convenience, we will make moderate use of second-order formulations, and quantify over predicate symbols. This could be eliminated with the use of an "apply" predicate and axiom schemas systematically relating predicate symbols to corresponding individuals, e.g., the axiom schema for unary predicates  $p$ ,

$$(\forall x)[apply(*p*,x) \equiv p(x)]$$

#### 4.1.1 Temporal Sequences

It will be convenient to have a relation "ibefore" that generalizes over several interval and instant relations, covering both "intBefore" and "intMeets" for proper intervals.

$$\begin{aligned} (\forall T_1, T_2)[ibefore(T_1, T_2) \\ \equiv [before(T_1, T_2) \vee \\ [ProperInterval(T_1) \wedge ProperInterval(T_2) \wedge intMeets(T_1, T_2)]]] \end{aligned}$$

It will also be useful to have a relation "inside" that generalizes over all temporal entities and aggregates. A predicate "inside-1" is defined first that generalizes over instants and intervals and covers "intStarts", "intFinishes" and "intEquals" as well as "intDuring" for intervals. We break the definition into several cases.

$$\begin{aligned}
& (\forall T_1, T_2)[iinside-1(T_1, T_2)] \\
& \equiv [T_1 = T_2 \\
& \quad \vee [Instant(T_1) \wedge ProperInterval(T_2) \wedge inside(T_1, T_2)] \\
& \quad \vee [(\exists t) \text{begins}(t, T_1) \wedge \text{ends}(t, T_1) \\
& \quad \quad \wedge ProperInterval(T_2) \wedge inside(t, T_2)] \\
& \quad \vee [ProperInterval(T_1) \wedge ProperInterval(T_2) \\
& \quad \quad \wedge [intStarts(T_1, T_2) \vee intDuring(T_1, T_2) \\
& \quad \quad \quad \vee intFinishes(T_1, T_2) \vee intEquals(T_1, T_2)]]]]
\end{aligned}$$

The third disjunct in the definition is for the case of 0-length intervals, should they be allowed and distinct from the corresponding instants.

A temporal aggregate is first of all a set of temporal entities, but it has further structure. The relation "ibefore" imposes a natural order on some sets of temporal entities, and the predicate "tseq" is used to describe those sets.

$$\begin{aligned}
& (\forall s)[tseq(s) \equiv (\forall t)[member(t, s) \supset TemporalEntity(t)] \\
& \quad \wedge (\forall t_1, t_2)[member(t_1, s) \wedge member(t_2, s) \\
& \quad \quad \supset [t_1 = t_2 \vee ibefore(t_1, t_2) \vee ibefore(t_2, t_1)]]]
\end{aligned}$$

That is, a temporal sequence is a set of temporal entities totally ordered by the "ibefore" relation. A temporal sequence has no overlapping temporal entities.

It will be useful to have the notion of a temporal sequence whose elements all have a property p.

$$(\forall s, p)[tseqp(s, p) \equiv tseq(s) \wedge (\forall t)[member(t, s) \supset p(t)]]$$

The same temporal aggregate can be broken up into a set of intervals in many different ways.

A minimal temporal sequence is one whose intervals are maximal, so that the number of intervals is minimal. We can view a week as a week or as 7 individual successive days; the first would be minimal. We can go from a non-minimal to a minimal temporal sequence by concatenating intervals that meet.

$$\begin{aligned}
& (\forall s)[min-tseq(s) \\
& \quad \equiv (\forall t_1, t_2)[member(t_1, s) \wedge member(t_2, s) \\
& \quad \quad \supset [t_1 = t_2 \vee (\exists t)[ibefore(t_1, t) \wedge ibefore(t, t_2) \wedge \sim member(t, s)]]]]
\end{aligned}$$

That is, s is a minimal temporal sequence when any two distinct intervals in s have a temporal entity not in s between them.

A temporal sequence  $s_1$  is a minimal equivalent temporal sequence to temporal sequence  $s_2$  if  $s_1$  is minimal and equivalent to  $s_2$ .

$$\begin{aligned}
& (\forall s_1, s_2)[\text{min-equiv-tseq}(s_1, s_2)] \\
& \equiv \text{min-tseq}(s_1) \wedge \text{tseq}(s_2) \\
& \quad \wedge (\forall t, t_2)[\text{TemporalEntity}(t) \wedge \text{iinside-1}(t, t_2) \wedge \text{member}(t_2, s_2)] \\
& \quad \supset (\exists t_1)[\text{member}(t_1, s_1) \wedge \text{iinside}(t, t_1)]]]
\end{aligned}$$

#### 4.1.2 Temporal Sequences and Their Elements

"iinside-1" can now be generalized to the predicate "iinside", which covers both temporal entities and temporal sequences. A temporal entity is "iinside" a temporal sequence if it is "iinside-1" one of the elements of its minimal equivalent temporal sequence.

$$\begin{aligned}
& (\forall t, s)[\text{iinside}(t, s)] \\
& \equiv [\text{TemporalEntity}(t) \wedge \text{TemporalEntity}(s) \wedge \text{iinside-1}(t, s)] \\
& \quad \vee [\text{TemporalEntity}(t) \wedge \text{tseq}(s) \\
& \quad \quad \wedge (\exists s_1, t_1)[\text{min-equiv-tseq}(s_1, s) \wedge \text{member}(t_1, s_1) \\
& \quad \quad \quad \wedge \text{iinside-1}(t, t_1)]]]
\end{aligned}$$

A notion of "isubset" can be defined on the basis of "iinside".

$$(\forall s, s_0)[\text{isubset}(s, s_0) \equiv [\text{tseq}(s) \wedge \text{tseq}(s_0) \wedge (\forall t)[\text{member}(t, s) \supset \text{iinside}(t, s_0)]]]$$

That is, every element of temporal sequence  $s$  is inside some element of the minimal equivalent temporal sequence of  $s_0$ .

A relation of "idisjoint" between two temporal sequences can also be defined.

$$\begin{aligned}
& (\forall s_1, s_2)[\text{idisjoint}(s_1, s_2)] \\
& \equiv [\text{tseq}(s_1) \wedge \text{tseq}(s_2) \\
& \quad \wedge \sim(\exists t, t_1, t_2)[\text{member}(t_1, s_1) \wedge \text{member}(t_2, s_2) \\
& \quad \quad \wedge \text{iinside}(t, t_1) \wedge \text{iinside}(t, t_2)]]]
\end{aligned}$$

That is, temporal sequences  $s_1$  and  $s_2$  are disjoint if there is no overlap between the elements of one and the elements of the other.

The last temporal entity in a temporal sequence is the one with any of the others "ibefore" it.

$$\begin{aligned}
& (\forall t, s)[\text{last}(t, s)] \\
& \equiv [\text{tseq}(s) \wedge \text{member}(t, s) \wedge (\forall t_1)[\text{member}(t_1, s) \supset [t_1 = t \vee \text{ibefore}(t_1, t)]]]
\end{aligned}$$

More generally, the  $n$ th element of temporal sequence can be defined.

$$\begin{aligned}
& (\forall t, s)[\text{nth}(t, n, s)] \\
& \equiv [\text{tseq}(s) \wedge \text{member}(t, s) \wedge \text{natnum}(n)]
\end{aligned}$$

$$\begin{aligned} & \wedge (\exists s_1)[(\forall t_1)[member(t_1, s_1) \\ & \equiv [member(t_1, s) \wedge ibefore(t_1, t)]] \wedge card(s_1) = n-1]] \end{aligned}$$

That is, the  $n$ th element of a temporal sequence has  $n-1$  elements before it.

### 4.1.3 Everynthp

The predicate "ngap" will enable us to define "everynthp" below. Essentially, the idea is a temporal sequence  $s$  containing every  $n$ th element of  $s_0$  for which  $p$  is true. The predicate "ngap" holds between two elements of  $s$  and says that there are  $n-1$  elements between them that are in  $s_0$  and not in  $s$  for which  $p$  is true.

$$\begin{aligned} & (\forall t_1, t_2, s, s_0, p, n) [ngap(t_1, t_2, s, s_0, p, n) \\ & \equiv [member(t_1, s) \wedge member(t_2, s) \wedge tseqp(s, p) \\ & \wedge tseq(s_0) \wedge isubset(s, s_0) \wedge natnum(n) \\ & \wedge (\exists s_1)[card(s_1) = n-1 \wedge idisjoint(s, s_1) \\ & \wedge (\forall t)[member(t, s_1) \\ & \equiv [iinside(t, s_0) \wedge p(t) \wedge ibefore(t_1, t) \wedge ibefore(t, t_2)]]]] \end{aligned}$$

The predicate "everynthp" says that a temporal sequence  $s$  consists of every  $n$ th element of the temporal sequence  $s_0$  for which property  $p$  is true. It will be useful in describing temporal aggregates like "every third Monday in 2001", where  $s$  is the desired temporal sequence ("every third Monday in 2001"),  $s_0$  is the context temporal sequence ("in 2001"),  $n$  is 3 ("third"), and  $p$  is Monday.

$$\begin{aligned} & (\forall s, s_0, p, n) [everynthp(s, s_0, p, n) \\ & \equiv [tseqp(s, p) \wedge tseq(s_0) \wedge natnum(n) \\ & \wedge (\exists t_1)[nth(t_1, 1, s) \wedge \sim(\exists t)[iinside(t, s_0) \wedge ngap(t, t_1, s, s_0, p, n)]] \\ & \wedge (\exists t_2)[last(t_2, s) \wedge \sim(\exists t)[iinside(t, s_0) \wedge ngap(t_2, t, s, s_0, p, n)]] \\ & \wedge (\forall t_1)[last(t_1, s) \vee (\exists t_2) ngap(t_1, t_2, s, s_0, p, n)]] \end{aligned}$$

That is, the first element in  $s$  has no  $p$  element  $n$  elements before it in  $s_0$ , the last element in  $s$  has no  $p$  element  $n$  elements after it, and every element but the last has a  $p$  element  $n$  elements after it.

The variable for the temporal sequence  $s_0$  is, in a sense, a context parameter. When we say "every other Monday", we are unlikely to mean every other Monday in the history of the Universe. The parameter  $s_0$  constrains us to some particular segment of time. (Of course, that segment could in principle be the entire time line.)

The definition of "everyp" is simpler:

$$\begin{aligned} & (\forall s, s_0, p) [everyp(s, s_0, p) \\ & \equiv (\forall t)[member(t, s) \equiv [iinside(t, s_0) \wedge p(t)]] \end{aligned}$$

It is a theorem that every  $p$  is equivalent to every first  $p$ .

$$(\forall s, s_0, p)[\text{everyyp}(s, s_0, p) \equiv \text{everynthp}(s, s_0, p, 1)]$$

"every-other-p" could be defined similarly, but the resulting simplification from "everynthp(s, s<sub>0</sub>, p, 2)" would not be sufficient to justify it.

More examples will be shown in Section 4.3 to illustrate how to use our ontology to represent different kinds of temporal aggregates expressions.

## 4.2 Embedding iCalendar Recurrence Sets in OWL-Time

Internet Calendaring and Scheduling Core Object Specification (iCalendar) is a widely supported standard for personal data interchange. It provides the definition of a common format for openly exchanging calendaring and scheduling information across the Internet. The recurrence set in iCalendar is the complete set of recurrence instances for a calendar component. iCalendar recurrence sets are a subset of OWL-Time temporal sequences.

A systematic way has been developed to map a recurrence set in iCalendar to a temporal sequence in OWL-Time. Here only an illustrative subset of recurrence sets will be demonstrated.

As shown in the next section with natural language examples, there are cases that can be expressed in OWL-Time more accurately than in iCalendar, and there are also cases that iCalendar can't express, but OWL-Time can. Moreover, embedding recurrence sets in OWL-Time gives access to the full ontology of time for temporal reasoning.

### 4.2.1 Reify Recurrence Rules

In iCalendar, a recurrence set  $s$  is defined by a recurrence rule  $r$  which can be expressed as  $\text{RecurrenceSetRule}(s, r)$ . First, we list the properties of a recurrence rule  $r$ :

$$(\forall r) [\text{rule}(r) \supset (\exists tu, g) [\text{freq}(r) = tu \wedge \text{TemporalUnit}(tu) \wedge \text{gap}(g, r)]]$$

"freq(r)" corresponds to the **FREQ** property of recurrence rules. Since it is a required property, it is defined as a function mapping from a recurrence rule to a temporal unit,  $tu$ . For example, **FREQ = WEEKLY** will have a return value of temporal unit **\*Week\***.

"gap(g,r)" corresponds to the **INTERVAL** property of recurrence rules. For example, "every 3<sup>rd</sup> month" will have an **INTERVAL** property with a value of 3. If it is missing, we assume it is given a default value of 1 during the translation process from iCalendar to OWL-Time.

It is required in iCalendar that either **UNTIL** or **COUNT** property may appear in a recurrence rule, but they must not occur in the same rule.

$$(\forall r) [\text{rule}(r) \supset (\exists n) [\text{count}(n, r) \wedge \text{natnum}(n)] \vee (\exists t, y, mo, d, h, mi, s, z) [\text{until}(t, r) \wedge \text{timeOf}(t, y, mo, d, h, mi, s, z)]]$$

"count(n,r)" corresponds to the **COUNT** property of recurrence rules. "until(t,r)" corresponds to the **UNTIL** property of recurrence rules.

Then, we specify the optional properties of a recurrence rule as in:

$$(\forall r, ls) [\text{rule}(r) \wedge \text{bysecond}(ls, r) \supset (\forall s) [\text{member}(s, ls) \supset \text{integer}(s) \wedge 0 \leq s \leq 59]]$$

This corresponds to the BYSECOND property of recurrence rules. BYMINUTE, BYHOUR, and so on are defined similarly.

### 4.2.2 Map Recurrence Sets

Now we can map from the recurrence set generated by a recurrence rule  $r$  with either COUNT or UNTIL to a temporal sequence  $s$ :

$$\begin{aligned} & (\forall r, n, s, g) [RecurrenceSetRule(s, r) \wedge count(n, r) \wedge gap(g, r) \\ & \quad \supset (\exists s_0) [card(s) = n \wedge everynthp(s, s_0, map2p(freq(r)), g)]] \end{aligned}$$

$$\begin{aligned} & (\forall r, t, s, g) [RecurrenceSetRule(s, r) \wedge until(t, r) \wedge gap(g, r) \\ & \quad \supset (\exists t', s_0) [last(t', s_0) \wedge ends(t, t') \wedge everynthp(s, s_0, map2p(freq(r)), g)]] \end{aligned}$$

“map2p” is a function that maps from temporal units to their corresponding unary predicate names. For example,

$$map2p(*year*) = year1, \text{ where } (\forall y) [year1(y) \equiv (\exists n, x) [calInt(y, n, *Year*, x)]]$$

iCalendar can express very complicated recurrence sets. For example, “The 1<sup>st</sup> and 2<sup>nd</sup> hours of the 4<sup>th</sup> and 5<sup>th</sup> months of every year” will have a recurrence rule in which BYHOUR = 1, 2, BYMONTH = 4, 5, and FREQ = YEARLY.

To formalize this we need recursion through the sequence of temporal units (with the predicate “everythtempunit”) and recursion through the list of integers associated with each temporal unit (with the predicate “byTulistRecurs”). We do the latter first.

$$\begin{aligned} & (\forall s, ls, r, tu, g) [RecurrenceSetRule(s, r) \wedge bytempunit(ls, r, tu) \wedge gap(g, r) \\ & \quad \supset (\exists s', s_0) [everynthp(s', s_0, map2p(freq(r)), g) \\ & \quad \quad \wedge byTulistRecurs(s, ls, s', tu, freq(r))]] \end{aligned}$$

$$\begin{aligned} & (\forall s, ls, s', tu, tu') [byTulistRecurs(s, ls, s', tu, tu') \wedge card(ls) > 1 \\ & \quad \supset (\exists s_1, ls_1, i, s_i, i) [ls = union(\{i\}, ls_1) \wedge \sim member(i, ls_1) \\ & \quad \quad \wedge everythtempunit(s_i, s', i, tu, tu') \wedge s = union(s_1, s_i) \\ & \quad \quad \wedge byTulistRecurs(s_1, ls_1, s', tu, tu')]] \end{aligned}$$

Base case of the recursion:

$$\begin{aligned} & (\forall s, s', i, tu, tu') [byTulistRecurs(s, \{i\}, s', tu, tu') \\ & \quad \supset everythtempunit(s, s', i, tu, tu')] \end{aligned}$$

“bytempunit” is a generalized relation from “bysecond”, “byminute”, and so on. It is a relation among a recurrence rule  $r$ , a temporal unit  $tu$ , and a list of values associated with BYxxx with that temporal unit. For example,

$$(\forall ls, r) [bytempunit(ls, r, *Second*) \equiv bysecond(ls, r)]$$

“byTulistRecurs” is a relation among two temporal sequences  $(s, s')$ , their temporal units  $(tu, tu')$ , and a list of values associated with  $tu$ .

“everyithtempunit” is an important relation among two temporal sequences ( $s, s'$ ), their temporal units ( $tu, tu'$ ), and an integer value  $i$ . For example, “everyithtempunit( $s, s', 2, *Month*, *Year*$ )” specifies that the members of temporal sequence  $s$  are every 2<sup>nd</sup> month (i.e., February) of the members of temporal sequence  $s'$  (with temporal unit of  $*Year*$ ); in English, it means “every other February” with an implicit context, i.e., a sequence of years.

It’s possible that the temporal units of  $FREQ$  and  $BYxxx$  are not successive. For example, “the 1<sup>st</sup> two hours in every month”, which can be expressed in iCalendar with  $FREQ = MONTHLY, BYHOUR = 1, 2$ . The temporal units of  $FREQ (*Month*)$  and  $BYHOUR (*Hour*)$  are not successive, and what the example actually says is “the 1<sup>st</sup> two hours of the 1<sup>st</sup> day of every month”. Our axiom needs to be able to handle this case, as well as the normal case where the temporal units are successive. The predicate “everyithtempunit” is defined recursively:

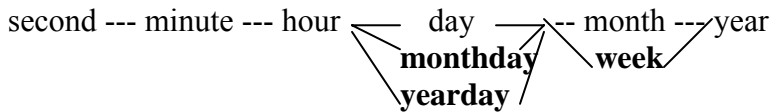
$$\begin{aligned}
& (\forall s_1, s_2, i, tu_1, tu_2) [everyithtempunit(s_1, s_2, i, tu_1, tu_2) \wedge tulevel(tu_2) > tulevel(tu_1) + 1 \\
& \quad \wedge tu_1 \neq *Monthday* \wedge tu_1 \neq *Yearday* \wedge tu_1 \neq *Week* \\
& \quad \supset (\exists s') [everyithtempunit(s_1, s', i, tu_1, level2tu(tulevel(tu_2) - 1)) \\
& \quad \quad \wedge everyithtempunit(s', s_2, 1, level2tu(tulevel(tu_2) - 1), tu_2)]
\end{aligned}$$

Base case:

$$\begin{aligned}
& (\forall s_1, s_2, i, tu_1, tu_2) [everyithtempunit(s_1, s_2, i, tu_1, tu_2) \wedge tulevel(tu_2) = tulevel(tu_1) + 1 \\
& \quad \wedge tu_1 \neq *Monthday* \wedge tu_1 \neq *Yearday* \wedge tu_1 \neq *Week* \\
& \quad \supset (\forall t_1) [member(t_1, s_1) \equiv (\exists t_2) [member(t_2, s_2) \wedge iinside-1(t_1, t_2) \\
& \quad \quad \wedge calclockInt(t_1, i, tu_1, t_2)]]]
\end{aligned}$$

where  $calclockInt(y, n, u, x) \equiv calInt(y, n, u, x) \vee clockInt(y, n, u, x)$

“tulevel” is a function mapping from a temporal unit to its level value according to the hierarchy below. For example, “tulevel( $*second*$ ) = 1, tulevel( $*month*$ ) = 5”. “level2tu” is an inverse function of “tulevel”. It maps from the level value to the associated temporal unit. For example, “level2tu(1) =  $*second*$ , level2tu(5) =  $*month*$ ”. This hierarchy is consistent with the temporal unit ordering in iCalendar.



The temporal units in bold, i.e.,  $*Monthday*$ ,  $*Yearday*$ , and  $*Week*$ , need to be axiomatized sepecially.

- When  $*Monthday*$  and  $*Month*$  are together,:

$$\begin{aligned}
& everyithtempunit(s_1, s_2, i, *Monthday*, *Month*) \\
& \quad \equiv [member(t_1, s_1) \wedge member(t_2, s_2)]
\end{aligned}$$

$$\equiv [iinside-1(t_1, t_2) \wedge calclockInt(t_1, i, tu_1, t_2)]$$

- When \*Yearday\* and \*Year\* are together, force them to the base case, i.e., skip \*Month\*:

$$\begin{aligned} &everyithtempunit(s_1, s_2, i, *Yearday*, *Year*) \\ &\equiv [member(t_1, s_1) \wedge member(t_2, s_2)] \\ &\equiv [iinside-1(t_1, t_2) \wedge calclockInt(t_1, i, tu_1, t_2)] \end{aligned}$$

- When \*Week\* and \*Year\* are together:

$$\begin{aligned} &everyithtempunit(s_1, s_2, i, *Week*, *Year*) \\ &\equiv [member(t_1, s_1) \wedge member(t_2, s_2)] \\ &\equiv [iinside-1(t_1, t_2) \wedge calclockInt(t_1, i, tu_1, t_2)] \end{aligned}$$

iCalendar allows one to specify lists of dates/times as well, with the attributes RDATE and EXDATE. These are translated into simple temporal sequences. A recurrence set is generated by generating recurrence sets specified by the RRULE and/or RDATE attributes and by the EXRULE and/or EXDATE attributes, subtracting the latter from the former, and constraining the result by the DTSTART attribute.

### 4.3 Natural Language Examples in FOL

In the previous section, we've shown how iCalendar statements can be systematically mapped to OWL-Time predicates; in this section we demonstrate how these predicates in OWL-Time can be used to express natural language expressions.

We will first take an example from iCalendar, and show how to express it in OWL-Time. Then we will show an example that OWL-Time can do better than iCalendar. After showing an example that iCalendar can't handle, but OWL-Time can, we will finally show more natural language expressions that can be represented using OWL-Time predicates.

1) “*Every other week on Monday, Wednesday and Friday until December 24, 1997, but starting on Tuesday, September 2, 1997.*” (Taken from RFC 2445 page 120.)

```
DTSTART;TZID=US-Eastern:19970902T090000
RRULE:FREQ=WEEKLY;INTERVAL=2;UNTIL=19971224T000000Z;
WKST=SU;BYDAY=MO,WE,FR
```

iCalendar uses “FREQ=WEEKLY;INTERVAL=2” to represent “every other week”, and uses “BYDAY=MO, WE,FR” to get “every Monday, Wednesday, and Friday”. WKST specifies the start of the week, which is Sunday in this example.

This example can be expressed in OWL-Time as a temporal sequence  $s$  for which the following is true:

$$(\exists s, s', T, t_1, t_2) [everynthp(s', \{T\}, Week1, 2) \wedge$$

$$byTulistRekurs(s, \{1, 3, 5\}, s', *Day*, *Week*) \wedge begins(t_1, T) \wedge ends(t_2, T)$$

$$\wedge dateOf(t_1, 1997, 9, 2) \wedge dateOf(t_2, 1997, 12, 24)]$$

where  $(\forall w) [Week1(w) \equiv (\exists n, x) [callInt(w, n, *Week*, x)]]$

$s$  is the desired temporal sequence representing “every Monday, Wednesday, and Friday” of  $s'$  which is a temporal sequence of “every other week from 09/02/1997 to 12/24/1997”.

2) “Every 3<sup>rd</sup> Monday in 2001.”

This looks like a simple example, but iCalendar can't express it exactly. Though iCalendar can express “the 3<sup>rd</sup> Monday” using BYDAY=3MO, it can't express “every 3<sup>rd</sup> Monday”. In order to express this phrase in iCalendar, we have to paraphrase it first to: “Every 3<sup>rd</sup> week on Monday in 2001”.

However, the paraphrased one is not exactly the same as the original, since it depends on what the first week is. In iCalendar, the first week is defined as the week that “contains at least four days in that calendar year”. Based on this definition, however, if the first week starts from a Tuesday, the first 3<sup>rd</sup> Monday will be different from the first Monday of the 3<sup>rd</sup> week!

In order to express year 2001, in iCalendar it has to specify the start date (01/01/2001) using DTSTART and the end date (12/31/2001) using UNTIL. Here is how to express “Every 3<sup>rd</sup> week on Monday in 2001” in iCalendar:

```
DTSTART;TZID=US-Eastern:20010101T000000
RRULE:FREQ=WEEKLY;INTERVAL=3;UNTIL=20011231T000000Z;
WKST=SU;BYDAY=MO
```

In OWL-Time, this example can be expressed very straightforwardly as a temporal sequence  $s$  for which the following is true:

$$(\exists y, z) [yr(y, 2001, CE(z)) \wedge everynthp(s, \{y\}, Monday1, 3)]$$

where  $(\forall d) [Monday1(d) \equiv (\exists w) [Monday(d, w)]]$

3) “Every Monday that's a holiday.”

There is no way in iCalendar to express “conditional temporal aggregates” or any temporal aggregates that are not in a form of standard temporal units, such as “holidays”, “voting dates”, “days with classes”, “months starting with a Monday”, and so on.

OWL-Time, however, can express any kind of temporal aggregates, using the argument  $p$  in “everynthp( $s, s_0, p, n$ )” or “everyp( $s, s_0, p$ )”. All the conditions and non-temporal-unit concepts can be captured using this  $p$ .

For example, we can express “Every Monday that's a holiday” in OWL-Time as a temporal sequence  $s$  for which the following is true:

$$(\exists s, s_0) [everyp(s, s_0, HolidayMonday)]$$

where  $(\forall d) [HolidayMonday(d) \equiv (\exists w) [Monday(d, w)] \wedge holiday(d)]$

4) More natural language expressions represented in OWL-Time are as follows, where  $s$  is the set corresponding to the noun phrase:

- “*The past three Mondays.*”

$$(\exists s, s_0, t) [everyp(s, s_0, MondayI) \wedge card(s) = 3 \wedge last(t, s_0) \wedge ends(nowfn(D), t)]$$

In our treatment of temporal deictics, the function “nowfn” maps a document  $d$  into the instant or interval viewed as “now” from the point of view of that document, and  $D$  is the document this phrase occurs in.

- “*Every other Monday in every 4<sup>th</sup> month in every year.*”

$$(\exists s, s_1, s_2, s_0) [everyp(s_1, s_0, YearI) \wedge everynthp(s_2, s_1, MonthI, 4) \\ \wedge everynthp(s, s_2, MondayI, 2)]$$

where  $(\forall y) [YearI(y) \equiv (\exists n, x) [calInt(y, n, *Year*, x)]]$

$$(\forall m) [MonthI(m) \equiv (\exists n, x) [calInt(m, n, *Month*, x)]]$$

- “*Four consecutive Mondays.*”

$$(\exists s, s_0) [everyp(s, s_0, MondayI) \wedge card(s) = 4]$$

In order to translate from natural language sentences to our representation, we first run a semantic parser to get a “surface” logical form from a given sentence. Then we use some additional rules to map from the “surface” logical form to our domain ontology. For example, given the above input sentence, the “surface” logical form would be:

$$(\exists s, d) [plur(d, s) \wedge MondayI(d) \wedge consecutive(s, MondayI) \wedge card(s) = 4]$$

“plur” takes as its arguments both a set and a representative member of the set. “consecutive” takes as its arguments both a set and the property of the set, meaning that the members of the set are not only consecutive but also share a certain property. “card” is a function that returns the cardinality/size of the set.

Then we map the above “surface” logical form to our ontology predicates by applying the following rule:

$$(\forall s, p) consecutive(s, p) \equiv (\exists s_0) everyp(s, s_0, p)$$

## 4.4 Temporal Aggregates in OWL

### 4.4.1 Temporal Sequences and Their Members

In order to encode the temporal aggregates ontology in OWL, we first defined temporal sequence. It has only one optional property `hasMemeber` which maps from a temporal

sequence to any temporal thing. A temporal sequence can have no (empty sequence) or many members:

```
<owl:Class rdf:ID="TemporalSeq">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMember" />
      <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">0
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasMember">
  <rdfs:domain rdf:resource="#TemporalSeq" />
  <rdfs:range rdf:resource="#TemporalThing" />
</owl:ObjectProperty>
```

Since we also want to have a backward link pointing from the temporal sequence member to its associated sequence, a `TemporalSeqMember` class is defined. It's a subclass of `Temporal Thing`, and has a required pair of properties: `isMemberOf` and `hasPosition`, so that it can not only point back to the associated sequence but also locate itself in the sequence:

```
<owl:Class rdf:ID="TemporalSeqMember">
  <rdfs:subClassOf rdf:resource="#TemporalThing"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isMemberOf" />
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasPosition" />
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="isMemberOf">
  <rdfs:domain rdf:resource="#TemporalSeqMember" />
  <rdfs:range rdf:resource="#TemporalSeq" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="hasPosition">
  <rdfs:range rdf:resource="&xsd;integer" />
</owl:DatatypeProperty>
```

Since `hasPosition` is also used for other classes, as will see later, it only has a range of integers.

For a temporal sequence member that is associated with multiple sequences, multiple instances of `TemporalSeqMember` must be defined. The reason for defining it in this way is that for a given temporal sequence member instance, it will only have one pair of `isMemberOf` and `hasPosition` values, so that it's not confusing which `hasPosition` value should be paired with which `isMemberOf` value. Moreover, different temporal sequences may apply different attributes to their members.

#### 4.4.2 Temporal Aggregate Description

The most important class in the OWL encodings of the temporal aggregates ontology is the temporal aggregate description class. Analogous to the calendar-clock description, it specifies the temporal aggregate description for temporal sequences, and it's associated with the temporal sequence class by `hasTemporalAggregate-Description` property.

The temporal aggregate description has the following fields/properties: `hasStart`, `hasEnd`, `hasContext-TemporalSeq`, `hasithTemporalUnit`, `hasTemporalUnit`, `hasContextTemporalUnit`, `hasPosition`, `hasGap`, and `hasCount`:

```
<owl:Class rdf:ID="TemporalAggregateDescription">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasStart" />
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasCount" />
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasTemporalAggregateDescription">
  <rdfs:domain rdf:resource="#TemporalSeq" />
  <rdfs:range rdf:resource="#TemporalAggregateDescription" />
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="hasStart">
  <rdfs:domain rdf:resource="#TemporalAggregateDescription" />
  <rdfs:range rdf:resource="#InstantThing" />
</owl:ObjectProperty>
```

The optional properties `hasStart` and `hasEnd` map from the temporal aggregate description to the instant thing, specifying the start and the end instants of a temporal sequence. The calendar and clock properties described in Section 3 can then be used to specify the start and the end times or dates the instants are in.

```
<owl:ObjectProperty rdf:ID="hasContextTemporalSeq">
  <rdfs:domain rdf:resource="#TemporalAggregateDescription" />
  <rdfs:range rdf:resource="#TemporalSeq" />
```

```
</owl:ObjectProperty>
```

The optional property `hasContextTemporalSeq` maps from the temporal aggregate description to the temporal sequence, specifying the context (super) temporal sequence of a given (sub) temporal sequence.

It corresponds to  $s_0$  in `everynthp(s,s0,p,n)` and  $s'$  in `byTulistRekurs(s,ls,s',tu,tu')`. When it's not present, context-free temporal aggregates (e.g., "every Monday") can be represented.

```
<owl:DatatypeProperty rdf:ID="hasithTemporalUnit">
  <rdfs:domain rdf:resource="#TemporalAggregateDescription" />
  <rdfs:range rdf:resource="xsd:positiveInteger" />
</owl:DatatypeProperty>
```

The required property `hasithTemporalUnit` maps from the temporal aggregate description to positive integers, specifying the  $i$ th temporal unit elements in the temporal sequence.

It corresponds to  $ls$  in `byTulistRekurs(s,ls,s', tu,tu')`. Thus it's very possible to have many such property values for a given temporal sequence. For example, "every 3<sup>rd</sup> Monday, Tuesday, and Friday" (such examples will be illustrated in detail in the next section).

```
<owl:ObjectProperty rdf:ID="hasTemporalUnit">
  <rdfs:domain rdf:resource="#TemporalAggregateDescription" />
  <rdfs:range rdf:resource="#TemporalUnit" />
</owl:ObjectProperty>
</owl:ObjectProperty>
```

The required properties `hasTemporalUnit` and the optional `hasContextTemporalUnit` map from the temporal aggregate description to the temporal unit, as defined in Section 3.1. They specify the temporal unit of the given temporal sequence and the context temporal sequence respectively. They correspond to  $tu$  and  $tu'$  in `byTulistRekurs(s,ls,s',tu,tu')`.

The context temporal unit is associated with the context temporal sequence property. Thus if the context temporal sequence is not present, so is the context temporal unit, but not vice versa, since it's possible that the temporal unit of the context temporal sequence is unknown or not relevant.

```
<owl:DatatypeProperty rdf:ID="hasPosition">
  <rdfs:range rdf:resource="xsd:integer" />
</owl:DatatypeProperty>
```

The optional property `hasPosition` is a shared property with the `TemporalSeqMember` class. It specifies the position of the element in the temporal sequence. For example, "the first two Tuesdays in every May" would have `hasPosition` value of 2. It's also possible to have negative positions. For example, "the last Thursday in every November" would have `hasPosition` value of -1.

If this property value is not present, all the positions will be included in the temporal sequence. For example, "the Thursdays in every November" includes all the

Thursdays in every November, while “the last Thursday in every November” only includes the last Thursday in every November.

```
<owl:DatatypeProperty rdf:ID="hasGap">
  <rdfs:domain rdf:resource="#TemporalAggregateDescription" />
  <rdfs:range rdf:resource="&xsd;positiveInteger" />
</owl:DatatypeProperty>
```

The optional property `hasGap` maps from the temporal aggregate description to positive integers, specifying the gap between the elements in the temporal sequence. If it's not present, the default value of 1 will be used, for example, as in “every Monday”.

It corresponds to  $n$  in  $\text{everynthp}(s, s_0, p, n)$ . For example, “every 3<sup>rd</sup> Monday” would have `hasGap` value of 3.

```
<owl:DatatypeProperty rdf:ID="hasCount">
  <rdfs:domain rdf:resource="#TemporalAggregateDescription" />
  <rdfs:range rdf:resource="&xsd;positiveInteger" />
</owl:DatatypeProperty>
```

The optional property `hasCount` maps from the temporal aggregate description to positive integers, specifying the cardinality or the size of the temporal sequence. For example, “four consecutive Sundays” would have `hasCount` value of 4.

## 4.5 Two Natural Language Examples in OWL

In this section, we will take two natural language examples from Section 4.3, and express them in OWL. One example is a complex multiple-layered temporal aggregate, and the other is a conditional temporal aggregate. Both FOL and OWL representations will be shown.

- *Every other week on Monday, Wednesday and Friday until December 24, 1997, but starting on Tuesday, September 2, 1997.*<sup>20</sup>

FOL:

```
( $\exists s, s', T, t_1, t_2$ ) [ $\text{everynthp}(s', \{T\}, \text{Week}1, 2) \wedge \text{byTulistRekurs}(s, \{1, 3, 5\}, s', *Day*, *Week*)$ 
 $\wedge \text{begins}(t_1, T) \wedge \text{ends}(t_2, T) \wedge \text{dateOf}(t_1, 1997, 9, 2) \wedge \text{dateOf}(t_2, 1997, 12, 24)$ ]
where ( $\forall w$ ) [ $\text{Week}1(w) \equiv (\exists n, x) [\text{callInt}(w, n, *Week*, x)]$ ]
```

OWL:

```
<time-entry:TemporalSeq rdf:ID="tseq">
  <time-entry:hasTemporalAggregateDescription rdf:resource="#MWFeveryOtherWeek" />
</time-entry:TemporalSeq>

<time-entry:TemporalSeq rdf:ID="tseq-everyOtherWeek">
  <time-entry:hasTemporalAggregateDescription rdf:resource="#everyOtherWeek" />
</time-entry:TemporalSeq>

<time-entry:TemporalAggregateDescription rdf:ID="everyOtherWeek">
  <time-entry:hasTemporalUnit rdf:resource="&time-entry;unitWeek" />
  <time-entry:hasGap rdf:datatype="&xsd;positiveInteger">2</time-entry:hasGap>
```

<sup>20</sup> This example is taken from iCalendar RFC 2445 page 120.

```

</time-entry:TemporalAggregateDescription>

<time-entry:TemporalAggregateDescription rdf:ID="MWFeveryOtherWeek">
  <time-entry:hasStart rdf:resource="#tseqStart" />
  <time-entry:hasEnd rdf:resource="#tseqUntil" />
  <time-entry:hasContextTemporalSeq rdf:resource="#tseq-everyOtherWeek" />
  <time-entry:hasithTemporalUnit rdf:datatype="&xsd;positiveInteger">1
</time-entry:hasithTemporalUnit>
  <time-entry:hasithTemporalUnit rdf:datatype="&xsd;positiveInteger">3
</time-entry:hasithTemporalUnit>
  <time-entry:hasithTemporalUnit rdf:datatype="&xsd;positiveInteger">5
</time-entry:hasithTemporalUnit>
  <time-entry:hasTemporalUnit rdf:resource="&time-entry;unitDay" />
  <time-entry:hasContextTemporalUnit rdf:resource="&time-entry;unitWeek" />
</time-entry:TemporalAggregateDescription>

<time-entry:Instant rdf:ID="tseqStart">
  <time-entry:inCalendarClock rdf:resource="#tseqStartDescription" />
</time-entry:Instant>

<time-entry:Instant rdf:ID="tseqUntil">
  <time-entry:inCalendarClockDataType rdf:datatype="&xsd;dateTime">1997-12-24
</time-entry:inCalendarClockDataType>
</time-entry:Instant>

<time-entry:CalendarClockDescription rdf:ID="tseqStartDescription">
  <time-entry:unitType rdf:resource="&time-entry;unitDay" />
  <time-entry:year rdf:datatype="&xsd;gYear">1997</time-entry:year>
  <time-entry:month rdf:datatype="&xsd;gMonth">9</time-entry:month>
  <time-entry:day rdf:datatype="&xsd;gDay">2</time-entry:day>
  <time-entry:dayOfWeekField rdf:datatype="&xsd;nonNegativeInteger">2
</time-entry:dayOfWeekField>
</time-entry:CalendarClockDescription>

```

The FOL axiom defines  $s$  as the set corresponding to the given temporal aggregate. The first part of the axiom defines  $s'$  as the set corresponding to “every other week”, and it serves as the context temporal sequence for the desired temporal sequence  $s$ . Predicates `begins` and `ends` are used to represent the start and the end times of the given temporal aggregate.

Besides what is shown in the first example, the OWL encodings for this one show how `hasithTemporalUnit` is used to represent a list of temporal elements in the temporal sequence (i.e., “on Monday, Wednesday, and Friday”), and how `hasStart` and `hasEnd` are used and combined with the calendar and clock representations to represent the start and end dates of the given temporal aggregate.

This example also shows the tradeoffs of using XSD `dateTime` and the `CalendarClockDescription` class defined in OWL-Time, as mentioned in Section 3. In this example, the end date is represented using XSD `dateTime`, while the start date is represented using the `CalendarClockDescription` class. As we can see, XSD `dateTime` is simpler, but there’s some information it cannot represent, for example, the start date is Tuesday, and this is the reason why `CalendarClockDescription` class (with `dayOfWeekField` property) is used for the start date. In fact, `CalendarClock-`

Description class can also represent other information that XSD `dateTime` cannot, such as “week” and “day of year”. Moreover, each field of the class is separate so that it’s easier to extract the values of some fields for the later use and easier to reason about.

- *Every Monday that's a holiday.*

FOL:

```
(∃ s,s0) [everyp(s,s0,HolidayMonday)]
where (∀ d) [HolidayMonday(d) ≡ (∃ w) [Monday(d,w) ∧ holiday(d)21]
```

OWL:

```
<EveryHolidayMonday rdf:ID="tseq" />

<owl:Class rdf:ID="EveryHolidayMonday">
  <rdfs:subClassOf rdf:resource="&time-entry;TemporalSeq"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMember" />
      <owl:allValuesFrom rdf:resource="#EveryHolidayMondayMember" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="EveryHolidayMondayMember">
  <rdfs:subClassOf rdf:resource="&time-entry;TemporalSeqMember"/>
  <rdfs:subClassOf rdf:resource="&time-entry;Holiday"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isMemberOf" />
      <owl:allValuesFrom rdf:resource="#EveryMonday" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="EveryMonday">
  <rdfs:subClassOf rdf:resource="&time-entry;TemporalSeq"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasTemporalAggregateDescription" />
      <owl:hasValue rdf:resource="#everyMonday" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<time-entry:TemporalAggregateDescription rdf:ID="everyMonday">
  <time-entry:hasithTemporalUnit rdf:datatype="&xsd;positiveInteger">1
</time-entry:hasithTemporalUnit>
  <time-entry:hasTemporalUnit rdf:resource="&time-entry;unitDay" />
</time-entry:TemporalAggregateDescription>
```

This example shows an important advantage of using our temporal aggregates ontology. It can represent *conditional temporal aggregates* which is hard or impossible in some

<sup>21</sup> It says *d* is a Holiday.

other representations. For example, there is no way in iCalendar to express such conditional temporal aggregates or any temporal aggregates that are not in a form of standard temporal units, such as “holidays”, “voting dates”, “days with classes”, “months starting with a Monday”, and so on.

As we can see, the FOL axiom is much simpler than the corresponding OWL encodings. It defines  $s$  as the set corresponding to the given temporal aggregate, and a new predicate (i.e., `HolidayMonday`) for this conditional temporal aggregate.

The OWL encodings show how this kind of conditional temporal aggregates can be defined in our representation in OWL.

The most important class in this example is the `EveryHolidayMondayMember` class which defines a class for the members of the desired temporal sequence class (i.e., `EveryHolidayMonday`). This class is both a temporal sequence member and a holiday, and its associated temporal sequence class must be “every Monday” (i.e., `EveryMonday` class).

The desired temporal sequence is an instance of the `EveryHolidayMonday` class whose members are only from the `EveryHolidayMondayMember` class.

## 5 Temporal Arithmetic

As long as we stay within the year-month system or the week-day-hour-minute-second system, temporal arithmetic is just arithmetic and requires only a few simple axioms to encode. When we mix months and days, problems arise.

We are currently working on a set of relatively simple rules that will allow us to do temporal arithmetic with months and days with a moderate degree of consistency. To get a flavor of the problems, consider that January 31, 2003, plus 2 months equals March 31, 2003. But if we add the months one at a time, we get a different result. January 31, 2003, plus one month is February 28, 2003. February 28, 2003, plus one month would seem to be March 28, 2003. If we want to avoid results like this, we need, in some sense, to keep track of the history of the computation. This motivating example is taken from the documentation for the Java functions `add()` and `roll()` of class `Calendar`<sup>22</sup>, where they explained the behavior of the functions using the above example and claimed that the final date after adding the second month should be March 31, 2003, not March 28, 2003, as most users will intuitively expect.

This problem has already been recognized by Biron and Malhotra (2004) for XML Schema datatypes<sup>23</sup>, where two new datatypes, `yearMonthDuration` (for the year-month system) and `dayTimeDuration` (the week-day-hour-minute-second system), were derived from datatype `duration`. The algorithm they developed to add durations to `dateTimes`<sup>24</sup> simply adds each of their fields/units, respectively, without overflow, and didn’t consider the problem mentioned in the above motivating example. Using their algorithm, the final date after adding the second month will be March 28, 2003. They

---

<sup>22</sup> <http://java.sun.com/j2se/1.4.2/docs/api/java/util/Calendar.html>

<sup>23</sup> <http://www.w3.org/TR/xmlschema-2/>

<sup>24</sup> <http://www.w3.org/TR/xmlschema-2/#adding-durations-to-dateTimes>

have also shown that there are cases that the properties of commutativity and associativity cannot hold using their algorithm:

$$(2000-03-30 + P1D) + P1M = 2000-03-31 + P1M = 2000-04-30$$

$$(2000-03-30 + P1M) + P1D = 2000-04-30 + P1D = 2000-05-01$$

We will show in this section the temporal arithmetic rules we created that will be able to handle the motivating example, and satisfy different desired properties.

Since the difficult part is when months and days are mixed, in this section, we will only concentrate on temporal arithmetic involving dates with both months and days, and all other fields of a date or time (e.g., years, hours) will be omitted in the examples and rules.

## 5.1 Desired Properties for Temporal Arithmetic Computation

Before creating the rules for temporal arithmetic computation, we need to have a list of properties which we hope can hold during the computation. These properties also guided the process of creating the rules. Here are the desired properties, with the most desired ones at the top:

- 1) Motivating example:      Jan. 31, 2003 + 1 month = Feb. 28, 2003  
    Feb. 28, 2003 + 1 month = March 31, 2003  
    Jan. 31, 2003 + 2 months = March 31, 2003

- 2) Addition-Simplicity property:  
 $(m_1, d_1)^{25} + [m_2, d_2]^{26} = (m_1+m_2, d_1+d_2)$ ,  
 if  $d_1+d_2 \leq$  number of days in month  $(m_1+m_2)^{27}$

- 3) Subtraction-Simplicity property:  
 $(m_1, d_1) - [m_2, d_2] = (m_1-m_2, d_1-d_2)$ ,  
 if  $d_1 > d_2, d_1-d_2 \leq$  number of days in  $(m_1-m_2)$ .

- 4) Subtraction property:      date1 + duration = date2  $\Leftrightarrow$  date2 - duration = date1  
     $\Leftrightarrow$  date2 - date1 = duration

(E.g., motivating example: March 31, 2003 - 1 month = Feb. 28, 2003  
 Feb. 28, 2003 - 1 month = Jan. 31, 2003

March 31, 2003 - Feb. 28, 2003 = 1 month  
 Feb. 28, 2003 - Jan. 31, 2003 = 1 month      )

<sup>25</sup> (m, d) denotes a date with a month m and a day d.

<sup>26</sup> [m, d] denotes a duration with m months and d days.

<sup>27</sup>  $(m_1+m_2)$  is the month where the result date is in.

- 5) Commutativity property:  $\text{date1} + \text{duration1} + \text{duration2}$   
 $= \text{date1} + \text{duration2} + \text{duration1}$
- 6) Associativity property:  $(\text{date1} + \text{duration1}) + \text{duration2}$   
 $= \text{date1} + (\text{duration1} + \text{duration2})$

## 5.2 Meaning of “Day Lost (DL)”

In order to keep track of the history of temporal arithmetic computation, we introduce a notion, called “day lost (DL)”. Its meaning is as follows:

1) *If the day of a month is the end of the month (e.g., (2, 28)<sup>3</sup>):*

Day lost (“DL”) means the number of days lost in the current month. For example, adding 1 month to January 31<sup>st</sup> results in February 28<sup>th</sup>, where 3 days are “lost” in the computation. Thus we use (2, 28)<sup>3</sup> to “remember” that 3 days were lost in February.

2) *Otherwise (e.g., (3, 2)<sup>3</sup>):*

Day lost is just a record of the number of days lost in the past. For example, adding 1 month and 2 days to January 31<sup>st</sup> results in March 2<sup>nd</sup>, where 3 days are “lost” in February.

“Day lost (DL)” not only is used to keep track of the history of the computation, but also plays a crucial role in explaining the inconsistency of the results with respect to different desired properties (e.g., commutativity and associativity). For example, it can be used to explain this inconsistency mentioned earlier:

$$\begin{aligned} (2000-03-30 + P1D) + P1M &= 2000-03-31 + P1M = 2000-04-30 \\ (2000-03-30 + P1M) + P1D &= 2000-04-30 + P1D = 2000-05-01 \end{aligned}$$

The different results are due to one day “lost” in the first computation when adding 1 month (P1M) to 2000-03-31, since there are 30 days (not 31 days) in April.

## 5.3 Temporal Arithmetic Rules

In this section, we will describe the rules we created for temporal arithmetic, including adding durations to dates, subtracting durations from dates, and computing duration between two dates. Examples will be shown to demonstrate how to use the rules. All the rules can be translated into FOL axioms straightforwardly.

**Notes on notations:**

- $(m, d)^{DL}$  denotes a *date* with a month “m”, a day “d”, and a day lost “DL”. For example,  $(2, 28)^3$  means February 28<sup>th</sup> with 3 days lost. DL is omitted when DL = 0.
- $[m, d]$  denotes a *duration* with “m” months and “d” days. For example,  $[3, 2]$  means 3 months and 2 days.
- $\#(m)$  denotes the number of days in month “m”. For example  $\#(3) = 31$ .

**Assumptions:**

- 1) The year is 2005, so that the number of days in February is 28.
- 2) Both dates and durations are in a canonical form:

For dates:

$$1 \leq m \leq 12$$

$$1 \leq d \leq \text{last day of the current month}$$

For durations:

$$1 \leq m \leq 11$$

$$1 \leq d < \text{last day of the resulting month}$$

**5.3.1 Adding Durations to Dates**

There are three rules for adding durations to dates: *add-decompose*, *add-month*, and *add-day*. In the following table, the rules are shown on the left column, and one representative example is shown for each line of the rules on the right column for the demonstration and justification purpose.

<b>Temporal Arithmetic Rule</b>	<b>Example</b>
<p><b>(1) Add-decompose:</b></p> $(m_1, d_1)^{DL} + [m_2, d_2]$ <p>(1.1) <math>= (m_1, d_1)^{DL} + [m_2, 0] + [0, d_2]</math></p>	$(2, 28)^2 + [1, 2]$ $= (2, 28)^2 + [1, 0] + [0, 2]$
<p><b>(2) Add-month:</b> // <math>N = \#(m_1+m_2)</math></p> $(m_1, d_1)^{DL} + [m_2, 0]$ <p>= if <math>(d_1 = \#(m_1))</math></p>	
<p>(2.1) <math>(m_1+m_2, d_1+DL), \quad d_1+DL \leq N</math></p>	$(2, 28)^2 + [2, 0] = (4, 30)$
<p>(2.2) <math>(m_1+m_2, N)^{d_1+DL-N}, \quad d_1+DL &gt; N</math></p>	$(3, 31) + [1, 0] = (4, 30)^1$
<p>else (i.e., <math>d_1 &lt; \#(m_1)</math>)</p> <p>(2.3) <math>(m_1+m_2, d_1), \quad d_1 \leq N</math></p>	$(3, 2)^3 + [1, 0] = (4, 2)$

(2.4)	$(m_1+m_2, N)^{d_1-N}$ ,	$d_1 > N$	$(1, 30) + [1, 0] = (2, 28)^2$
<b>(3) Add-day:</b>		// $N = \#(m_1)$	
	$(m_1, d_1)^{DL} + [0, d_2]$		
	= if ( $d_1 = \#(m_1)$ )		
(3.1)	$(m_1+1, d_2)^{DL}$		$(2, 28)^1 + [0, 1] = (3, 1)^1$
	else (i.e., $d_1 < \#(m_1)$ )		
(3.2)	$(m_1, d_1+d_2)^{DL}$ ,	$d_1+d_2 < N$	$(3, 2)^3 + [0, 20] = (3, 22)^3$
(3.3)	$(m_1, d_1+d_2)$ ,	$d_1+d_2 = N$	$(3, 2)^3 + [0, 29] = (3, 31)$
(3.4)	$(m_1+1, d_1+d_2-N)$ ,	$d_1+d_2 > N$	$(2, 20) + [0, 10] = (3, 2)$

For any given duration, before it adds to a date, *add-decompose* rule has to be applied first to separate the months and days by decomposing the duration to two durations with only months and days respectively, and then add months first to the date using *add-month* rule, then the days using *add-day* rule.

Here are two more examples of adding durations to dates. The first one is from the motivating example, and the second one demonstrates the complete procedure (3 steps) for adding durations to dates, when the duration has both months and days (the rule used for each computation is also shown with corresponding rule index numbers):

**E.g.1.**  $(1, 31) + [1, 0]$   
 $= (2, 28)^3$  // add-month (2.2)

$(2, 28)^3 + [1, 0]$   
 $= (3, 31)$  // add-month (2.1)

**E.g.2.**  $(1, 29) + [1, 2]$   
 $= (1, 29) + [1, 0] + [0, 2]$  // add-decompose (1.1)  
 $= (2, 28)^1 + [0, 2]$  // add-month (2.4)  
 $= (3, 2)^1$  // add-day (3.1)

### 5.3.2 Subtracting Durations from Dates

There are three rules for subtracting durations from dates: *sub-decompose*, *sub-day*, and *sub-month*:

Temporal Arithmetic Rule	Example
<p><b>(1) Sub-decompose</b></p> $(m_1, d_1)^{DL} - [m_2, d_2]$ <p>(1.1) <math>= (m_1, d_1)^{DL} - [0, d_2] - [m_2, 0]</math></p>	$(2, 28)^2 - [1, 2]$ $= (2, 28)^2 - [0, 2] - [1, 0]$
<p><b>(2) Sub-day</b></p> $(m_1, d_1)^{DL} - [0, d_2]$ <p>(2.1) <math>= (m_1, d_1 - d_2)^{DL}, \quad d_1 &gt; d_2</math></p> <p>(2.2) <math>(m_1 - 1, d_1 + \#(m_1 - 1) - d_2)^{DL}, \quad d_1 \leq d_2</math></p>	$(4, 30)^1 - [0, 15] = (4, 15)^1$ $(3, 2)^3 - [0, 10] = (2, 20)^3$
<p><b>(3) Sub-month</b> <span style="float: right;">// N = #(m<sub>1</sub>-m<sub>2</sub>)</span></p> $(m_1, d_1)^{DL} - [m_2, 0]$ <p>= if (d<sub>1</sub> = #(m<sub>1</sub>))</p> <p>(3.1) <math>(m_1 - m_2, d_1 + DL), \quad d_1 + DL \leq N</math></p> <p>(3.2) <math>(m_1 - m_2, N)^{d_1 + DL - N}, \quad d_1 + DL &gt; N</math></p> <p>else (i.e., d<sub>1</sub> &lt; #(m<sub>1</sub>))</p> <p>(3.3) <math>(m_1 - m_2, N)^{d_1 - N}, \quad d_1 &gt; N</math></p> <p>(3.4) <math>(m_1 - m_2, d_1), \quad d_1 \leq N</math></p>	$(2, 28)^2 - [1, 0] = (1, 30)$ $(4, 30)^1 - [2, 0] = (2, 28)^3$ $(3, 30)^2 - [1, 0] = (2, 28)^2$ $(5, 16) - [1, 0] = (4, 16)$

For any given duration, before it subtracts from a date, *sub-decompose* rule has to be applied first to separate the months and days by decomposing the duration to two durations with only days and months respectively, and then subtract days first from the date using *sub-day* rule, then the months using *sub-month* rule.

Here are two more examples of subtracting durations from dates. The first one is from the motivating example, and the second one demonstrates the complete procedure (3 steps) for subtracting durations from dates, when the duration has both months and days:

**E.g.3.**  $(3, 31) - [1, 0]$

$$= (2, 28)^3 \quad // \text{ sub-month (3.2)}$$

$$(2, 28)^3 - [1, 0]$$

$$= (1, 31) \quad // \text{ sub-month (3.1)}$$

**Note:** E.g.1. and E.g.3. show that the rules work for *the motivating example* (desired property 1), and the first part of the “*subtraction property*” (desired property 2) holds, since

$$\begin{aligned} (1, 31) + [1, 0] &= (2, 28)^3 \Leftrightarrow (2, 28)^3 - [1, 0] = (1, 31) \\ (2, 28)^3 + [1, 0] &= (3, 31) \Leftrightarrow (3, 31) - [1, 0] = (2, 28)^3 \\ (1, 31) + [1, 0] + [1, 0] &= (3, 31) \Leftrightarrow (3, 31) - [1, 0] - [1, 0] = (1, 31) \end{aligned}$$

**E.g.4.**  $(3, 2)^1 - [1, 2]$

$$\begin{aligned} &= (3, 2)^1 - [0, 2] - [1, 0] && // \text{ sub-decompose (1.1)} \\ &= (2, 28)^1 - [1, 0] && // \text{ sub-day (2.2)} \\ &= (1, 29) && // \text{ sub-month (3.1)} \end{aligned}$$

**Note:** E.g.2. and E.g.4. also show that the first part of the “*subtraction property*” (desired property 2) holds for this example, since

$$(1, 29) + [1, 2] = (3, 2)^1 \Leftrightarrow (3, 2)^1 - [1, 2] = (1, 29)$$

### 5.3.3 Computing Duration between Two Dates

There is only one rule for computing duration between two dates:

Temporal Arithmetic Rule	Example
<p><b>(1) Sub-between-dates</b></p> <p><math>(m_2, d_2, DL_2) - (m_1, d_1, DL_1)</math></p> <p>= if <math>(d_1 = \#(m_1))</math></p> <p>(1.1) <math>[m_2 - m_1, d_2 - (d_1 + DL_1)], \quad d_2 \geq d_1 + DL_1</math></p> <p>(1.2) <math>[m_2 - m_1 - 1, (d_2 + DL_2) + \#(m_2 - 1) - (d_1 + DL_1)],</math>  <math>d_2 &lt; d_1 + DL_1, d_2 &lt; \#(m_2)</math></p> <p>(1.3) <math>[m_2 - m_1, 0], \quad d_2 &lt; d_1 + DL_1, d_2 = \#(m_2)</math></p> <p>else (i.e., <math>d_1 &lt; \#(m_1)</math>)</p> <p>(1.4) <math>[m_2 - m_1, d_2 - d_1], \quad d_2 \geq d_1</math></p> <p>(1.5) <math>[m_2 - m_1 - 1, d_2], \quad d_2 &lt; d_1, d_2 &lt; \#(m_2), \#(m_2 - 1) \leq d_1</math></p> <p>(1.6) <math>[m_2 - m_1 - 1, d_2 + \#(m_2 - 1) - d_1],</math></p>	<p><math>(5, 31) - (2, 28)^2 = [3, 1]</math></p> <p><math>(3, 2)^3 - (1, 31) = [1, 2]</math></p> <p><math>(4, 30)^1 - (2, 28)^3 = [2, 0]</math></p> <p><math>(3, 20)^2 - (2, 10) = [1, 10]</math></p> <p><math>(3, 2) - (1, 28) = [1, 2]</math></p> <p><math>(3, 2) - (1, 20) = [1, 10]</math></p>

	$d_2 < d_1, d_2 < \#(m_2), \#(m_2-1) > d_1$	
(1.7)	$[m_2-m_1, 0], d_2 < d_1, d_2 = \#(m_2)$	$(2, 28)^1 - (1, 29) = [1, 0]$

Here are two more examples of computing duration between two dates. The first one is from the motivating example:

**E.g.5.**  $(3, 31) - (2, 28)^3$   
 $= [1, 0]$  // sub-between-dates (1.1)

$(2, 28)^3 - (1, 31)$   
 $= [1, 0]$  // sub-between-dates (1.3)

$(3, 31) - (1, 31)$   
 $= [2, 0]$  // sub-between-dates (1.1)

**Note:** E.g.1., E.g.3., and E.g.5. show the complete “*subtraction property*” (desired property 2) holds for the motivating example, since

$$(1, 31) + [1, 0] = (2, 28)^3 \Leftrightarrow (2, 28)^3 - [1, 0] = (1, 31)$$

$$\Leftrightarrow (2, 28)^3 - (1, 31) = [1, 0]$$

$$(2, 28)^3 + [1, 0] = (3, 31) \Leftrightarrow (3, 31) - [1, 0] = (2, 28)^3$$

$$\Leftrightarrow (3, 31) - (2, 28)^3 = [1, 0]$$

They also show the “*associativity property*” holds for the motivating example, since

$$\{(1, 31) + [1, 0]\} + [1, 0] = (3, 31) \Leftrightarrow (1, 31) + \{[1, 0] + [1, 0]\}$$

$$= (1, 31) + [2, 0] = (3, 31)$$

**E.g.6.**  $(3, 2)^1 - (1, 29)$   
 $= [1, 2]$  // sub-between-dates (1.5)

**Note:** E.g.2., E.g.4., and E.g.6. also show that the complete “*subtraction property*” (desired property 2) holds for this example, since

$$(1, 29) + [1, 2] = (3, 2)^1 \Leftrightarrow (3, 2)^1 - [1, 2] = (1, 29)$$

$$\Leftrightarrow (3, 2)^1 - (1, 29) = [1, 2]$$

**E.g.7.**  $(1, 31) + [1, 2] + [2, 0]$

$$\begin{aligned}
&= (1, 31) + [1, 0] + [0, 2] + [2, 0] && // \text{add-decompose (1.1)} \\
&= (2, 28)^3 + [0, 2] + [2, 0] && // \text{add-month (2.2)} \\
&= (3, 2)^3 + [2, 0] && // \text{add-day (3.1)} \\
&= (5, 2) && // \text{add-month (3.4)}
\end{aligned}$$

$$\begin{aligned}
&(1, 31) + [2, 0] + [1, 2] \\
&= (3, 31) + [1, 2] && // \text{add-month (2.1)} \\
&= (3, 31) + [1, 0] + [0, 2] && // \text{add-decompose (1.1)} \\
&= (4, 30)^1 + [0, 2] && // \text{add-month (2.2)} \\
&= (5, 2)^1 && // \text{add-day (3.1)}
\end{aligned}$$

**Note:** E.g.7. shows that the “*commutativity property*” holds for this example, if ignoring the supplemental information (day lost “DL”).

**E.g.8.** At the beginning of the section, we have shown the following example from the XML Schema datatype document<sup>28</sup> that demonstrates the case where the properties of commutativity and associativity cannot hold using their algorithm:

$$\begin{aligned}
(2000-03-30 + P1D) + P1M &= 2000-03-31 + P1M = 2000-04-30 \\
(2000-03-30 + P1M) + P1D &= 2000-04-30 + P1D = 2000-05-01
\end{aligned}$$

Can our rules handle this problem?

$$\begin{aligned}
&(3, 30) + [0, 1] + [1, 0] \\
&= (3, 31) + [1, 0] && // \text{add-day (3.3)} \\
&= (4, 30)^1 && // \text{add-month (2.2)}
\end{aligned}$$

$$\begin{aligned}
&(3, 30) + [1, 0] + [0, 1] \\
&= (4, 30) + [0, 1] && // \text{add-month (2.3)} \\
&= (5, 1) && // \text{add-day (3.1)}
\end{aligned}$$

---

<sup>28</sup> <http://www.w3.org/TR/xmlschema-2/#adding-durations-to-dateTimes>

I think the rules handle this problem very elegantly. They generated the result dates that we intuitively expect, but also include supplemental information (day lost “DL”) to explain the reason why the commutativity property doesn’t hold: there was “one day lost” in the computation when you add one day first, then one month to March 30. In fact, the commutativity property can be considered to hold, if we specify that  $(4, 30)^1$  is “equal” to  $(5, 1)$ .

All these rules can be translated into FOL axioms in OWL-Time straightforwardly. For example, the rule “add-month (2.1)”

$$(m_1, d_1)^{DL} + [m_2, 0] = (m_1+m_2, d_1+DL), \text{ if } (d_1 = \#(m_1)) \text{ AND } d_1+DL \leq \#(m_1+m_2)$$

can be translated as:

$$\begin{aligned} & \text{dateOf}(t_1, m_1, d_1, DL) \wedge \text{durationOf}(T, m_2, 0) \wedge \text{begins}(t_1, T) \\ & \wedge \text{ends}(t_3, T) \wedge \text{dateOf}(t_3, m_3, d_3, DL_3) \wedge \text{Hath}(n_1, *Day*, m_1) \\ & \wedge \text{Hath}(n_3, *Day*, m_3) \wedge d_1 = n_1 \wedge d_1+DL \leq n_3 \\ & \supset m_3 = m_1+m_2 \wedge d_3 = d_1+DL \wedge DL_3 = 0 \end{aligned}$$

Although we have demonstrated how desired properties hold for different examples using the temporal arithmetic rules, it’s not possible to enumerate all the possible cases, and it’s very possible to have some exceptional cases not covered or not hold for given properties. Please see Section 7 for the proposed future work on this part, where a more rigorous approach will be proposed.

## 6 Extracting Typical Event Durations from News Articles

Suppose we read the sentence, “George W. Bush met with Vladimir Putin in Moscow.” We don’t know how long that meeting lasted, but we do get *some* temporal information from the sentence. We know the meeting lasted more than ten seconds and less than one year. As we guess narrower and narrower bounds, our chances of being correct go down. Just how accurately can we make duration judgments like this? How much agreement can we expect among people? Will it be possible to extract this kind of information from text automatically?

As part of our commonsense knowledge, we can estimate roughly how long events of different types last and roughly how long situations of various sorts persist. For example, we know government policies typically last somewhere between one and ten years, while weather conditions fairly reliably persist between three hours and one day.

Such information about typical durations of events is very useful for temporal reasoning tasks that need to decide whether or not a given event is happening at a given time. For example, given the information that someone is having a meeting now, can I schedule another meeting for him 3 seconds from now, 3 hours from now, or 3 days from now? For reliability, narrow bounds of duration are needed if we want to infer whether

event  $e$  is happening at time  $t$ , while wide bounds of duration are needed to infer whether event  $e$  is *not* happening at time  $t$ .

An immense amount of such temporal information in text is encoded in event descriptions that do not look at all temporal. To our knowledge, there has been no serious effort to model the typical durations of events, and then to perform reasoning over this information. Our goal is to be able to extract such information from texts, and to that end we are annotating the events in news articles with bounds on their durations. With a large enough annotated corpus, it will be possible to apply machine learning techniques and investigate which lexical and other features are predictive of this kind of temporal information. A close examination of the annotated corpus may also yield some general principles from which durations can be predicated.

In Section 6.1 we first describe our annotation guidelines, including the annotation strategy and assumptions, and the representative event classes with examples. The inter-annotator agreement study and the experimental results will be discussed and shown in Section 6.2. The news articles that we used are from the TimeBank corpus (Pustejovsky et al., 2003).

## 6.1 Annotation Guidelines and Events Classes

Every event to be annotated was already identified in TimeBank. Annotators are asked to provide lower and upper bounds on the duration of the event, and a judgment of level of confidence in those estimates, which is measured in the scale of one to ten. Graphical output is displayed to enable us to visualize quickly the level of agreement among the different annotators for a single event. For example, here is the output of the annotations (3 annotators) for the event “finished” in the sentence

*... after the victim, Linda Sanders, 35, had **finished** her cleaning and was waiting for her clothes to dry, ...*

```
Event "finished":
s          mi          hr
|-----|-----|-----
          ===== 1: [1 mi, 5 mi, 8]
=====      2: [1 s, 10 s, 6]
=====      3: [5 s, 10 mi, 8]
```

This graph shows that the first annotator believes that the event lasts minutes whereas the second annotator believes it could only last for several seconds. The third annotator annotates the event to range from a few seconds to a few minutes. The confidence level of the annotators is generally subjective but as all three are higher than 5, it shows reasonable confidence. The logarithmic scale is used (see Section 6.2.1 for details).

### 6.1.1 Annotation Instructions

Annotators were asked to make their judgments as among the intended readers of the article, using whatever world knowledge was relevant in understanding the article. They were asked to identify upper and lower bounds that would include 80% of the possible cases. For example, rainstorms of 10 seconds or of 40 days and 40 nights might occur, but they are clearly anomalous and should be excluded.

There are two strategies for considering the range of possibilities:

1. Pick the most probable scenario, and annotate its upper and lower bounds.
2. Pick the set of probable scenarios, and annotate the bounds of their upper and lower bounds.

We deemed the second to be the preferred strategy.

The judgments were to be made in context. First of all, information in the syntactic environment was to be considered before annotation. For example, there is difference in the duration of watching in the phrases “watch a movie” and “watch a bird fly”.

Moreover, the events were to be annotated in light of the information provided by the entire article and with the knowledge that an ordinary intelligent reader would bring to the article. This meant that the entire article should be read before annotating any of it. One might learn in the last paragraph, for example, that demonstrations mentioned in the first paragraph lasted three days. That information should be used for annotation.

However, they were not to use knowledge of the future when annotating a historical document. For example, an article from the fall of 1990 may talk about the coming war against Iraq. We today know exactly how long that lasted. But annotators were asked to try to put themselves in the shoes of the 1990 reader of that article, and make their judgments accordingly. This is because we want people’s judgments of approximate durations, rather than the exact durations.

Annotation is made easier and more consistent if initially coreferential and near-coreferential descriptions of events are identified. If the same demonstrations are mentioned in the first sentence and in a subsequent sentence, they should be given identical duration ranges. In the sentence, “The Israeli government has been **faced** with a growing grassroots **movement calling** for Israel’s withdrawal...”, the events described by “faced”, “movement”, and “calling” are not identical, but their durations are probably the same.

### 6.1.2 Analysis

When the articles were completely annotated by the three annotators, the results were analyzed and the differences were reconciled. Differences in annotation could be due to the differences in interpretations of the event; however, we found that the vast majority of radically different judgments could be categorized into relatively small number of classes. Some of these correspond to aspectual features of events, which have been intensively investigated (e.g., Vendler, 1967; Moens and Steedman, 1988). We then developed guidelines to cover those cases. These guidelines were then used to annotate a test set. It was shown that the agreement in the test set was greater than the agreement obtained when annotations were performed without the guidelines. (See Section 6.2.3 for the experimental results).

### 6.1.3 Event Classes

**Action vs. State:** Events and actions involve change, such as those described by words like “speaking”, “gave”, “production”, and “skyrocketed”. States involve things staying the same, such as being dead, being dry, being at peace. When we have a passive verb construction, there is an ambiguity about whether we have a state or an event. Consider the following sentence

*Three people were **injured** in the attack.*

Does the word “injured” describe an event or the state that resulted from the event? This matters because they have different durations. We have the feeling here that it is the event that is referred to. But in certain cases it is hard to say whether we have an action or a state. For the sentence

*Farkas was **retired**.*

We would normally say that retirement is a state. But in

*Farkas was ordered home and **retired**.*

“retired” represents the action of retiring the person. A test that can be applied is this: Imagine someone saying the sentence after the action had ended but the state was still persisting. Would they use the past or present tense? In the “injured” example, it is clear we would say “Three people were injured in the attack”, whereas we would say “Three people are injured from the attack.” The latest version of our annotation interface handles events of this type by allowing the annotator to say which interpretation he or she is giving it.

**Aspectual Events:** Some events are aspects of larger events, such as their start or finish. One could consider these instantaneous. That is, for the sentence “Three men **entered** the laundromat.” we might want to say that there was an instant at which the third man crossed the threshold, and that counts as the entering. However, we can convert the sentence to the progressive and see that it happens across some interval.

*A shot rang out as three men were **entering** the laundromat.*

The action of entering involves reaching for the door, pushing or pulling it open, walking through it, and so on. All this takes time. Annotation of aspectual verbs should be as this kind of extended action, because this will give us more information on typical durations than simply assuming such things are instantaneous.

**Reporting Events:** These are everywhere in the news. They can either be a news reader reporting, or quotes of what someone else has mentioned. There can be situations where the reporting is actually a summarization of a long press conference or the reporting is an elaboration and explanation of the actual event. It is important to identify these different types of reporting events so that they can be accurately annotated. We may distinguish different cases:

**1) Summary/multiple report:** Often when a reporting event appears in the beginning of the news article, it summarizes a long discussion or press conference and thus should be annotated as a relatively long reporting event. For example,

*A Brooklyn woman who was watching her clothes dry in a laundromat was killed Thursday evening when two would-be robbers emptied their pistols into the store, the police **said**.*

In this sentence, everything that the spokesperson (here the police) has said is compiled into a single sentence by the reporters. It is unlikely that the spokesperson said only a single sentence with all this information. Thus, it is reasonable to say the saying event took place over a longer time, perhaps as long as half an hour.

**2) Quoted Report :** This is when the reported content is quoted. The duration of the event should be the actual duration of the utterance of the quoted content. These are the shortest type of reporting events. The time duration can be easily verified by saying the sentence out loud and timing it. An example of this kind is

*"It looks as though they panicked," a detective **said** of the robbers.*

Here the duration of the saying event will be the actual time it takes to say the quoted sentence.

**3) Single Unquoted Report:** This is when the reporting description occurs without quotes but does not seem to be a summary of a longer reporting event. It is a single reporting event which could be as short as just the duration of the actual utterance of the reported content (lower bound), and as long as the duration of a briefing or press conference (upper bound).

If the sentence is very short, then it's likely that it is one complete sentence from the speaker's remarks, and a short duration should be given; if it is a long, complex sentence, then it's more likely to be a summary, and a longer duration should be given.

*The police **said** it did not appear that anyone else was injured.*

*Hernandez was kidnapped from his small, neighborhood store in the town of Trujillo Alto at 10 p.m. Wednesday, police **said**.*

If the first sentence were quoted text, it would be very much the same. Hence the duration of the saying is short. In the second sentence the complement of "said" is less likely as a single utterance. The location might be mentioned in the introductory statement and the timing and date might be mentioned later.

**Multiple Events:** Many occurrences of verbs and other event descriptors refer to multiple events, especially, but not exclusively, if the subject or object of the verb is plural. For example,

*Many people **withdrew** their money from the bank before it crashed.*

*John **closed** out all his bank accounts.*

In the first sentence we may not know whether to annotate the time for the withdrawal by a single person removing money or the time required for all the people to remove their money. In the second sentence we may not know whether to annotate the closing for each account being closed or for all the accounts finally being closed. These are multiple events.

There is no ambiguity here between single and aggregate events; both really do happen. This was a significant source in disagreements in our first round of annotation. We have now modified the annotation interface to allow the coder to identify the event as multiple, and to estimate the duration of both the single and the aggregate events.

**Events Involving Negation:** Negated events didn't happen, so it may seem strange to specify their duration. But whenever negation is used, there is a certain class of events whose occurrence is being denied. Annotators should consider this class, and make a judgment about the likely duration of the events in it. For example, consider the following sentence.

*Police gave no details about the negotiations with the kidnappers for the return of Hernandez.*

There is the time it takes to give details, and there is the longer period during which no details were given.

This is similar to the case of multiple events, and as in that case, we now have annotators give estimates for both the event negated and the negation of that event.

**Positive Infinite Durations:** These are states which continue essentially forever once they begin, as in

*He is dead.*

*The earth revolves around the sun.*

Here the time durations continue for an infinite amount of time, and we allow this as an annotation.

## 6.2 Inter-Annotator Agreement

The kappa statistic (Krippendorff, 1980; Siegel and Castellan, 1988; Carletta, 1996), which factors out the agreement that is expected by chance, has become the de facto standard to assess inter-annotator agreement. It's computed as:

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)}$$

P(A) is the observed agreement among the annotators, and P(E) is the expected agreement, which is the probability that the annotators agree by chance.

In order to compute the kappa statistic for our task, we have to compute P(A) and P(E) first. However, those computations are not straightforward.

What should count as agreement among annotators for our task?

What is the probability that the annotators agree by chance for our task?

### 6.2.1 What Should Count as Agreement?

Figuring out what should count as agreement is not only important for accessing inter-annotator agreement, but also very crucial for later evaluating the machine learning techniques for our task. For example, for a given event with a known gold standard duration range from 1 hour to 4 hours, if a machine learning program outputs a duration of 3 hours to 5 hours for this event, how should we evaluate this result?

In the literature on the kappa statistic, most only handle category data (either in nominal scales or ordinal scales); some can handle more general data, such as data in interval scales or ratio scales (Krippendorff, 1980; Carletta, 1996). However, none of them directly apply to our data which is range durations from a lower bound value to an upper bound value.

In fact, what we annotated for a given event is not just a range, but a duration distribution for the event, where the area between lower bound and upper bound covers about 80% of the entire distribution area. Since it's natural to assume the most likely duration for such distribution is its mean (average) duration, and the distribution flattens

out toward the upper and lower bounds, we use the normal distribution (i.e., Gaussian distribution) to model our duration distributions.

In order to determine a normal distribution, we need to know the two arguments: the mean and the standard deviation. For our duration distributions with given lower and upper bound values, the mean is the average of the bound values. Under the assumption that the area between lower and upper bounds covers 80% of the entire distribution area, the lower and upper bounds are each 1.28 standard deviations away from the mean. Then the standard deviation can be computed using either the upper bound value ( $X_{upper}$ ) or the lower bound value ( $X_{lower}$ ):

$$\sigma = \frac{X_{upper} - \mu}{1.28} = \frac{X_{lower} - \mu}{-1.28}, \text{ where } \mu = \frac{X_{upper} + X_{lower}}{2}$$

After modeling the annotation data using the normal distribution, we need to convert our annotation values from temporal units to numerical values. One possible way is just to convert all the temporal units to seconds. However, this way would not capture correctly the relative relations between duration ranges. For example, the difference is significant between 1 second and 20 seconds, while from 1 year and 1 second to 1 year and 20 seconds, the difference should be negligible. Consider the range from 1 year to 5 years and the range from 1 second to 5 seconds. The distance between 1 year and 5 years in seconds would be much larger than that between 1 second and 5 seconds, but intuitively, they represent the same level of uncertainty. In order to handle this problem, we use logarithmic scale for our data. After first converting from temporal units to seconds, we then take the natural logarithm of the data. This logarithmic scale also conforms to the half orders of magnitude (HOM) (Hobbs, 2000) which was shown to have utility in several very different linguistic contexts.

With this data model, the agreement between two annotations is the overlapping area between two normal distributions. The agreement among many annotations is the average overlapping of all the pairwise overlapping areas. For example, for a given event suppose the two annotations are:

- 1) Lower: 10 minutes; upper: 30 minutes
- 2) Lower: 10 minutes; upper 2 hours

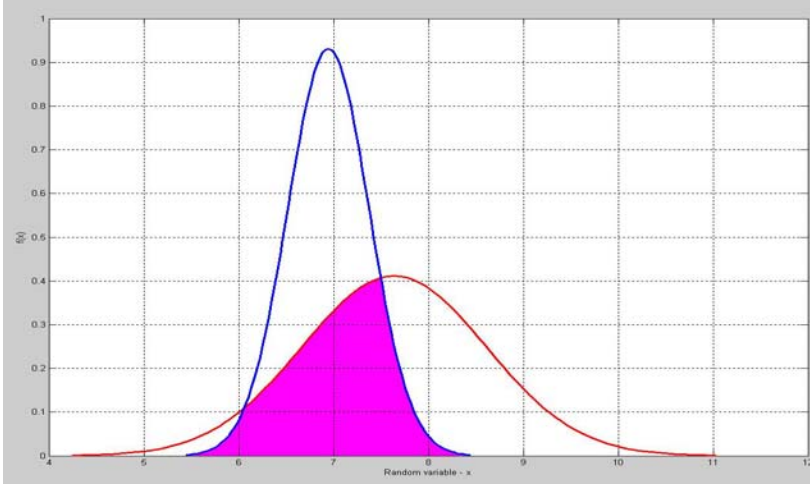
After converting to the natural logarithmic scale, they become:

- 1) Lower: 6.39692; upper: 7.49554
- 2) Lower: 6.39692; upper: 8.88184

We then compute their means and standard deviations:

- 1)  $\mu_1 = 6.94623$ ;  $\sigma_1 = 0.42861$
- 2)  $\mu_2 = 7.63938$ ;  $\sigma_2 = 0.96945$

Finally the distributions and the overlapping/agreement graph looks like:  
(overlapping/agreement = 0.508706)



### 6.2.2 Expected Agreement

What is the probability that the annotators agree by chance for our task? The first quick response to this question may be 0, if considering all the possible durations from 1 second to 1000 years or even positive infinity.

However, not all the durations are equally possible. As in (Krippendorff, 1980; Siegel and Castellan, 1988), we assume there exists one global distribution for our task (i.e., the duration ranges for all the events), and the annotations are consistent with this distribution. Since the “chance” for our task is based on the global distribution, in order to compute the expected agreement, we have to know this global distribution first.

With 15 annotated articles (a total of 587 events, 1761 annotated durations from three annotators), we first compute the distribution of the means of all the annotation durations. Its histogram graph is shown in Figure 6.1, where the horizontal axis represents the mean values in the natural logarithmic scale and the vertical axis represents the number of annotation durations.

There are two peaks in this distribution. One is from 5 to 7 in the natural logarithmic scale, which corresponds to about 1.5 minutes to 30 minutes. The other is from 14 to 17 in the natural logarithmic scale, which corresponds to about 8 days to 6 months. It is interesting to see that there are more events happening in minutes and months in the news articles we have annotated. One could speculate that this is because daily newspapers report short events that happened the day before and place them in the context of larger trends.

We also compute the distribution of the widths (i.e.,  $X_{upper} - X_{lower}$ ) of all the annotation durations, and its histogram graph is shown in Figure 6.2, where the horizontal axis represents the width values in the natural logarithmic scale and the vertical axis represents the number of annotation durations.

The peak of this distribution occurs at 2.5 in the natural logarithmic scale. This shows that for annotation durations, the most likely uncertainty factor from a mean (average) duration is 3.5:

$$\frac{X_{upper}}{\mu} = \frac{\mu}{X_{lower}} = e^{1.25} = 3.5, \text{ since } \log(X_{upper}) - \log(\mu) = \log\left(\frac{X_{upper}}{\mu}\right) = 2.5/2 = 1.25$$

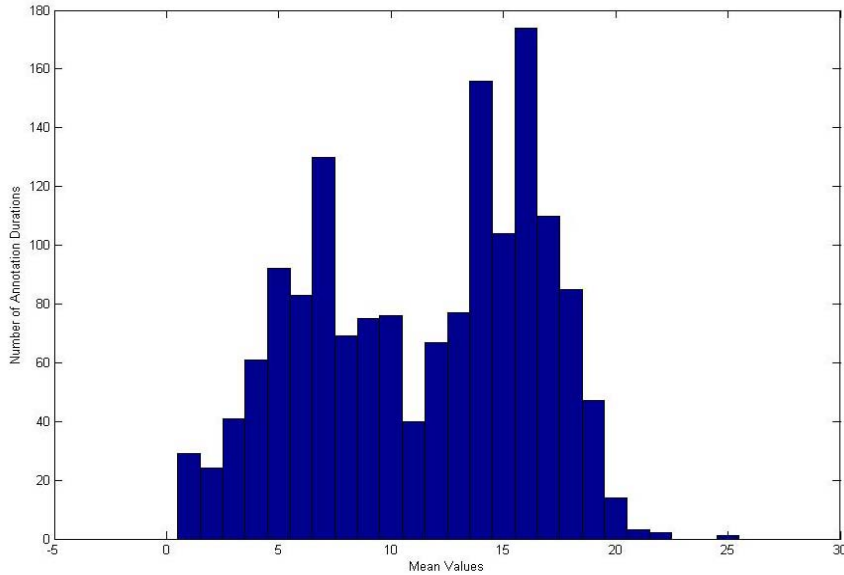


Figure 6.1: Distribution of means of annotation durations.

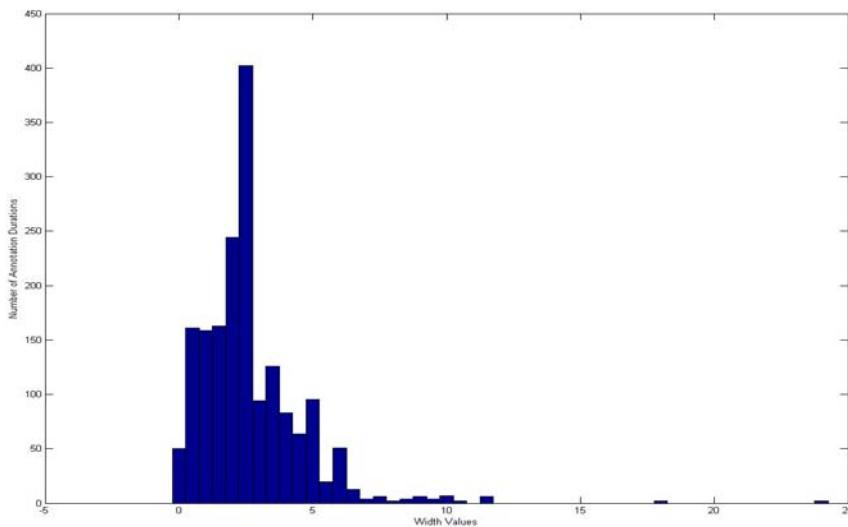


Figure 6.2: Distribution of widths of annotation durations.

This is actually at the HOM level that Hobbs and Kreinovich (2001) argue is the optimal granularity, the next estimate is 3 - 4 times larger than the previous one.

Since the global distribution is determined by the above mean and width distributions, we can then compute the expected agreement, i.e., the probability that the annotators agree by chance, where the chance is actually based on this global distribution. Two approaches were used to approximate this probability, both of which use a normal distribution to approximate the global distribution.

The first approach is to compute a fixed global normal distribution with the mean as the mean of the mean distribution and the standard deviation as the mean standard deviation. (This can be straightforwardly computed from the width distribution.) We then compute the expected agreement by averaging all the agreement scores (overlappings)

between this fixed distribution and each of the annotated duration distributions (1761 in all).

The second approach is to use 1000 normal distributions whose means are randomly generated from the mean distribution and standard deviations are randomly computed and generated from the width distribution. We then compute the expected agreement by averaging all the agreement scores (overlappings) between these 1000 random distributions.

In a sense, both of these capture the way an annotator might annotate if he or she did not read the article but only guessed on the basis of the global distribution.

As it turns out, the results of the two approaches of computing the expected agreement are very close; they differ by less than 0.01:  $P(E)_1 = 0.1439$ ,  $P(E)_2 = 0.1530$ . We will use the results of the second approach as the baseline in the next section.

### 6.2.3 Experiments

In order to see how effective our guidelines are, we conducted experiments to compare the inter-annotator agreement *before* and *after* annotators read the guidelines.

The data set is split into two sets. The first set contains 13 articles (521 events, 1563 annotation durations) which are all political and disaster news from ABC, APW, CNN, PRI, and VOA. The annotators annotated independently *before* reading the guidelines. The annotators were only given a short instruction on what to annotate and one sample article with annotations. The second set (test set) contains 5 articles (125 events, 375 annotation durations) which are also political and disaster news from the same news sources. The annotators annotated independently *after* reading the guidelines.

The comparison is shown in Figure 6.3, where the horizontal axis represents the overlapping thresholds and the vertical axis represents the agreement percentage, i.e., the percentage of annotation durations that agree for given overlapping thresholds. There are three lines in the graph. The top one with circles represents the after-guidelines agreement; the middle one with triangles represents the before-guidelines agreement; and the lowest one with squares represents the expected (baseline) agreement. This graph shows that, for example, if we define agreement to be a 10% overlap or better (an overlapping threshold of 0.1), we can get 0.8 agreement after reading the guidelines, 0.72 agreement before reading the guidelines, and 0.36 expected agreement with only the knowledge of the global distribution. From this graph, we can see that our guidelines are indeed effective in improving the inter-annotator agreement.

Table 6.1 shows more detailed experimental results. For each overlapping threshold, it shows the before-guidelines agreement, the after-guidelines agreement, and the expected (baseline) agreement, and also the kappa statistic computed from the after-guidelines agreement ( $P(A)$ ) and the expected (baseline) agreement ( $P(E)$ ).

Moreover, Table 6.1 also shows a factor value that represents how far apart the means of two annotations can be in order to overlap with the given overlapping threshold, assuming the width of the two annotations is the mean width (2.6 in the natural logarithmic scale), which can be derived from the width distribution in Section 6.2.2.

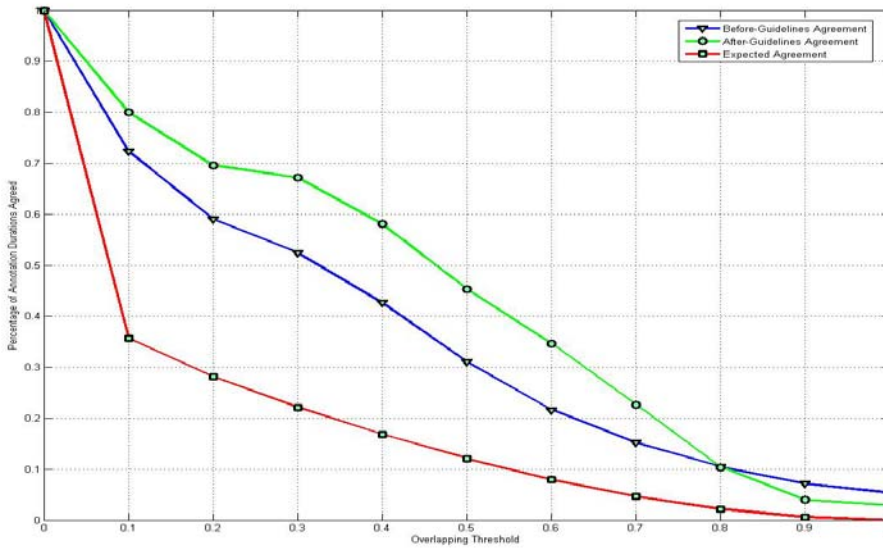


Figure 6.3: Inter-Annotator Agreement: Expected, Before-Guidelines, and After-Guidelines.

Overlapping Threshold	Expected Agreement	Before-G. Agreement	After-G. Agreement	Kappa (After-G. Agreement)	Factor
0.1	0.36	0.72	0.80	0.69	28.50
0.2	0.28	0.59	0.70	0.58	13.46
0.3	0.22	0.52	0.67	0.58	8.17
0.4	0.17	0.43	0.58	0.49	5.47
0.5	0.12	0.31	0.45	0.38	3.86
0.6	0.08	0.22	0.35	0.29	2.86
0.7	0.05	0.15	0.23	0.19	2.23
0.8	0.02	0.10	0.10	0.08	1.65
0.9	0.01	0.07	0.04	0.03	1.28
1.0	0.00	0.05	0.03	0.03	1.00

Table 6.1: Inter-Annotator Agreement with Different Overlapping Thresholds.

For example, when the overlapping threshold is 0.1, the factor value is 28.5, which means if one annotator guesses the mean duration for a given event is 1 minute, the other annotator will have 0.8 probability of guessing a mean duration from about 2 seconds (1 minutes / 28.5) to 28.5 minutes and their duration distributions will have at least 10% overlap. This also shows that if someone guesses 1 minute for a given event, it's not very likely (with a 0.2 probability) that the event will last more than 28.5 minutes or less than 2 seconds on average. Obviously, as Table 6.1 shows, if we want tighter bounds on the duration, our reliability will go down.

This factor is very useful for temporal reasoning tasks that need to know whether given events have already finished or not.

Event descriptions are a rich source of temporal information that has hitherto been untapped. Here we have described the beginning of an attempt to exploit that information.

### 6.3 Towards Learning Coarse-Grained Event Durations

We have almost finished annotating all the 48 non-financial (i.e. non-WSJ) articles (2195 events) in TimeBank. The rest of the un-annotated articles in TimeBank are all financial ones from WSJ.

The kinds of events appear in non-financial articles can be quite different from those in financial articles. In TimeBank, for example, the event “kill” appears many times in different non-financial articles (e.g., disaster and crime articles), while it doesn’t appear at all in any WSJ articles. On the other hand, the event “sale” occurs much more often in financial articles. Thus, it is reasonable to learn typical event durations from the current annotated non-financial articles first.

However, the size of this current annotated data is too small to any get good results using machine learning techniques to extract *fine-grained* duration information, where fine-grained duration includes both min duration and max duration for a given event, for example, from 3 minutes to 1 hour.

It may be possible, however, to learn *coarse-grained* duration information from the current corpus first. The most coarse-grained duration information can be binary duration information, i.e., classifying whether a given event is short or long. This binary classification of event durations for our corpus is justified by the bi-modal shape of the distribution of duration means in Figure 6.1. A natural binary division of the mean durations is the lowest point between the two peaks in Figure 6.1, i.e., 11 in the logarithmic scale (about 17 hours). A given fine-grained event duration can then be easily converted to the coarse-grained binary event duration by comparing its mean duration with the threshold (i.e., 11 in the logarithmic scale).

The same data sets (“before-guidelines” and “after-guidelines”) as the experiments in section 6.2.3. are used for the inter-annotator agreement study for this binary classification task, i.e., “before-guidelines” data set contains 13 articles (521 events, 1563 annotation durations) and “after-guidelines” data set contains 5 articles (125 events, 375 annotation durations). The results will not only show how much better annotators can agree on the coarser-grained binary event durations, but also give us an *upper bound* (human agreement) for this binary classification problem. Table 6.2 shows the experimental results.

Table 6.2 shows three Kappa values for each data set, according to different P(E) (expected/chance agreement) values. The first P(E) is computed based on the assumption that the distribution of proportions over the classes is equal for different annotators, i.e., there is one distributions for all annotators, derived from the total proportions of classes assigned by all annotators (Krippendorff, 1980; Siegel and Castellan, 1988); the second P(E) is computed based on the assumption that the distribution of proportions over the classes is *not* equal for different annotators, i.e., each annotator has a personal distribution, based on that annotator’s distribution of classes (Cohen, 1960); the third is computed based on the assumption that the distribution of proportions over the classes is

uniform, i.e., the annotators have no prior knowledge on the distribution of the classes. (Eugenio and Glass, 2004) points out that different ways of computing P(E) suffer from different bias and prevalence problems, and thus they should all be reported.

Before-Guidelines			After-Guidelines		
P(A)	P(E)	Kappa	P(A)	P(E)	Kappa
0.846	0.539	0.667	0.877	0.528	0.740
	0.539	0.667		0.526	0.741
	0.5	0.693		0.5	0.755

Table 6.2: Inter-Annotator Agreement for Binary Event Durations.

From the results shown in Table 6.2 we can see that the guidelines are effective in improving inter-annotator agreement for the binary classification of event durations. Since the agreement between humans (before reading the guidelines) is 84.6%, this can be used as an *upper bound* for this binary classification problem. A baseline program for this problem is to always assign the most frequent class, based on the proportion of the two classes from the current annotated data. Since based on the current annotated data  $P(\text{long events}) = 59.0\%$ , a *lower bound* (baseline) for this binary classification problem is 59.0%.

After having done this inter-annotator agreement study and decided the upper/lower bounds for this binary classification problem, the next step is to apply machine learning techniques to this corpus, and try to learn the coarse-grained binary event duration information. Please see the next section of proposed future work for more details.

## 7 Proposed Future Work

- In order to evaluate the ontology and demonstrate its reasoning capacity, we will implement a temporal reasoner that includes all the vocabulary and axioms in OWL-Time (including temporal aggregates) and make useful inferences for tasks from different applications. For FOL representation, we can use a FOL theorem prover (e.g., Otter (Kalman, 2001)), and all the axioms need to be converted into its input format first. For OWL representation, an OWL reasoner (e.g., Pellet<sup>29</sup>) can be used. However, since only a subset of OWL-Time axioms can be encoded in OWL, the kinds of inferences it can make are more limited.
- For temporal arithmetic, we need to be able to prove theoretically which desired properties can hold under what conditions, if not under all conditions, so that the user of the temporal arithmetic rules will have theoretical guarantees on what properties will hold for their computations. Since we also need to incorporate the temporal arithmetic rules into OWL-Time and all the rules can be translated into FOL

<sup>29</sup> <http://www.mindswap.org/2003/pellet/index.shtml>

straightforwardly, we can use a FOL theorem prover (e.g., Otter) to do the proof. Based on the result of the proof, we may generate different subsets of rules according to which properties are needed for different applications. For example, if an application doesn't need its temporal arithmetic computation to be commutable and associatable, the set of rules it need can be simpler.

- To automatically extract typical event durations, different machine learning techniques need to be explored with the current annotated non-financial articles to first learn to classify events into short and long events. If we can get satisfactory result from this binary duration classification task, as the size of the corpus grows, we can then move gradually from learning this most coarse-grained event durations to more and more fine-grained event durations, for example:

From: binary event duration (e.g. the event is *short* or *long*)

- Duration's most likely temporal unit (e.g., the event is most likely in *minutes* or *hours*)
- Duration's range in temporal units (e.g., the event lasts from *minutes* to *hours*)
- Duration's range in the half orders of magnitude (HOM) (e.g., the event lasts *from 5-20 minutes to 1-3 hours*)
- Duration's exact range (e.g., the event lasts *from 10 minutes to 1 hour*)

## 8 Thesis Contributions and Timeline

### 8.1 Contributions

- Developing a rich ontology of temporal concepts with axioms represented in both FOL and OWL Web Ontology Language, which can be used for the Semantic Web, natural language, and many other applications from different domains.
- Developing a temporal aggregates ontology that is rich enough to represent complex multiple-layered and conditional temporal aggregates in FOL and OWL, with a systematic way of mapping recurrence sets in iCalendar to temporal sequences in OWL-Time to gives it access to the full ontology of time for temporal reasoning.
- The first attempt to develop a set of rules for doing temporal arithmetic mixing months and days with a certain degree of consistency in terms of different desired arithmetic properties.
- The first attempt to model vague information about the typical duration of events, and then to perform reasoning over this information. Developing annotation guidelines for annotating events with bounds on their durations, categorizing representative event

classes, performing inter-annotator agreement study, and applying machine learning techniques to extract automatically such information from texts.

## 8.2 Timeline

- Phase 1. (3 months): Applying machine learning techniques to learn coarse-grained event duration information from the current annotated corpus.
- Phase 2. (5 months): Implementing and evaluating OWL-Time ontology using FOL theorem provers and OWL reasoners, including OWL-Time basic axioms, temporal aggregates ontology, and temporal arithmetic rules (also need to prove conditions for different properties to hold).
- Phase 3. (2 months): Thesis writing.

## 9 References

- Agarwal, P. 2005. Ontological considerations in GIScience. *International Journal of Geographical Information Science*, vol. 19, pp. 501-535. 2005.
- Allen, J. F. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23, pp. 123-154.
- Allen, J. F. and Ferguson, G. 1997. Actions and events in interval temporal logic. In *Spatial and Temporal Reasoning*. O. Stock, ed., Kluwer, Dordrecht, Netherlands, 205-245.
- van Benthem, J. 1983. *The Logic of Time*. Kluwer Academic Publishers.
- van Benthem, J. 1995. Temporal Logic, in D. M. Gabbay, C. J. Hogger, and J. A. Robinson, *Handbook of Logic in Artificial Intelligence and Logic Programming*, Volume 4, Oxford: Clarendon Press, pages 241-350.
- Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. The Semantic Web. *Scientific American*, 284(5):34-43, 2001.
- Bettini, C., Jajodia, S., Wang, X. S. 2000. *Time Granularities in Databases, Data Mining, and Temporal Reasoning*, Springer-Verlag, July 2000.
- Bettini, C., Wang, S. X. and Jajodia, S.. 2002. Solving multi-granularity temporal constraint networks, *Artificial Intelligence*, vol. 140, pp. 107-152.
- Biron, P. V. and Malhotra, A. 2004. XML Schema Part 2: Datatypes Second Edition. W3C Recommendation. <http://www.w3.org/TR/xmlschema-2/>
- Carletta, J. 1996. Assessing agreement on classification tasks: the kappa statistic. *Computational Linguistics*, 22(2):249-254.

- Chen, H.; Finin, T.; and Joshi, A. 2003. An ontology for context-aware pervasive computing environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 2003.
- Chen, H.; Perich, F.; Chakraborty, D.; Finin, T.; and Joshi, A. 2004. Intelligent agents meet semantic web in a smart meeting room. In *Proceedings of the third International Joint Conference on Autonomous Agents and Multi Agent Systems*, 2004.
- Cohen, J. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurements*, 20:37–46.
- Dawson, F. Stenerson, D. 1998. Internet Calendaring and Scheduling Core Object Specification (iCalendar), RFC2445, *Internet Society*.
- Dechter, R., I. Meiri and J. Pearl (1991). Temporal constraint networks. *Artificial Intelligence* 49: 61-95.
- Díaz, O.; Iturrioz, J.; Irastorza, A. 2005. Improving portlet interoperability through deep annotation. In *Proceedings of the 14th international conference on World Wide Web*, 2005.
- Dowty, D. 1979. *Word Meaning and Montague Grammar*, Dordrecht, Reidel.
- Dowty, D. 1982. Tenses, time-adverbs, and compositional semantic theory, *Linguistics and Philosophy*, 5, 23-55.
- Dumas, M.; O’Sullivan J.; Heravizadeh, M.; Edmond, D.; and Hofstede, A. 2001. Towards a semantic framework for service description. In *Proceedings of the IFIP Conference on Database Semantics*, Hong Kong.
- Eugenio, B. D. and Glass, M. 2004. The Kappa statistic: a second look. *Computational Linguistics*, 30(1):95–101.
- Galton, A. 1984. *The Logic of Aspect*, Clarendon Press Oxford.
- Genesereth, M. 1991. “Knowledge Interchange Format”, In *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning*, Allen, J., Fikes, R., Sandewall, E. (eds), Morgan Kaufman Publishers, pp 238-249.
- Han, B. and Lavie, A. 2004. A Framework for Resolution of Time in Natural Language. *TALIP Special Issue on Temporal Information Processing*, 2004.
- Harabagiu, S. and Bejan, C. 2005. Question Answering Based on Temporal Inference. In *Proceedings of the AAI-2005 Workshop on Inference for Textual Question Answering*, Pittsburgh, PA, USA, 2005.
- Haring, G., Juiz, C., Kurz, C., Puigjaner, R., and Zottl, J. 2004. Framework for the Performance Assessment of Architectural Options on Intelligent Distributed Applicatio. *Performance Metrics for Intelligent Systems (PerMIS '04)*, 2004.
- Hayes, P. 1995. A Catalog of Temporal Theories. *Tech report UIUC-BI-AI-96-01*, University of Illinois 1995.
- Hobbs, J. R. 2000. Half Orders of Magnitude, *KR-2000 Workshop on Semantic Approximation, Granularity, and Vagueness*, Breckenridge, Colorado.

- Hobbs, J. R. and Kreinovich, V. 2001. Optimal Choice of Granularity in Commonsense Estimation: Why Half Orders of Magnitude, In *Proceedings of Joint 9th IFSA World Congress and 20th NAFIPS International Conference*, Vancouver, British Columbia, pp. 1343-1348.
- Hobbs, J. R. 2002. Towards an Ontology for Time for the Semantic Web. In *Proceedings of LREC 2002 Workshop on Annotation Standards for Temporal Information in Natural Language*, pp. 28-35, Las Palmas, Spain, May 2002.
- Hobbs, J. R. and Pustejovsky, J. 2003. Annotating and reasoning about time and events. In *Proceedings of AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*, Stanford, CA, March 2003.
- Hobbs, J. R. and Pan, F. 2004. An Ontology of Time for the Semantic Web. *ACM Transactions on Asian Language Processing (TALIP): Special issue on Temporal Information Processing*, Vol. 3, No. 1, pp. 66-85.
- Jurafsky, D. and Martin, J. 2000. *Speech and Language Processing*, Prentice Hall.
- Kalman, J. A. 2001. *Automated Reasoning with Otter*, Rinton Press.
- Khushraj, D.; Lassila, O.; and Finin, T. 2004. Semantic Tuple Spaces, *Processing in Mobile Ad-Hoc Networks Proceedings of the International Conference on Mobile and Ubiquitous Systems: Networking and Services*, Boston, August 2004.
- Krippendorff, K. 1980. *Content Analysis: An introduction to its methodology*. Sage Publications.
- Lenat, D. 1995. "Cyc: A Large-Scale Investment in Knowledge Infrastructure," *Communications of the ACM*, 38, no. 11, November 1995.
- McDermott, D. 1982. A temporal logic for reasoning about processes and actions, *Cognitive Science*, 6,101-155.
- McGuinness, D. L. and Harmelen, F. v, 2003. OWL Web Ontology Language Overview. *World Wide Web Consortium (W3C) Candidate Recommendation*.
- McIlraith, S. A., Son, T. C. and Zeng, 2001. H. Semantic Web Services. *IEEE Intelligent Systems* 16(2):46-53.
- Medjahed, B., Bouguettaya, A. and Elmagarmid, A, 2003. Composing Web Services on the Semantic Web. *The Very Large Data Base Journal, Special Issue on the Semantic Web*, Springer Verlag, 12(4).
- Moens, M. and Steedman, M. 1988. Temporal Ontology and Temporal Reference. *Computational Linguistics* 14(2): 15-28.
- Moffitt, M. D. and Pollack, M. E. 2005. Applying Local Search to Disjunctive Temporal Problems, *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-2005)*, Aug. 2005.
- Moldovan, D.; Harabagiu, S.; Girju, R.; Morarescu, P.; Lacatusu, F.; Novischi, A.; Badulescu, A.; Bolohan, O. 2002. LCC Tools for Question Answering. In *Proceedings of the TREC-2002 Conference*, NIST. Gaithersburg, MD.

- Motakis, I. and C. Zaniolo, 1997. Temporal Aggregation in Active Databases Rules. In *International Conference on the Management of Data*, May.
- Nevatia, R.; Hobbs, J. R.; Bolles, B. 2004. An Ontology for Video Event Representation. *IEEE Workshop on Event Detection and Recognition*, June 2004.
- Niles, I. and Pease, A. 2001. Towards a standard upper ontology. In *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, 2001.
- OWL-S Coalition, 2004. OWL-S 1.1 Release. (<http://www.daml.org/services/owl-s/1.1/>)
- Pan, F. and Hobbs, J. R. 2003. A Time Zone Resource in OWL. Homepage at: <http://www.isi.edu/~pan/timezonehomepage.html>
- Pan, F. and Hobbs, J. R. 2004. Time in OWL-S. In *Proceedings of AAAI Spring Symposium on Semantic Web Services*, Stanford University, CA, March 2004.
- Pan, F. 2005. A Temporal Aggregates Ontology in OWL for the Semantic Web. In *Proceedings of the AAAI Fall Symposium on Agents and the Semantic Web*, Arlington, Virginia, 2005.
- Pan, F. and Hobbs, J. R., 2005. Temporal Aggregates in OWL-Time. In *Proceedings of the 18<sup>th</sup> International Florida Artificial Intelligence Research Society Conference (FLAIRS-2005)*, Clearwater Beach, Florida, pp. 560-565, AAAI Press.
- Pan, F. 2005. Temporal Aggregates for Web Services on the Semantic Web. In *Proceedings of the IEEE International Conference on Web Services (ICWS-2005)*, Orlando, Florida, pp. 831-832, IEEE Computer Society, 2005.
- Patel-Schneider, P. F. 2005. A Proposal for a SWRL Extension towards First-Order Logic, *W3C member submission*.
- Pease, A. 2004. Standard Upper Ontology Knowledge Interchange Format. Available at [http://cvs.sourceforge.net/viewcvs.py/\\*checkout\\*/sigmakee/sigma/suo-kif.pdf?rev=1.4](http://cvs.sourceforge.net/viewcvs.py/*checkout*/sigmakee/sigma/suo-kif.pdf?rev=1.4)
- Prior, A. 1957. Time and Modality. *Oxford: Oxford University Press*.
- Pustejovsky, J.; Gaizauskas, R.; Sauri, R.; Setzer, A.; Ingria, R. 2002. Annotation Guideline to TimeML 1.0, available at <http://time2002.org>.
- Pustejovsky, J.; Hanks, P.; Sauri, R.; See, A.; Gaizauskas, R.; Setzer, A.; Radev, D.; Sundheim, B.; Day, D.; Ferro, L.; and Lazo, M. 2003. The timebank corpus. In *Corpus Linguistics 2003*, Lancaster, U.K.
- Smith, C. 1991. The Parameter of Aspect, *Dordrecht, Reidel*.
- Siegel, S. and Castellan, N. J. 1988. Jr. Nonparametric Statistics for the Behavioral Sciences. *McGraw-Hill*, second edition.
- Steedman, M. 1997. Temporality. In J. van Benthem and A. ter Meulen, (eds.), *Handbook of Logic and Language*, Elsevier North Holland, 1997, 895-935.
- Steedman, M. 2002. The Productions of Time, *Draft tutorial notes about temporal semantics*, Draft 4.1, July 2002.

- Toivonen, S.; Denker, G. 2004. The Impact of Context on the Trustworthiness of Communication: An Ontological Approach. *ISWC Workshop on Trust, Security, and Reputation on the Semantic Web 2004*.
- Tsamardinos, I. and Pollack, M. E. Efficient Solution Techniques for Disjunctive Temporal Reasoning Problems, *Artificial Intelligence*, 151(1-2):43-90, 2003.
- Tuchinda, R.; Thakkar, S.; Gil, Y.; Deelman, E. 2004. Artemis: Integrating Scientific Data on the Grid, *Proceedings of Sixteenth Conference on Innovative Applications of Artificial Intelligence (IAAI)*, San Jose, California, 2004.
- Vendler, Z. 1967. *Linguistics in Philosophy*, Ithaca, Cornell University Press.
- Verkuyl, H. 1989. Aspectual classes and aspectual composition, *Linguistics and Philosophy*, 12, 39-94.
- Weissenberg, N. and Gartmann, R. 2003. Ontology Architecture for Semantic GeoServices for Olympia 2008, In: Bernard, L., A. Sliwinski and C. Senkler (Eds). *Münsteraner GI-Tage, Münster*, 2003. IfGIprints 18. 267-283.
- Zhou, Q. and Fikes, R. 2002. A Reusable Time Ontology. *Proceeding of the AAAI Workshop on Ontologies for the Semantic Web*, 2002.
- Zhu, H.; Madnick, S.E.; Siegel, M.D. 2004. Effective Data Integration in the Presence of Temporal Semantic Conflicts, *Proceedings of 11th International Symposium on Temporal Representation and Reasoning (TIME 2004)*, pp109-114, Normandie, France, July 1-3, 2004.