

# **Temporal Reasoning for RDF(S): A Markov Logic based Approach**

Jakob Huber

`jakob@informatik.uni-mannheim.de`

Chair of Artificial Intelligence  
Prof. Dr. Heiner Stuckenschmidt  
University of Mannheim

August 2014



# Abstract

In this work, we propose a formalism that is suitable to carry out temporal reasoning for probabilistic knowledge bases. In particular, we focus on detecting erroneous statements by exploiting temporal relations of facts. Therefore, we rely on RDF(S) [Hayes, 2004; Brickley and Guha, 2004] and its associating entailment rules which provide a data representation model as well as a basic logical expressiveness. Moreover, we use Allen's interval algebra [Allen, 1983] to express the relations of facts based on their associated temporal information. We carry out reasoning by transforming the statements and constraints to Markov Logic [Domingos and Lowd, 2009] and compute the most probable consistent state (MAP inference) with respect to the defined constraints. Moreover, we evaluate the proposed approach in order to demonstrate its practicality and flexibility.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Problem Statement . . . . .	3
1.3	Research Questions . . . . .	4
1.4	Outline and Contributions . . . . .	5
<b>2</b>	<b>Preliminaries</b>	<b>7</b>
2.1	Time Algebra . . . . .	7
2.2	RDF and RDFS . . . . .	9
2.3	Reasoning using Markov Logic . . . . .	13
<b>3</b>	<b>Approach</b>	<b>19</b>
3.1	Basic Idea . . . . .	19
3.2	Statements . . . . .	22
3.3	Constraints & Rules . . . . .	25
	3.3.1 RDF(S) Reasoning . . . . .	26
	3.3.2 Temporal Constraints . . . . .	29
3.4	Optimizations . . . . .	31
3.5	Discussion . . . . .	33
<b>4</b>	<b>Implementation</b>	<b>35</b>
4.1	Statements . . . . .	36
4.2	Constraints & Rules . . . . .	39
4.3	Interval Relations . . . . .	41
<b>5</b>	<b>Evaluation and Applications</b>	<b>53</b>
5.1	Standard RDF(S) Reasoning . . . . .	57
	5.1.1 Data & Constraints . . . . .	57
	5.1.2 Results & Discussion . . . . .	59
5.2	Linked Open Data - DBpedia Extract . . . . .	62

5.2.1	Data . . . . .	62
5.2.2	Constraints . . . . .	63
5.2.3	Experiments . . . . .	67
5.2.4	Discussion . . . . .	79
5.3	Sensor Data . . . . .	81
5.3.1	Dataset & Constraints . . . . .	81
5.3.2	Data Model . . . . .	85
5.3.3	Experiments . . . . .	93
5.3.4	Discussion . . . . .	101
<b>6</b>	<b>Related Work</b>	<b>105</b>
<b>7</b>	<b>Conclusion</b>	<b>115</b>
7.1	Summary . . . . .	115
7.2	Future Work . . . . .	117
	<b>Bibliography</b>	<b>iii</b>
<b>A</b>	<b>Markov Logic Model</b>	<b>ix</b>
A.1	Basic Model . . . . .	ix
A.2	RDF(S) Entailment Rules . . . . .	x
A.3	RDF and RDFS Vocabulary . . . . .	xi

# Chapter 1

## Introduction

This work is concerned with developing a formalism that is suitable to carry out temporal reasoning for probabilistic knowledge bases. In particular, we focus on detecting erroneous statements by exploiting temporal relations of facts. Therefore, we rely on RDF(S) [Hayes, 2004; Brickley and Guha, 2004] and its associating entailment rules which provide a data representation model as well as a basic logical expressiveness. Moreover, we use Allen’s interval algebra [Allen, 1983] to express the relations of facts based on their associated temporal information. We carry out reasoning by transforming the statements and constraints to Markov Logic [Domingos and Lowd, 2009] and compute the most probable consistent state (MAP inference) with respect to the defined constraints.

This chapter gives an introduction to this work and is structured as follows: First, we explain why the topic of this work is relevant (see Section 1.1). In Section 1.2, we present an example in order to illustrate the basic problems that we address with our approach. In the following section, we state the research questions of this work. Finally, we outline the structure of this work in Section 1.4.

### 1.1 Motivation

The Semantic Web [Berners-Lee et al., 2001] provides a framework that allows publishing data in a structured form on the Web. Linked Data [Bizer et al., 2009a] builds on techniques that are associated with the Semantic Web in order to create and to link structured datasets that are accessible by humans and machines. The Linked Open Data cloud<sup>1</sup> comprises a large amount of datasets covering various domains that adhere to the Linked Data principles [Bizer et al., 2009a]. The

---

<sup>1</sup><http://lod-cloud.net/>

published datasets use the Resource Description Framework (RDF) [Hayes, 2004] which is a graph-based data model that expresses facts as triples. Moreover, the datasets rely on the semantics of RDF that are provided by the standard vocabulary RDF(S) [Brickley and Guha, 2004]. It allows to structure data in an ontological form. However, many datasets are automatically generated by extracting content from unstructured sources. The datasets are not always consistent and contain erroneous statements [Mendes et al., 2012] as the information extraction algorithms are not perfect. Moreover, it is possible that linked datasets provide contradictory information. Hence, there is a need to detect erroneous statements and to resolve conflicts in order to improve the data quality.

However, the data quality of the most notable datasets of the Linked Open Data cloud is already good as the statements are derived from trustworthy and semi-structured sources (e.g., DBPedia [Auer et al., 2007; Bizer et al., 2009b; Lehmann et al., 2014], YAGO [Hoffart et al., 2011, 2013]). Other approaches collect the data on the Web by extracting information from unstructured web pages (e.g., NELL [Carlson et al., 2010], TextRunner [Yates et al., 2007], ReVerb [Etzioni et al., 2011]). Those open information extraction systems [Etzioni et al., 2008] collect millions of facts, i.e., relations between entities, which are annotated with confidence values that reflect the trust in the correctness of the statement. Moreover, some systems do also extract temporal information that indicates when a specific statement holds [Ling and Weld, 2010; Talukdar et al., 2012a]. Extracting temporal information is necessary as information changes over time (e.g., status, membership). The derived datasets contain a large amount of wrong statements due to limitations of the information extraction systems and the fact that some websites provide erroneous information. Thus, it is necessary to remove the respective statements in order to obtain a high quality knowledge base.

In summary, various approaches provide facts as triples (e.g., relations between entities) that are partially annotated with validity times and confidence scores. Hence, in order to detect inconsistencies it is possible to define (temporal) consistency constraints. Some researchers proposed methods to learn the temporal ordering of facts that are associated with an entity [Talukdar et al., 2012b]. The ordering of facts can be expressed in constraints and rules that are suitable to identify wrong statements. Moreover, the terminological part of the underlying ontology of a knowledge base enables to infer new facts as well as to detect inconsistencies. Thus, probabilistic and non-probabilistic statements and constraints respectively rules are associated with a knowledge base. Hence, a reasoning framework that considers all of these aspects is required to obtain high quality knowledge bases

that contain facts extracted from the Web. Moreover, other application areas, e.g., event recognition [Artikis et al., 2012], have comparable requirements.

## 1.2 Problem Statement

In this section, we provide an example that illustrates the problems that we want to address within this work. We mentioned in the previous section (see Section 1.1) that we target temporal probabilistic knowledge bases containing erroneous facts. Basic facts are modeled as triples that express relations between entities. Moreover, a fact can be annotated with a confidence value and temporal information. A confidence value states the probability that the statement is true while temporal information indicates when the statement holds. In order to detect inconsistencies, we define constraints and rules. In the following, we list facts which might be contained in a knowledge base and outline why constraints and rules are suitable to detect erroneous statements. The first set of facts states that `Jack` is the father of `John` and lists birth dates for both persons:

```
(F1) 0.7 (Jack, fatherOf, John)
(F2) 0.9 (Jack, birthYear, 1961) [1961]
(F3) 0.6 (Jack, birthYear, 1981) [1981]
(F4) 0.9 (John, birthYear, 1981) [1981]
```

Confidence values are attached to all facts as they are not retrieved from very trustworthy sources. In fact, based on general knowledge one can judge that at least one statement has to be wrong:

```
(C1) Persons have only one birthday.
(C2) Parents are born before their children are born.
```

These are both hard constraints that should never be violated in a consistent knowledge base. In order to define the respective constraints, it might be beneficial to rely on the underlying ontology of the knowledge base:

```
(T1) (fatherOf, domain, Person)
(T2) (fatherOf, range, Person)
```

These statements have neither confidence value nor temporal information assigned as they hold independently from a specific point in time. However, we can infer the type of `Jack` and `John` if we rely on terminological knowledge and the associated entailment rules. We extend the knowledge base with additional facts that state that `John` attended universities:

(F5) 0.9 (John, attended, University\_1) [2002, 2004]  
(F6) 0.4 (John, attended, University\_2) [2003, 2006]

Both facts are temporal probabilistic statements. For the following reason, one could argue that at least one fact is wrong:

(C3) Usually, someone attends only one university  
at a time.

This is a soft constraint that might be violated if the conflicting statements have a high confidence. Therefore, it is also necessary to annotate the respective constraint with a confidence value.

In this work, we present an approach that resolves those conflicts by considering the confidence values of the respective statements and constraints. Therefore, we propose a formalism that is suitable to express temporal and non-temporal as well as probabilistic and non-probabilistic facts and constraints. In order to carry out the reasoning process and to resolve inconsistencies, we use a Markov Logic solver.

### 1.3 Research Questions

This work is concerned with temporal reasoning in probabilistic knowledge bases using Markov Logic. Thus, we want to answer the following questions:

**RQ1** Is Markov Logic suitable for temporal probabilistic RDF(S) reasoning?

**RQ1-1** Can we propose a Markov Logic based formalism that allows to define the required types of statements (non-temporal and temporal, weighted and unweighted) and constraints?

**RQ1-2** Is the expressiveness of the proposed formalism sufficient, i.e., can we consider the RDF(S) entailment rules and incorporate temporal reasoning?

**RQ2** Implementation: Is it possible to express the required statements and constraints in a RDF document?

**RQ3** Evaluation: How well does the introduced approach scale?

**RQ4** Evaluation: Can the introduced approach be used to detect and to resolve conflicts? Are the results good?

The answers to Question RQ1, which comprises RQ1-1 and RQ1-2, are crucial as they affect all of the listed research questions. We need to develop a formalism that is suitable to express temporal and non-temporal as well as probabilistic and non-probabilistic facts and constraints (RQ1-1). Using this formalism it must be possible to resolve conflicts. Question RQ1-2 deals with possible limitations of the expressiveness of the selected framework and introduced formalism. For instance, we want to carry out RDF(S) reasoning and incorporate temporal facts. Hence, the formalism has to support the respective statements and rules. If the answers to RQ1 and the respective sub-questions are positive, we successfully developed a concept for temporal probabilistic RDF(S) reasoning.

The next question (RQ2) aims at the problem of annotating temporal information and probabilities to RDF statements. This is necessary as RDF does not explicitly support the required features. However, we want to express all facts of a dataset in a RDF document as RDF(S) provides the basic logical expressiveness for our approach. Moreover, we also want to define the required rules and constraints in RDF.

The remaining questions (RQ3, RQ4) are concerned with the analysis of our approach with respect to its practicality. It is important that the developed application is able to process knowledge bases that contain a large number of facts and conflicts. Thus, we investigate the scalability of the application (RQ3). Moreover, the application must also be able to detect and to resolve conflicts correctly in order to improve the data quality of the respective knowledge bases (RQ4). We achieved the primary goal of this work if the answers to these questions are positive.

## 1.4 Outline and Contributions

This document is structured in seven chapters. The first chapter (this chapter) gives an introduction to the topic and motivates the approach presented in this work. In Chapter 2, we introduce the frameworks and methods that provide the basis of our approach. We describe methods that are suitable to express temporal relations between facts (see Section 2.1). We also give a brief introduction to the RDF data model and the associated entailment rules (see Section 2.2). Moreover, we outline the characteristics of Markov Logic (see Section 2.3). In particular, we explain how we use it in the context of this work and also justify why we use the Markov Logic solver `rockIt` [Noessner et al., 2013] in order to implement our approach.

We present our approach and answer Question RQ1 in Chapter 3. The chapter starts with an outline of the basic concept that illustrates how we incorporate the different requirements (see Section 3.1). Subsequently, we introduce the types of the statements and constraints that are supported by our approach in Section 3.2 and Section 3.3. Moreover, we describe extensions of the proposed formalism that enable efficient reasoning and are required by some use cases in Section 3.4. We conclude Chapter 3 with a brief discussion of the presented approach in Section 3.5. In Chapter 4, we present and discuss details of the implementation of the proposed approach. In particular, we describe how we annotate statements with weights and temporal information in RDF (see Section 4.1). We introduce an approach that relies on the RDF(S) vocabulary in order to define constraints (see Section 4.2). Hence, these sections are concerned with Question RQ2. However, the most important section of this chapter is Section 4.3 in which we explore different models for calculating interval relations and justify why only the selected model is applicable. Thereby, we will partially answer Question RQ3.

In Chapter 5, we present and discuss the results of the evaluation. Hence, we give the answers to the questions RQ3 and RQ4 in this chapter. The chapter is split into three parts as we applied our approach to three different use cases. In Section 5.1, we test the basic functionality of our application by applying it to a benchmark (Lehigh University Benchmark [Guo et al., 2005]) for non-probabilistic and non-temporal knowledge base systems. This test case is relevant as it provides a possibility to investigate the scalability of our approach and serves as a baseline for more complex use cases. In Section 5.2, we derive facts from DBpedia in order to create datasets containing temporal and weighted statements. We use our application to detect and to remove erroneous statements from these datasets. Hence, this use case represents the primary application area of our system. The last part of the experiments (see Section 5.3) is used to demonstrate the flexibility of our approach by applying it to a different domain. So, we transform a sensor data dataset that is used to evaluate activity recognition algorithms to a RDF data model. Moreover, this use case is valuable as it requires a large amount of interacting weighted constraints and rules which increases the difficulty to resolve conflicts.

In Chapter 6, we give an overview on related work. Thereby, we focus on existing approaches for executing temporal reasoning in probabilistic knowledge bases. Additionally, we summarize the state of the research with respect to annotating RDF statements with temporal information and probabilities. Moreover, we outline how Markov Logic is used in related areas. Finally, we summarize this work and briefly answer the research question in Section 7.1. Moreover, we present options for future work (see Section 7.2).

## Chapter 2

# Preliminaries

In this chapter, we outline the foundations that are required to describe the problem and solutions presented in this work. Section 2.1 summarizes existing concepts to express relations between time points and time intervals. In Section 2.2, we present the characteristics of the standards RDF and RDFS. In Section 2.3, we explain the concept of Markov Logic and outline how it can be used for reasoning.

### 2.1 Time Algebra

In this section, we present a set of relations that can be used to express relations between temporal information. This is required as we want to define constraints that rely on such relations. The relations have to be jointly exhaustive and pairwise disjoint which gives us at least and at most one relation between two temporal annotated statements. Jointly exhaustive means that at least one of the relations holds between two time intervals while pairwise disjoint indicates that only one of the relations holds between two time intervals [Ligozat and Renz, 2004]. Thus, it is ensured that the relations allow to define precise constraints – depending on the granularity of the chosen constraint set. It also leads to an upper limit for the number of relations in a knowledge base. We can distinguish between time points and time intervals. A time interval is limited by two time points so that we have to rely implicitly on relations between time points even if our dataset contains only time intervals.

A time algebra does not only provide a set of relations but also a composition table. It allows us to check if the existing relations are correct and to infer missing information, i.e., which relations hold between intervals, given information about related statements. The composition tables in this section have to be read from left

	<	=	>
<	<	<	<,,>
=	<	=	>
>	<,,>	>	>

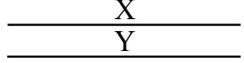
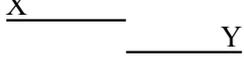
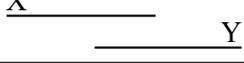
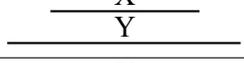
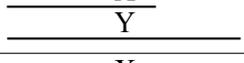
**Table 2.1:** Basic Point Algebra: Composition Table.

to right. Hence, given  $t_1 r_1 t_2 r_2 t_3$  ( $r_1, r_2$  being temporal relations and  $t_1, t_2, t_3$  being time points), we need to select  $r_1$  in the first column and  $r_2$  in the first row of the table in order to identify the cell that lists the relation  $r_3$  such that  $t_1 r_3 t_3$  holds. However, it is possible that a combination of two relations leads to more than one relation. As only one of them can be correct, we have to take related statements into account to determine the correct relation.

The basic point algebra [Vilain and Kautz, 1986] provides the relations *before* (<), *equal* (=), and *after* (>) that can be used to express the relation among time points. It would be possible to leave out the relation *before* or the relation *after* as they are inverse to each other but we think it is more convenient to use both of them. This is especially true when we calculate the relation between two intervals. Table 2.1 provides the respective composition table.

Allen [1983] defines a set of relations that hold between intervals. They are designed for situations in which the time course of events is critical. Moreover, they are also applicable when the known temporal relation is relative and not absolute which fulfills the need of many applications. The relations are *before* (<), *equal* (=), *meets* (*m*), *overlaps* (*o*), *during* (*d*), *starts* (*s*), *finishes* (*f*). The characteristics of these relations are outlined in Table 2.2. In total, there are thirteen relations as all but the relation *equal* have an inverse relation. However, these are not required in this work as the non-inverse relations are sufficient to formulate the constraints. The non-inverse relations are still jointly exhaustive when we apply them to an unordered pair of intervals. The relations *during*, *starts* and *finishes* can be collapsed to one containment relation *con*. This can be convenient in situations where the initial model is to fine-grained. Table 2.3 shows the composition rules of Allen's interval algebra. Based on the composition rules and the fact that the relations are pairwise disjoint, we are able to identify temporal conflicts (see Example 1).

**Example 1** We consider a knowledge base that contains the intervals  $a, b, c$  and the statements “ $a$  before  $b$ ” and “ $b$  before  $c$ ” and “ $c$  before  $a$ ”. Based on these statements, we can infer “ $a$  before  $c$ ” (Table 2.3). This leads to a conflict as the knowledge base also contains the statement “ $c$  before  $a$ ”. Hence, two different

Relation	Illustration	Interpretation
<i>X before Y</i>		X ends before Y starts.
<i>X equal Y</i>		X and Y have the same start and end point.
<i>X meets Y</i>		X starts before Y and ends just before Y starts.
<i>X overlaps Y</i>		X starts before Y and ends after Y starts and before Y ends.
<i>X during Y</i>		X starts after Y starts and ends before Y ends.
<i>X starts Y</i>		X starts when Y starts but ends before Y ends.
<i>X finishes Y</i>		X starts after Y starts and ends when Y ends.

**Table 2.2:** Allen Interval's Algebra: Overview on the Relations.

relations are stated between *a* and *c*. In order to resolve the conflict, it is necessary to remove one statement (e.g., *c before a*).

## 2.2 RDF and RDFS

In this section, we present the concepts of RDF 1.0 (Resource Descriptions Framework) [Hayes, 2004]<sup>1</sup> and RDFS 1.0 (RDF Schema) [Brickley and Guha, 2004]<sup>2</sup> which define the data model that we want to use in this work. Additionally, we explain an approach to annotate RDF statements as those statements do neither contain a weight nor a temporal interval.

RDF is a standard that is used in the Semantic Web [Berners-Lee et al., 2001] to model knowledge bases in an ontological form. The data is maintained in a data structure that corresponds to a labeled directed graph that expresses the relations among the resources represented in the knowledge base. A RDF database consists of statements having a *subject*, a *predicate*, and an *object*. Thus, a statement is

<sup>1</sup><http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>

<sup>2</sup><http://www.w3.org/TR/2004/REC-rdf-schema-20040210/>

	<	<i>m</i>	<i>o</i>	<i>s</i>	<i>f</i>	<i>d</i>	<i>e</i>
<	<	<	<	<	<, <i>o</i> , <i>m</i> , <i>d</i> , <i>s</i>	<, <i>o</i> , <i>m</i> , <i>d</i> , <i>s</i>	<
<i>m</i>	<	<	<	<i>m</i>	<i>d</i> , <i>s</i> , <i>o</i>	<i>o</i> , <i>d</i> , <i>s</i>	<i>m</i>
<i>o</i>	<	<	<, <i>o</i> , <i>m</i>	<i>o</i>	<i>d</i> , <i>s</i> , <i>o</i>	<i>o</i> , <i>d</i> , <i>s</i>	<i>o</i>
<i>s</i>	<	<	<, <i>o</i> , <i>m</i>	<i>s</i>	<i>d</i>	<i>d</i>	<i>s</i>
<i>f</i>	<	<i>m</i>	<i>s</i> , <i>o</i> , <i>d</i>	<i>d</i>	<i>f</i>	<i>d</i>	<i>f</i>
<i>d</i>	<	<	<, <i>o</i> , <i>m</i> , <i>d</i> , <i>s</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>
<i>e</i>	<	<i>m</i>	<i>o</i>	<i>s</i>	<i>f</i>	<i>d</i>	<i>e</i>

**Table 2.3:** Allen’s Interval Algebra: Composition Table.

also called triple. The predicate describes the relation between the *subject* and the *object* of the triple. Hence, in a graphical representation the *predicate* corresponds to a labeled directed edge from the node of the *subject* to the node of the *object*. In general, all elements of a RDF database are identified by an URI (Uniform Resource Identifier). However, this is not true for blank nodes and literals. Blank nodes are anonymous nodes that are implicitly created and thus, do not have an explicit URI. The object of an RDF triple can be a literal, i.e., a data value (e.g., dates) which does also not have an URI. A RDF knowledge base can be accessed using the query language SPARQL [Harris and Seaborne, 2013]<sup>3</sup>. It is similar to query languages of relation databases (e.g., SQL) and allows creating, modifying and deleting statements.

RDF(S) provides a vocabulary that allows to model additional vocabularies. Hence, it contains various classes and properties that are required to introduce a proprietary vocabulary. The most important classes and properties are listed in Table 2.4.

Moreover, the vocabulary supports the data structures collection and container. A collection is of type `rdf:List` and can be assembled using the properties `rdf:first` (first element), `rdf:rest` (next element) and `rdf:nil` (end of the list). A container can be ordered (`rdf:Seq`), unordered (`rdf:Bag`), or provide alternatives (`rdf:Alt`). Elements are added using the property `rdf:_n` (*n* being the number of the element). Another feature, which will be explained in the next section, is reification. It allows to annotate statements (RDF triples). Therefore, it uses the class `rdf:Statement` to create a node that identifies a statement and the properties `rdf:subject`, `rdf:predicate` and `rdf:object` to de-

<sup>3</sup><http://www.w3.org/TR/2013/REC-sparql11-query-20130321/>

<code>rdf:Resource</code>	The class resource, everything is type of this class.
<code>rdf:Class</code>	The class of classes.
<code>rdf:Property</code>	The class of (RDF) properties.
<code>rdf:type</code>	The subject is an instance of a class.
<code>rdfs:subClassOf</code>	The subject is a subclass of a class.
<code>rdfs:subPropertyOf</code>	The subject is a sub-property of a property.
<code>rdfs:domain</code>	The domain of a property, i.e., type of the subject.
<code>rdfs:range</code>	The range of a property, i.e., type of the object.

**Table 2.4:** RDF(S) Vocabulary: Most relevant Classes & Properties.

scribe the respective statement.

The standard vocabulary is used to define the RDF(S) entailment rules<sup>4</sup> (see Table 2.5) that a RDF(S) reasoner has to support. The rules `rdf1`, `rdf2`, `rdfs1`, `rdfs4a` and `rdfs4b` assign the respective RDF classes to the entities. The characteristics of these classes are partially expressed in the rules `rdfs6`, `rdfs8`, `rdfs9`, `rdfs10` and `rdfs13`. The rules `rdfs2/rdfs3` assign a type to the subject/object of a triple based on the domain/range of the property. The transitivity of the properties `rdfs:subPropertyOf` and `rdfs:subClassOf` is modeled in the rules `rdfs5` and `rdfs11`. Moreover, the consequences of the introduced hierarchy on the assertions are expressed in the rules `rdfs7` and `rdfs9`. The rule `rdfs12` is related to the container data structure in RDF.

### Annotating RDF Statements

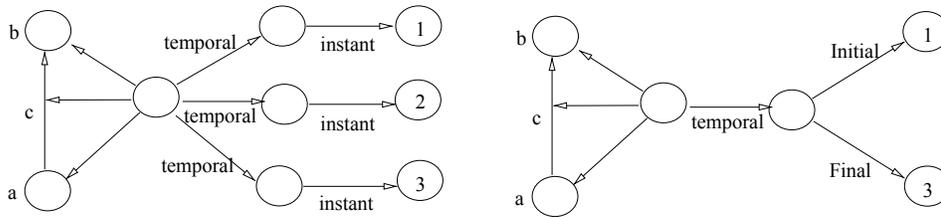
The standard RDF model misses two features with respect to our approach. First, in order to carry out temporal reasoning we need statements that are annotated with temporal information. Second, it would be beneficial if a weight is attached to each statement that expresses the confidence. RDF allows adding information to statements via reification<sup>5</sup>. Reification in RDF was introduced to add provenance to statements [Hayes, 2004], e.g., when was it created or who created it. The basic idea is to create a node of type `rdf:Statement` and to attach all parts of the respective triple using the properties `rdf:subject`, `rdf:predicate` and `rdf:object` to it. Hence, four triples are necessary to reify a statement. They are sometimes referred to as “reification quad”. In contrast to the intuitive assumption, the reification quad refers to a particular instance of a triple in the knowledge base

<sup>4</sup><http://www.w3.org/TR/2004/REC-rdf-mt-20040210/#rules>

<sup>5</sup><http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#reification>

Rule Name	If the data contains:	then add:
rdf1	uuu aaa yyy .	aaa rdf:type rdf:Property.
rdf2	uuu aaa lll . where lll is a well-typed XML literal.	_:nnn rdf:type rdf:XMLLiteral. where _:nnn identifies a blank node allocated to lll.
rdfs1	uuu aaa lll. where lll is a plain literal (with or without a language tag).	_:nnn rdf:type rdfs:Literal. where _:nnn identifies a blank node allocated to lll.
rdfs2	aaa rdfs:domain xxx. uuu aaa yyy.	uuu rdf:type xxx.
rdfs3	aaa rdfs:range xxx. uuu aaa vvv.	vvv rdf:type xxx.
rdfs4a	uuu aaa xxx.	uuu rdf:type rdfs:Resource.
rdfs4b	uuu aaa vvv.	vvv rdf:type rdfs:Resource.
rdfs5	uuu rdfs:subPropertyOf vvv. vvv rdfs:subPropertyOf xxx.	uuu rdfs:subPropertyOf xxx.
rdfs6	uuu rdf:type rdf:Property.	uuu rdfs:subPropertyOf uuu.
rdfs7	aaa rdfs:subPropertyOf bbb. uuu aaa yyy .	uuu bbb yyy.
rdfs8	uuu rdf:type rdfs:Class.	uuu rdfs:subClassOf rdfs:Resource.
rdfs9	uuu rdfs:subClassOf xxx. vvv rdf:type uuu.	vvv rdf:type xxx.
rdfs10	uuu rdf:type rdfs:Class.	uuu rdfs:subClassOf uuu.
rdfs11	uuu rdfs:subClassOf vvv. vvv rdfs:subClassOf xxx .	uuu rdfs:subClassOf xxx .
rdfs12	uuu rdf:type rdfs:ContainerMembershipProperty.	uuu rdfs:subPropertyOf rdfs:member.
rdfs13	uuu rdf:type rdfs:Datatype.	uuu rdfs:subClassOf rdfs:Literal.

**Table 2.5:** RDF(S) Entailment rules. “aaa”, “bbb”, stand for a URI reference of predicates; “uuu”, “vvv” stand for a URI reference or a blank node identifier; “xxx”, “yyy” stand for a URI reference, a blank node identifier or a literal; “lll” stands for a literal; and “\_:nnn” stands for a blank node identifier.



**Figure 2.1:** Point-based labeling (left) and interval-based labeling (right) [Gutierrez et al., 2005].

and not an arbitrary triple with the same subject, predicate and object. However, this cannot be expressed in RDF which makes application-dependent interpretation necessary in order to interpret the reification appropriately.

Gutierrez et al. [2005, 2007] developed an approach, called Temporal RDF, which is similar to the reification concept of RDF and allows assigning temporal information to RDF statements (see Figure 2.1). A statement is identified by a node that is connected to the triple (a b c). In order to create this node, they could have used the standard RDF reification vocabulary but they decided not to choose this vocabulary as they wanted to stress the fact that their concept is independent of any view about the concept of reification in RDF. However, they assign the temporal information to the node that identifies the statement using the property `temporal`. In particular, they support to add a time point, via the property `instant`, or to assign a time interval, via the properties `initial` and `final`.

In our work, we will use a concept that is similar to this approach as we also assign one or multiple intervals to a triple using reification (see Chapter 4). The examples in this work use the RDF syntax Turtle<sup>6</sup> [Carothers and Prud'hommeaux, 2014].

## 2.3 Reasoning using Markov Logic

In this section, we introduce the reasoning method that we use in this work as well as a concrete implementation. Thus, this section falls into two parts: First, we explain Markov Logic which is the framework of the selected reasoning approach. Second, we outline the key characteristics of the Markov Logic solver `rockIt` and justify why we choose this implementation.

<sup>6</sup><http://www.w3.org/TR/turtle/>

## Markov Logic

Markov Logic (ML) [Richardson and Domingos, 2006; Domingos and Lowd, 2009] combines probability and relational logic. This is required by many applications that have to deal with uncertainty, which is modeled by probability, and complexity, which is expressed in first-order logic. Hence, ML is the theoretic foundation of statistic relational learners that handle both aspects. The basic concept of ML is to attach weights to first-order formulas and to treat them as templates for features of a Markov Network (also known as Markov Random Field) that is used to compute the probability distribution of a set of random variables. In fact, a Markov Logic Network relies on Markov Networks having binary features (random variables).

In the following, we summarize the features of first-order logic, outline how a Markov Network is used to compute the probability distribution, and describe how Markov Logic combines these frameworks.

**First-order Logic.** First-order logic [Genesereth and Nilsson, 1987] allows constructing knowledge bases based on formulas that use constants, variables, functions, and predicates. Constants represent objects of the domain of interest, variables range over the objects of the domain, functions map tuple of objects to (other) objects, and predicates model the relations between the objects. The range of the variables can be limited by introducing types for constants and variables. Based on this, the following building blocks are defined:

- A **term** is an expression representing an object in the domain, i.e., a constant, a variable, or a function.
- An **atom** (atomic formula) is defined as a predicate symbol applied to a tuple of terms.
- A positive **literal** is a non-negated atom, a negative literal is a negated atom.
- A **formula** is recursively constructed from atomic formulas using logical connectors (conjunction ( $F_1 \wedge F_2$ ), disjunction ( $F_1 \vee F_2$ ), implication ( $F_1 \Rightarrow F_2$ ), equivalence ( $F_1 \Leftrightarrow F_2$ )) and quantifiers (universal quantification ( $\forall x F(x)$ ), existential quantification ( $\exists x F(x)$ )).

Moreover, a **ground term** is a term that does not contain variables, i.e., all variables are replaced by constants (grounding), and a ground atom is an atomic formula that has only ground terms. All formulas of a first-order logic knowledge base are implicitly conjoined, which leads to the requirement that a possible world must assign a positive truth value to each ground term. A possible world represents a

Weight	First-Order Logic	Weight	Clausal Form
1.0	$\forall x F_1(x) \Rightarrow F_2(x)$	1.0	$\neg F_1(x) \vee F_2(x)$
1.0	$\forall x F_1(x) \wedge F_2(x) \Rightarrow F_3(x)$	1.0	$\neg F_1(x) \vee \neg F_2(x) \vee F_3(x)$
1.0	$\forall x F_1(x) \Leftrightarrow F_2(x)$	0.5	$\neg F_1(x) \vee F_2(x)$
		0.5	$F_1(x) \vee \neg F_2(x)$
1.0	$\forall x F_1(x) \Rightarrow F_2(x) \wedge F_3(x)$	0.5	$\neg F_1(x) \vee F_2(x)$
		0.5	$\neg F_1(x) \vee F_3(x)$

**Table 2.6:** Converting First-Order Logic Formulas to the Clausal Form.

truth assignment to each possible ground atom. Hence, a formula  $F$  is satisfiable if there is at least one world in which it is true. A knowledge base  $KB$  entails a formula  $F$  ( $KB \models F$ ) if  $F$  is true in all worlds where  $KB$  is true. In order to apply automated inference, e.g., to determine if a knowledge base entails a formula, it is convenient to convert the formulas to the **clausal form** (conjunctive normal form). In this form is each clause a disjunction of literals. Table 2.6 shows how to convert universally quantified variables to clausal form. We skip the rules of existential quantified formulas as they are not required in the context of this work. We do also not include functions as we focus on function free first-order logic.

**Markov Network.** A Markov Network [Pearl, 1988] models the joint distribution for a set of variables  $X = (X_1, X_2, \dots, X_n)$ . It consists of an undirected graph  $G$  and a set of potential functions  $\phi_k$  for each clique  $k$ . Each variable is assigned to a node so that the joint distribution, which depends on the state of its cliques  $x_{\{k\}}$ , is given by

$$P(X = x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}}) \quad (2.1)$$

with the partition function  $Z = \sum_{x \in X} \prod_k \phi_k(x_{\{k\}})$ . In log-linear models [Koller and Friedman, 2009], the clique potential is replaced by an exponentiated weighted sum of the features of the state. A feature may be a real-valued function but it is also possible to define binary features, i.e.,  $f_j \in \{0, 1\}$ :

$$P(X = x) = \frac{1}{Z} \exp \left( \sum_j w_j f_j(x) \right) \quad (2.2)$$

One key feature of the Markov Network is that each node is independent from all others given its neighbors (i.e., its Markov blanket). This enables efficient inference algorithms. A Markov Logic Network relies only on binary features.

**Markov Logic Network.** Markov Logic [Richardson and Domingos, 2006] extends first-order logic by assigning weights to formulas. Thus, a world which violates a few formulas has still a probability greater than zero. A Markov Logic Network (MLN) is defined as a set  $L$  of pairs  $(F_i, w_i)$ , i.e., first-order formulas  $F_i$  with weights  $w_i$  that are real numbers, and a finite set of constants  $C = \{c_1, c_2, \dots, c_{|C|}\}$ . These two sets are used to create the Markov network  $M_{L,C}$  in the following way:

- $M_{L,C}$  contains one binary node for each possible ground atom appearing in  $L$  which has the value 1 if the ground atom is positive, and 0 otherwise.
- $M_{L,C}$  contains one feature (i.e., an edge) for each possible grounding of each formula  $F_i$  in  $L$  with its weight corresponding to the weight of the respective formula. The value of the feature is 1 if the ground formula is positive, 0 otherwise.

Ground atoms that appear together in a positive formula are connected by an edge whose weight is associated with the weight of the formula. Thus, a MLN is a template for constructing Markov Networks. Based on its definition and Equation 2.1 and 2.2 the probability distribution over possible worlds  $x$ , specified by the Markov Network  $M_{L,C}$ , is defined by:

$$P(X = x) = \frac{1}{Z} \exp \left( \sum_{i=1}^F w_i n_i(x) \right) \quad (2.3)$$

where  $F$  represents the number of formulas in the MLN and  $n_i(x)$  is the number of true groundings of  $F_i$  in  $x$ . So, a world which violates some constraints can still have a high weight. Contradictory formulas can be resolved by considering their weights. Moreover, increasing the weights to infinite makes the MLN a purely logic knowledge base.

**Maximum A-Posterior (MAP) Inference.** One basic inference task in Markov Logic is to determine the world which is most probable given some evidence. This task is called MAP inference and is equivalent to determine the world that maximizes the sum of weights in Equation 2.3:

$$\arg \max_x f(X = x | E = e) \quad (2.4)$$

The evidence variables ( $E$ ) are often called observed variables while the remaining variables ( $X$ ) are called hidden variables. Thus, the task of a Markov Logic query engine is to infer the variable assignment  $x$  that leads to the maximal probability. The respective assignment of  $x$  is called MAP state. Computing the MAP state

is a difficult inference task (inference in Markov Networks is #P-complete [Roth, 1996]) as local maxima in the Markov Network do not necessarily lead to the global maximum [Domingos and Lowd, 2009].

### **rockIt – A fast Markov Logic Solver**

Noessner et al. [2013] developed the Markov Logic solver rockIt that computes the MAP state of a Markov Logic Network in an efficient way. The system transforms a MAP query to an integer linear program (ILP) and makes use of different optimization techniques. In particular, they parallelize most parts of the process and apply cutting plane inference (CPI) [Riedel, 2008] and cutting plane aggregation (CPA). The idea is to add the violated constraints to the ILP until no violated grounded formulas exist. An ILP solver resolves the conflicts and returns additional ground formulas. Hence, this step is repeated several times as the intermediate solution can change after each iteration. It is proven that CPI always returns the optimal MAP state. rockIt is the first system that uses CPA which means that several ground clauses are aggregated to one ILP constraint. This leads to fewer variables in the ILP and more explicit symmetries that are easier to detect by the heuristics of ILP solvers. So, rockIt determines the MAP state by formulating compact ILPs that can be solved by a state of the art ILP solver. Another key feature is that the system leverages relational database management systems to perform the grounding of the formulas. Using a reliable and proven technology does not only lead to a good performance but it also makes it possible to only compute the groundings that match a formula. We decided to use rockIt in this work as experiments show that this system is more efficient and faster than other Markov Logic solvers [Noessner et al., 2013].

rockIt supports all features of Markov Logic that we will use in this work. This includes weighted formulas (soft constraints) and non-weighted formulas (hard constraints). The same holds for ground predicates. However, the predicates are grouped into observed predicates (evidence variables) and hidden predicates (non-evidence / query variables). Many Markov Logic solvers provide the possibility to define observed predicates as this allows evaluating the constraints more efficiently. This is caused by the fact that the closed world assumption holds for this type of predicates, i.e., ground atoms that are not listed in the evidence are considered as false. Hence, a ground formula containing a positive observed ground predicate can be directly evaluated. In contrast to this, negative observed predicates can be directly dropped from a ground formula.

However, all types of constraints can be used to define formulas. The formulas need to be expressed in the clausal form. It is possible but not required to assign a real-valued weight to a formula. Formulas without a weight are hard constraints that cannot be violated in the MAP state. All formulas are implicitly universally quantified but it is also possible to express existential formulas and cardinality formulas. However, the latter two are not required in the context of this work. We only use universally quantified formulas.

We give an overview on the syntax of the building blocks as we use the syntax of `rockIt` in this work. The formulas can only use the defined predicates. It is possible to define hidden predicates and observed predicates. Observed predicates are preceded by a star (“\*”):

```
*observedPredicate(x, x)
hiddenPredicate(x, x)
```

The variables of the formulas are typed which enables efficient grounding. In order to use weighted statements, it is necessary to define a helper predicate and a constraint that maps the helper predicate to a hidden predicate:

```
*weightedHelper(x, x, float_)
w: !weightedHelper(a, b, w) v hiddenPredicate(a, b)
```

Hence, the ground formula has the same weight as the respective ground axiom which leads to the same result as directly assigning it to the ground axiom. Soft constraints are preceded by a weight while hard constraints are terminated by a point (“.”). However, we only use the weighted helper predicates in Chapter 4 (Implementation), Chapter 5 (Evaluation) and Appendix A. In Chapter 3 (Approach), we directly attach the weights to the hidden predicates, i.e., `w hiddenPredicate(a, b)`, as this notation is more intuitive and general (not `rockIt` specific). An exclamation mark indicates a negated literal:

```
// hard constraint:
!observedPredicate(a, b) v hiddenPredicate(a, b) .

// soft constraint:
0.5 !hiddenPredicate(a, b) v hiddenPredicate(b, a)
```

Those are the features of `rockIt` that we use in this work. The ground axioms can be passed to the system by providing them in an evidence file. `rockIt` computes the MAP state and returns all ground axioms having a hidden predicate that are part of the MAP state.

## Chapter 3

# Approach

In this chapter, we introduce an approach to carry out reasoning for probabilistic temporal knowledge bases. Our approach bases on RDF(S) and Markov Logic. Hence, we give an overview on the chosen frameworks and justify why they fit to our approach in Section 3.1. In Section 3.2, we show how we express the RDF data model in Markov Logic and how we extend it with probabilities and temporal information. In order to enable reasoning, we need to define constraints and rules (see Section 3.3). In Section 3.4, we propose some optimizations that are useful in some cases. Finally, we discuss the introduced approach in Section 3.5.

### 3.1 Basic Idea

Knowledge bases that contain data which is extracted from the Web [Carlson et al., 2010; Etzioni et al., 2011; Bizer et al., 2009b] provide their content as tuples that have at least three elements. Hence, a relation connects two entities or assigns information to an entity. The Resource Description Framework (RDF) [Hayes, 2004] provides a structured and semantically enhanced data model that allows maintaining knowledge bases. It is widely used in the Semantic Web [Berners-Lee et al., 2001] and in particular in the Linked Open Data cloud [Bizer et al., 2009a]. Hence, it is reasonable to use this framework as it is a proven approach to maintain knowledge bases. However, the fact that the statements are only represented as triples introduces two problems with respect to our approach: First, it is not possible to directly assign a weight to a statement. Second, the statements are not temporal annotated. Those are only minor obstacles that can be circumvented and do not outweigh the advantages of the data model and its semantics.

The primary goal of our approach is to improve the data quality of a probabilistic knowledge base in terms of the correctness by removing erroneous statements. In order to detect inconsistent parts of a dataset, we need to define constraints and rules that have to be respected by the statements. In general, there are constraints that must always hold while others may be violated in some cases. Similar, some statements of a knowledge base are definitely true (non-probabilistic statements) while other statements are only true with a certain probability. Hence, we need a reasoner that handles both aspects. All of these required features can be modeled in Markov Logic. A common reasoning task in Markov Logic is to infer the consistent state that has the highest weight (MAP state) considering the weights (probabilities) of the statements and the defined constraints. Hence, the MAP state represents a consistent set of statements. This means that a Markov Logic solver removes statements from the dataset that are in conflict with other statements due to the defined constraints. Thereby, the reasoner resolves conflicts by removing the statements that are more likely erroneous. Thus, Markov Logic enables reasoning in probabilistic knowledge bases. However, the focus of this work is on temporal reasoning. Hence, we also use the relations of Allen's interval algebra in order to define temporal constraints.

In the following, we take up the example from Section 1.2 in order to explain how the MAP state conforms to a consistent dataset. We introduced a knowledge base containing the following statements:

```
(F1) 0.7 (Jack, fatherOf, John)
(F2) 0.9 (Jack, birthYear, 1961) [1961]
(F3) 0.6 (Jack, birthYear, 1981) [1981]
(F4) 0.9 (John, birthYear, 1981) [1981]
(F5) 0.9 (John, attended, University_1) [2002,2004]
(F6) 0.4 (John, attended, University_2) [2003,2006]
(T1) (fatherOf, domain, Person)
(T2) (fatherOf, range, Person)
```

Moreover, we proposed the following constraints that allow detecting erroneous statements:

```
(C1) Persons have only one birthday.
(C2) Parents are born before their children are born.
(C3) Usually, someone attends only one university
      at a time.
```

Markov Logic's MAP inference task identifies the subset of statements that are consistent with respect to the defined constraints. In the introduced example, we

see that all constraints rely on the temporal component of the statements. We want to use Allen’s interval algebra (see Section 2.1) that provides predicates that express temporal relations of intervals respectively facts. For instance, we can use the temporal relation `before` to define Constraint C2 or the relation `overlaps` to define Constraint C3. Moreover, we might want to make sure that Constraint C1 only applies for entities of the class `Person`. The initial dataset contains statements that express the birth year of entities (F2, F3, F4) but it is not explicitly stated that these entities are contained in the class `Person`. However, by applying the RDF(S) entailment rules we can infer `Jack` and `John` are persons due to Fact F1 and the terminological knowledge expressed in the statements T1 and T2. Hence, the following facts are in conflict due to the defined constraints:

- **V1:** F2 and F3 violate C1 as both statements express the birthday of `Jack`.
- **V2:** F3 and F4 violate C2 as both statements are annotated with the same temporal information and the fact that `Jack` is the father of `John` (F1).
- **V3:** F5 and F6 violate C3 as the annotated intervals overlap which means that `John` attended more than one university at a time.

In order to resolve the conflicting situations, it is necessary to remove statements from the dataset. For the conflicts **V1** and **V2**, it is sufficient to remove Fact F3. This fact has a lower weight than the facts F1, F2 and F4. Even though **V1** and **V2** are caused by different constraints, it is important to consider all related conflicting facts (F3 occurs in both sets) as local maxima do not necessarily lead to the global maximum. The violation **V3** can only be resolved if we assign a weight to Constraint C3. Only if the weight of the constraint is higher than 0.4, i.e., the weight of the fact with the lower weight (F6), it is necessary to remove Fact F6. We set the weight of the constraint to 0.6 what means that the statement F6 has to be removed. A Markov Logic solver handles such and more complex scenarios by computing the MAP state. Hence, the MAP state corresponds to the consistent subset of statements having the highest weight. So, we end up with the following consistent set of statements:

```
(F1) (Jack, fatherOf, John)
(F2) (Jack, birthYear, 1961) [1961]
(F4) (John, birthYear, 1981) [1981]
(F5) (John, attended, University_1) [2002,2004]
(T1) (fatherOf, domain, Person)
(T2) (fatherOf, range, Person)
(N1) (John, type, Person)
(N2) (Jack, type, Person)
```

The output of the Markov Logic solver contains not only all consistent statements that were contained in the initial dataset but also inferred statements ( $N1$ ,  $N2$ ). The statements  $N1$  and,  $N2$  got inferred due to the domain and range restriction of the property `fatherOf`. However, it is not necessary to take these statements into account when creating the cleansed RDF document.

## 3.2 Statements

We want to use Markov Logic to carry out reasoning. Hence, we need to define a Markov Logic model that is suitable to express all types of statements that might occur in a probabilistic temporal knowledge base. In general, such a knowledge base contains four different types of statements as there are two major distinction criteria that provide two possibilities:

**Standard vs. Temporal:** Not all statements of a temporal knowledge are temporal statements as some independently hold from a certain point in time. For instance, the terminological part of the underlying ontology of the knowledge base can be expressed with non-temporal (standard) statements.

**Weighted vs. Unweighted:** All statements that may be wrong should have a weight that expresses the probability that it is true. This allows the reasoner to remove them from the dataset. Contrary, statements that are definitely true have no weight. Hence, they cannot be removed from the dataset.

So, the type depends on the characteristic of the statement. The following Example 2 provides one statements per type.

**Example 2 Standard:** *The terminological part of the dataset is represented as standard triples which always hold.*

```
Scientist    subClassOf    Person.
```

**Standard (weighted):** *Relations between entities can be expressed as a weighted triple.*

```
0.95 John   type    Person.
```

**Temporal:** *Temporal statements that are not weighted are rare in our use case. They cannot be removed from the dataset.*

```
John    attended    University_1. [2002, 2004]
```

*Temporal (weighted): We primarily target these types of statements with our approach as they are temporal annotated and can be removed from the dataset.*

```
0.05 John attended University_1. [1980,1984]
```

We see that all types of statements rely on a triple consisting of a subject, a predicate and an object. The triple gets extended with temporal information (i.e., an interval) or a weight which can be interpreted as the likelihood that it is correct. So, we need first-order logic predicates that cover all types of statements. The weights are not problematic as they can be assigned to any ground predicate in Markov Logic. Thus, we only need to define predicates covering the following elements:

```
Subject Predicate Object
Subject Predicate Object Interval
```

The predicate of a statement expresses the relation between the subject and the object. In first-order logic it is common to define predicate symbols to express such relations. Thus, an intuitive model would be the following:

```
Predicate(Subject, Object)
Predicate(Subject, Object, Interval)
```

However, this model does not fulfill our requirements as our approach relies on the RDF data model and must be able to carry out RDF(S) reasoning. The proposed model cannot express all statements that are valid in RDF without defining a huge number of predicates. The most noticeable limitation is that a predicate of a triple can also be the subject or object of another triple (see Example 3). Hence, predicates of RDF triples would occur as predicate symbols as well as ground values of predicates. So, they are represented in two unrelated sets which hinder to extend the model with reasoning capabilities. Thus, the proposed model is impractical with respect to our approach.

**Example 3** *An element of a RDF triple can occur at any position.*

```
parentOf rdf:type rdf:Property.
Jack parentOf John.
fatherOf rdfs:subPropertyOf parentOf.
```

*Hence, the property parentOf occurs at any position of a triple.*

Hence, we need to define predicates that are more convenient for our use case. The previous observations indicate that the predicate of a RDF triple has to be part of the elements that are connected by a first-order logic predicate. So, the first-order logic predicate symbol does not indicate the relation. It can only express the type of the statement which leads to the following predicates:

```
triple(Subject, Predicate, Object)
quad(Subject, Predicate, Object, Interval)
```

This model allows us to express any RDF triples as they can be directly transformed to ground axioms having one these predicates. The same holds for temporal annotated statements which have an interval as fourth element (see Example 4). We map all intervals (and time points) that occur in a dataset to a unique identifier (see Section 4.3). However, this procedure can lead to a situation in which the object of a statement expresses the same fact as the annotated interval (see Example 5). We accept this as a generic model provides more flexibility which outweighs the drawbacks of redundant information.

**Example 4** *We transform the statements of Example 2 to the introduced model:*

**Standard:**

```
Scientist    subClassOf    Person
=>
triple(Scientist, subClassOf, Person)
```

**Standard (weighted):**

```
0.95 John    type    Person
=>
0.95 triple(John, type, Person)
```

**Temporal:**

```
John    attended    University_1 [2002,2004]
=>
quad(John, attended, University_1, interval1)
interval1 := [2002,2004]
```

**Temporal (weighted):**

```
0.05 John    attended    University_1 [1980,1984]
=>
0.05 quad(John, attended, University_1, interval2)
interval2 := [1980,1984]
```

**Example 5** *Instantiating the introduced model may lead to redundant information in a statement. The statement: “John is born in the year 1981” can be expressed as the following triple:*

```
John    birthYear    1981
```

However, in order to interpret it as a temporal statement, we need to annotate it:

```
John    birthYear    1981 [1981,1981]
```

This leads to following ground predicate:

```
quad(John, birthYear, 1981, intervall)
intervall := [1981,1981]
```

So, the object and the interval express the same information.

We introduced predicate symbols that can be used to express any type of statement that we target with our approach. However, the characteristics of Markov Logic allow us to extend the model in order to enable more efficient reasoning. It is possible to restrict the elements of a first-order logic predicate to a specific type in order to limit the possible groundings. We have to deal with resources of a RDF dataset (nodes in the data graph) and with intervals. Hence, the subject, predicate and object need to have the same type. This is perfect for all types of RDF resources but the literals. It is necessary to highlight literals (e.g., objects of statements having a datatype property) because some RDF(S) entailment rules apply only for them. Hence, we need to add another observed predicates that allows us to declare which resources are in fact literals. We can use an observed predicate as it is known which resources are literals before we execute the Markov Logic solver. So, we rely on the following predicate symbols:

```
triple(r, r, r)
quad(r, r, r, Interval)
*literal(r)
```

In the next section, we use these predicates to define constraints and rules.

### 3.3 Constraints & Rules

In this section, we outline which constraints and rules are supported by our approach. We decided to rely on RDF as it provides not only a data model but also gives a lower bound in terms of the logical expressiveness due to the vocabulary and entailment rules provided by the RDF(S) standard (see Section 3.3.1). Moreover, we include the temporal relations of Allen's interval algebra in order to enable temporal reasoning (see Section 3.3.2).

We use a Markov Logic solver as reasoner as it supports all required types of constraints and rules. We can define constraints that always hold (hard constraints)

or constraints which have a weight and can be violated (soft constraints) if this leads to a MAP state with a higher weight. The constraints can be defined using the introduced predicate symbols. Constraints and rules need to be written as implications ( $A \Rightarrow B$ ) and it must also be possible to transform them to the conjunctive normal form. Hence, the consequence (right-hand side) of an implication cannot be a conjunction of literals that are connected by a variable that does not occur on the left-hand side of the implication. Moreover, all variables of the constraints are implicitly universally quantified.

Hence, the supported constraints are disjunctions of literals having the predicate `triple` or `quad` (see Section 3.2). Moreover, the constraints can contain predicates that are associated to the relations of Allen's interval algebra (see Section 3.3.2). We use those building blocks to define constraints and rules. However, we usually refer to them as constraints because we convert them to formulas that are used to detect inconsistent statements. We express the RDF(S) entailment rules as well as domain-specific constraints with the same formalism.

### 3.3.1 RDF(S) Reasoning

In order to be able to carry out RDF(S) reasoning, we need to include the respective entailment rules (see Table 2.5 in Chapter 2) as well as the RDF<sup>1</sup> and RDFS<sup>2</sup> vocabularies. It is necessary to consider the vocabularies as the entailment rules rely on them. We can directly transform them to the introduced first-order logic predicate symbol `triple` as both vocabularies are defined in RDF documents (see Appendix A.3).

The documentation groups the entailment rules in the categories “simple entailment rules”, “RDF entailment rules”, “RDFS entailment rules” and “datatype entailment rules”. We focus on the RDF(S) entailment rules as the others rule sets are not required by our approach. The “simple entailment rules” create generalizations of the RDF graph by allocating blank nodes to the subjects and objects of all triples. Hence, the original graph is an instance of the inferred graph. Those rules also ensure that every sub-graph of the original graph is an instance of the entailed graph. However, we implicitly apply those rules by mapping every resource of the graph to a constant in the Markov Logic Network. We disregard the “datatype entailment rules” as they are not relevant for our approach. We treat all entities occurring in a RDF graph as resources and do only highlight which of them are

---

<sup>1</sup><http://www.w3.org/1999/02/22-rdf-syntax-ns>

<sup>2</sup><http://www.w3.org/2000/01/rdf-schema>

literals. Thereby, we do not distinguish between the types of the literals. In consequence of that, we leave out the rule `rdf2` and apply the rule `rdfs1` to any literal that occurs in the graph. Nevertheless, we consider all “RDF entailment rules” and “RDFS entailment rules” but the extensional RDFS entailment rules. The extensional rules are only valid for the stronger semantic conditions of the RDFS vocabulary and require more complex inference rules. We do not include those rules as there is no complete set of the required rules available. Moreover, the rules are not valid with respect to the standard semantic conditions of RDFS.

The entailment rules are defined as implications that require adding another triple to the knowledge base if the conditions are fulfilled. The implications have either only one condition ( $A \Rightarrow B$ ) or two linked conditions ( $A \wedge B \Rightarrow C$ ) which allow inferring another statement. The rules are all hard constraints and must be respected in a valid RDF(S) dataset. Hence, we can transform them to the proposed Markov Logic model. We give one example for each rule type. So, rule `rdfs8` adds the assertion that each instance of the class `rdfs:Class` is a subclass of the class `rdfs:Resource`:

$$\text{triple}(c, \text{"rdf : type"}, \text{"rdfs : Class"}) \Rightarrow \text{triple}(c, \text{"rdfs : subclassOf"}, \text{"rdfs : Resource"})$$

Another example is rule `rdfs5` that exploits the transitivity of the property `rdfs:subPropertyOf`:

$$\begin{aligned} \text{triple}(a, \text{"rdfs : subPropertyOf"}, b) \wedge \text{triple}(b, \text{"rdfs : subPropertyOf"}, c) \\ \Rightarrow \text{triple}(a, \text{"rdfs : subPropertyOf"}, c) \end{aligned}$$

We define the other rules in the same way (see Appendix A.2). The variables that occur in the rules are partially restricted to specific types of entities, e.g., a variable can only be grounded with a property. We are not required to check this explicitly as the properties used in the rules already ensure that the restrictions are not violated, e.g., by their domain and range restrictions. However, the rules do not recognize if the original dataset is not valid.

So far, the defined entailment rules apply only to ground values using the predicate symbol `triple`. However, some knowledge that is encoded in temporal annotated statements (`quad`) needs also to be taken into account. Therefore, we define a rule that converts the respective statements:

$$\text{quad}(s, p, o, i) \Rightarrow \text{triple}(s, p, o)$$

This rule ensures that the non-temporal part of statements can be used for non-temporal reasoning. We argue that this rule is not problematic as long as constraints that do only apply for temporal statements are only defined for such statements. For instance, we should not define that two classes are in general disjoint if an entity was part of both classes at different points in time (see Example 6). Moreover, there might be scenarios in which it is necessary to define a constraint that applies to all instances that were part of a class at any point in time. If the dataset is incomplete, e.g., not all facts have a temporal annotation, it is necessary to use the non-temporal predicate. Thus, statements that are temporal annotated must be transformed to the non-temporal form.

**Example 6** *An entity is assigned to different classes at different points in time.*

```
quad(John, rdf:type, Child, t1)
quad(John, rdf:type, GrownUp, t2)
```

*We defined a rule that infers the following non-temporal statements:*

```
triple(John, rdf:type, Child)
triple(John, rdf:type, GrownUp)
```

*This is not an issue as long as we do not define a constraint that states that the classes `Child` and `GrownUp` are disjoint when we ignore the temporal information. However, it is possible to define a constraint that states an entity cannot be a `Child` and a `GrownUp` at the same time.*

Moreover, we add additional rules which are required to benefit from the RDFS entailment rules when carrying out temporal reasoning. The rules `rdfs2`, `rdfs3`, `rdfs7` and `rdfs9` are the only rules that infer statements that should be temporal annotated if the initial statement is also temporal annotated. Hence, we define the following rules:

$$\begin{aligned} \text{triple}(p, \text{"rdfs : domain"}, c) \quad \wedge \quad \text{quad}(s, p, o, i) &\Rightarrow \text{quad}(s, \text{"rdfs : type"}, c, i) \\ \text{triple}(p, \text{"rdfs : range"}, c) \quad \wedge \quad \text{quad}(s, p, o, i) &\Rightarrow \text{quad}(o, \text{"rdfs : type"}, c, i) \\ \text{triple}(a, \text{"rdfs : subPropertyOf"}, b) \quad \wedge \quad \text{quad}(u, a, y, i) &\Rightarrow \text{quad}(u, b, y, i) \\ \text{triple}(a, \text{"rdfs : subclassOf"}, b) \quad \wedge \quad \text{quad}(u, \text{"rdfs : type"}, a, i) &\Rightarrow \text{quad}(u, \text{"rdfs : type"}, b, i) \end{aligned}$$

The other entailment rules infer statements having the predicate `rdfs : subPropertyOf` (`rdfs5`, `rdfs6`, `rdfs12`) or `rdfs : subclassOf` (`rdfs8`, `rdfs10`, `rdfs11`, `rdfs13`). We do not add adjusted rules for these entailment rules as we do not annotate statements having these predicates with temporal information. We do also not

add a temporal component to the rules `rdf1`, `rdfs1`, `rdfs4a` and `rdfs4b` as they infer statements that assign resources to standard classes contained in the RDF(S) vocabulary (e.g., `rdfs:Resource`). It is shown that the RDF(S) entailment rules are complete [Hayes, 2004]. Thus, the rule set that we consider for our approach is also complete as we add all constraints but the rule that considers if a literal is a well typed XML literal. We could have added another predicate `XMLLiteral(r)` to implement the respective rule but that would not lead to any advantages with respect to the goal of our approach. So, we introduced a Markov Logic based formalism that enables RDF(S) reasoning by applying the defined entailment rules using a Markov Logic solver.

### 3.3.2 Temporal Constraints

We extend our approach with predicates representing the relations of Allen’s interval algebra (see Section 2.1). The predicates can be used to define temporal constraints and rely on ground values that express the relations between the intervals occurring in the dataset. The relations need to be calculated before instantiating the Markov Logic Network because comparison operators are not part of Markov Logic. We decided to transform time points to intervals with the same lower bound and upper bound as the relations of Allen’s interval algebra are sufficient to express the constraints that rely on time points. We justify this design choice and elaborate other possibilities in Section 4.3. However, the defined predicates are the following:

```
*tBefore(Interval, Interval)
*tMeets(Interval, Interval)
*tOverlaps(Interval, Interval)
*tStarts(Interval, Interval)
*tDuring(Interval, Interval)
*tFinishes(Interval, Interval)
*tEqual(Interval, Interval)
```

We define the predicates as observed predicates as we instantiate the Markov Logic Network with all relations that hold between the intervals occurring in the dataset. Hence, we can assume that the ground values of the interval relations are correct and complete. This enables a fast evaluation of the temporal constraints by the Markov Logic solver. Moreover, we benefit from exploiting the characteristics of observed predicates which allow us to assume that only the temporal relation that is initially passed to the Markov Logic solver holds between two intervals. Hence, we are not required to define constraints that assure the pairwise disjointness of the temporal relations. It is also noteworthy that we decouple the intervals and their

relations from the statements. Thus, the identifier of an interval represents a specific combination of a lower bound and an upper bound and is not dependent on a specific statement. The defined predicates can be used to define constraints and to carry out temporal reasoning as shown in Example 7. Temporal constraints are always domain-specific constraints and need to be defined for every use case. The general idea is to define constraints that express which temporal relation holds or does not hold between related statements. If statements do not match the restriction of the rule it is necessary to resolve the conflict by removing a statement. Due to the pairwise disjointness of the temporal predicates it is ensured that only one relation holds.

**Example 7** *The temporal predicates can be used to define temporal constraints. We define a constraint that ensures that someone is only born in one specific year:*

```
!quad(p,"birthYear",l1,i1) v !quad(p,"birthYear",l2,i2) v tEqual(i1,i2)
```

*The constraint uses the predicate `tEqual` to express that two statements declaring the birth year of an entity must be annotated with the same interval. Consider the following ground values:*

```
0.15 quad(John, birthYear, 1951, interval1)
0.95 quad(John, birthYear, 1981, interval2)
tBefore(interval1, interval2)
```

*They lead to a conflict as the birth years assigned to John are not equal. In this case, the reasoner will remove the statement with lower weight as only one statement can remain in the dataset. The higher weight can be interpreted as a higher confidence that a statement is true.*

**Grouping of interval relations.** The relations of Allen's interval algebra are pairwise disjoint and jointly exhaustive. Hence, there is exactly one relation that holds between two intervals. However, the relations might be too fine-grained for some use cases. Therefore, we provide a possibility to introduce new temporal relations that group the existing relations. We introduce a property `temporalProperty` whose sub-properties will be converted to predicates having two elements of type `Interval`. The original interval relations can be mapped to this predicate by defining the respective constraints (see Example 8).

**Example 8** *In order to define a more general interval relation, we introduce a new temporal property ends:*

```
ends rdfs:subPropertyOf temporalProperty.
```

*Based on this axiom, we create the following predicate:*

```
ends(Interval, Interval)
```

*Moreover, we need to map some of initial interval relations to this predicate by defining constraints. For instance, the following formulas ensure that `ends` groups `tEqual` and `tFinishes`:*

```
tEqual(i1, i2) => ends(i1, i2)
tFinishes(i1, i2) => ends(i1, i2)
```

*Finally, `ends` can be used to define temporal constraints.*

### 3.4 Optimizations

We introduced a basic model that can be used to carry out temporal reasoning in Section 3.2 and Section 3.3. In this section, we extend and adjust the Markov Logic model in order to optimize it for specific use cases.

**Observed predicates.** We introduced predicate symbols that can be used to express any type of statement that we target with our approach. However, the characteristics of Markov Logic allow us to extend the model in order to enable more efficient reasoning. So far, the introduced predicates (i.e., `triple` and `quad`) are hidden predicates. Markov Logic solvers allow also defining observed predicates. The advantage of using such predicates is that only the ground axioms that are passed to the solver are considered as true while all other possible groundings are considered as false statements (closed world assumption). In consequence of that, the reasoner can disregard the ground formula if the ground predicate is true or reduce the number of literals in the formula if the literal is false. Hence, the solver can faster decide if a formula is true or reduce its complexity (i.e., number of literals). We add the observed predicates `tripleO` and `quadO` to the model:

```
*tripleO(r, r, r)
*quadO(r, r, r, Interval)
```

Moreover, we define formulas that infer the respective hidden predicates:

```
!quadO(s, p, o, i) v quad(s, p, o, i).
!tripleO(s, p, o) v triple(s, p, o).
```

So, we added two observed predicates as well as two constraints in order to infer the unobserved counterparts (see Appendix A.1). Adding observed predicates for the statements enhances our approach as there are scenarios in which it makes sense to define constraints that have literals which only rely on the statements provided to the reasoner. For instance, it is possible to have a knowledge base in which the types of the entities are correct and complete. Thus, it would be beneficial to express the respective statements using observed predicates. Furthermore, the observed predicate must be used in the definition of the constraints in order to exploit the advantage of defining observed statements. So, the reasoner is not only able to evaluate the respective ground formulas faster but it is also ensured that the reasoner does not infer additional type assertions. The introduced rules that infer the respective hidden predicates ( $\text{quadO} \Rightarrow \text{quad}$ ) ensure that the RDF(S) entailment rules also apply for the observed statements. Also note that this extension has no influence on the performance if it is not used as all possible groundings of an observed predicate are considered as false if they are not passed to the solver.

**Types.** While our goal is to develop a temporal RDF(S) reasoner there might also be use cases that do not require the RDF(S) reasoning feature. Hence, we suggest excluding the RDF(S) entailment rules in order to improve the performance. Moreover, it is possible to drop the requirement that every element of a statement (i.e., subject, predicate and object) has the same type which leads to the following predicates:

```
triple(r1, r2, r3)
quad(r1, r2, r3, Interval)
*tripleO(r1, r2, r3)
*quadO(r1, r2, r3, Interval)
```

This reduces the number of possible groundings which also leads to a better performance as the number of possible worlds that are considered during the computation of the MAP state is reduced. However, this model can only be applied when the subjects, predicates and objects of the statements contained in the dataset are disjoint sets. Hence, the underlying graph must only contain graphs that are stars.

In this section, we defined observed predicates and outlined that the RDF(S) entailment rules are not always required which allows defining the predicates using differently typed elements. Overall, we have three different configurations that allow more efficient reasoning by disabling features in certain use cases:

- The most extensive model supports all features including RDF(S) reasoning.

- The next possibility is to exclude the RDF(S) entailment rules.
- The last model excludes the RDF(S) entailment rules and relies on predicates whose elements are not of the same type (e.g., `triple(r1, r2, r3)` instead of `triple(r, r, r)`).

### 3.5 Discussion

We proposed a formalism to carry out temporal reasoning for RDF(S) knowledge bases in this chapter. While the focus of this work is to develop an approach that solves (temporal) conflicts in a knowledge base, it is also possible to infer new statements. So far, we restricted the approach to RDF(S) but it is also possible to enhance it with some features of OWL2 [Motik et al., 2012] in order to extend the expressiveness. We discuss those aspects in the following. We also elaborate on the applications of weighted (soft) constraints as they are not as intuitive as hard constraints.

**Data cleansing vs. inferring new statements.** The primary goal of our approach is to detect and to resolve inconsistencies in a knowledge base. However, the defined rules do also infer new statements. Hence, the output of the reasoner (MAP state) contains some or all (if the dataset was already consistent) statements of the original dataset as well as inferred statements. If we focus on data cleansing we remove the statements from the dataset that are not part of the MAP state and do not add new statements to the dataset. In contrast to this, it is also possible to enhance the original dataset with the inferred statements. However, when following the latter approach it might be necessary to extend the constraints by adding formulas that assign a small negative weight to any ground predicate in order to limit the output to the statements that can be inferred by applying the rules:

```
-0.001 triple(s, p, o)
-0.001 quad(s, p, o, i)
```

Otherwise it is possible that MAP state contains any ground predicates that do not violate any constraints. However, this does not apply to rockIt as it uses a more efficient grounding technique that already fulfills this requirement.

**Weighted constraints.** Our approach supports two types of constraints. Hard constraints must always hold while soft constraints can be violated. However, using soft constraints makes only sense when this leads to a conflict that can be solved considering the involved statements and other constraints. Constraints with weights

might act like hard constraints if they do not lead to inconsistencies in the dataset. This can be problematic when one considers adding the inferred statements to the knowledge base (see Example 9).

**Example 9** *One might express the assumption that 10% of the persons that attended a university have the degree Ph.D. with the following constraint:*

```
0.1 !triple(p, "attended", u) v triple(p, "hasDegree", "Ph.D.")
```

*This rule is problematic as it assigns the degree Ph.D. to all persons who attended a university as this does not violate any constraint. Hence, it might be better to define a constraint that states that 90% have not received the degree:*

```
0.9 !triple(p, "attended", u) v !triple(p, "hasDegree", "Ph.D.")
```

*Thus, statements that express that a person has the degree only remain in the dataset if it is justified by other evidence. However, all statements that match the pattern `triple(p, "hasDegree", "Ph.D.")` must have a higher weight than 0.9 (barring other statements) in order to be retained in the dataset. As this weight is relatively high, it is worth to consider reducing the weight of the constraint.*

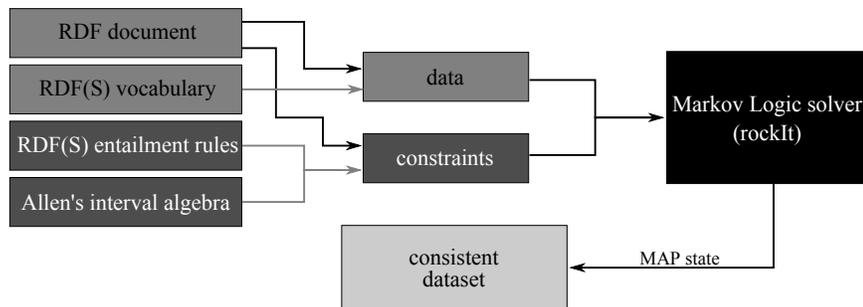
**Possible extensions.** Currently, we only include the RDF(S) entailment rules in order to give our approach a basic logical expressiveness. However, RDF(S) lacks some features that may be required in some situations. The most noticeable limitation is that it does not support to express disjointness among entities or classes. Thus, we cannot express all constraints that might be required to detect all inconsistencies in a knowledge base if we only rely on the RDF(S) entailment rules. However, the Web Ontology Language [Motik et al., 2012] defines a vocabulary that can partially be used for RDF datasets. For instance, it allows expressing disjointness and equality between the entities. These and additional features are required in some use cases. We do not include them but the rule set of our approach can be extended on demand. Hence, we only provide the least common rule set that is sufficient for many use cases.

## Chapter 4

# Implementation

In this chapter, we present some details of the implementation of the approach introduced in Chapter 3. We start with a brief description of the whole process in order to give an overview on the topics which we will treat in the following. We describe how we model the different types of statements (see Section 4.1) and the different types of constraints (see Section 4.2) in the first two sections. In the last section (Section 4.3), we discuss different models to carry out temporal reasoning in Markov Logic and justify which model fits best to our approach.

We developed an application that implements the formalism presented in the previous chapter. The workflow of this application is illustrated in Figure 4.1. The input of the application is a RDF document containing statements and domain-specific constraints (and rules). Moreover, it is possible to enable or to disable the RDF(S) reasoning feature and to decide if the inferred statements should be added to the cleansed dataset. In general, the application has two different parts: In the first part, it converts the RDF input to Markov Logic. In second part, it transforms



**Figure 4.1:** Overview on the workflow of our application.

the output of the solver back to a RDF document. So, we extract all statements contained in the RDF document and convert them to ground predicates for the Markov Logic Network. We also extract domain-specific constraints and transform them to formulas that will be considered by the Markov Logic solver. Depending on the selected reasoning method, we include the RDF(S) entailment rules as well as the predicates of Allen’s interval algebra and the RDF(S) vocabulary. This gives us a set of ground predicates, i.e., the statements, and a set of formulas, i.e., constraints, which can be processed by a Markov Logic solver. We use `rockIt` to compute the MAP state in order to resolve the inconsistencies in the dataset. Hence, its output is a set of ground predicates that can be transformed to a consistent dataset. Depending on the setting, we add or do not add the inferred statements to the final RDF document.

## 4.1 Statements

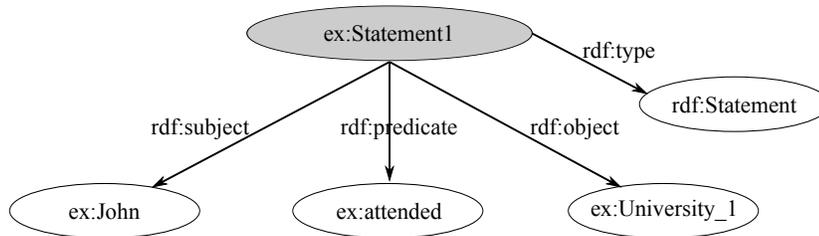
In this section, we explain how we annotate RDF statements in order to transform them to ground predicates in Markov Logic. We support standard statements (RDF triples) and temporal annotated statements. Each type of statement can also be weighted or observed (see Appendix A.1). Thus, the used annotation concept has to provide a possibility to distinguish between the following characteristics of a statement:

**Triple or Quad:** The most obvious distinguishing feature is if the statement is temporal annotated (`quad`) or not (`triple`). A temporal statement needs to be annotated with an interval, i.e., a lower bound and an upper bound, which indicates when the statement holds.

**Standard, Weighted or Observed:** The second level of features targets the influence of the statements on the result. While the standard statements (`triple`, `quad`) and the observed statements (`tripleO`, `quadO`) are part of the MAP state (if one consistent dataset exists) the weighted statements (`tripleW`, `quadW`)<sup>1</sup> can be discarded if they violate the defined constraints. Observed statements have the same characteristics as standard statements but allow defining constraints that can be processed more efficiently. So, we need a feature that indicates if a statement is observed. We also need a feature that expresses the weight of a statement.

---

<sup>1</sup>We need to define helper predicates for the weighted statements as `rockIt` does not allow annotating statements with a weight. It is necessary to define constraints that implicitly assign the weights to the respective standard form of the predicates.



**Figure 4.2:** Statements: Reification.

Our approach relies on the RDF data model. Hence, the basis of all types of statements is a standard RDF triple. In order to annotate it with additional information, e.g., a weight or an interval, we use the concept of reification. Hence, we create an additional node in the graph that is dedicated to a RDF triple. This node can be used to annotate the statement. We use the standard RDF reification vocabulary despite the fact that it was introduced to annotate a statement with provenance information. We could have introduced a new vocabulary for this purpose but we decided against it because we use the API Apache Jena<sup>2</sup> that already provides methods to process reified statements using the RDF vocabulary.

However, we transform all statements that are not reified to Markov Logic using the predicate `triple`. For all other types of statements, it is necessary to annotate additional information to a node representing the statement (see Figure 4.2). We use the following properties for this purpose (see Figure 4.3 and Example 10):

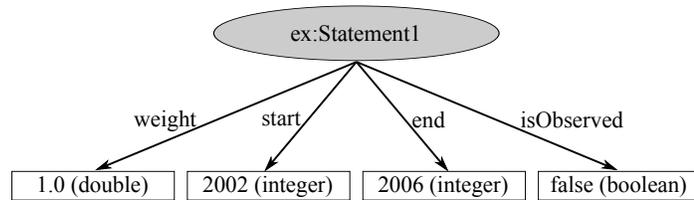
**start / end** These two properties allow specifying the upper bound and lower bound of a temporal interval that expresses when the statement is true. Only if these properties are set, we transform the statement to a `quad`, `quadW` or `quadO`.

**weight** This property assigns a weight to the statement. We transform all weighted statements to Markov Logic using the predicate `tripleW` or `quadW`.

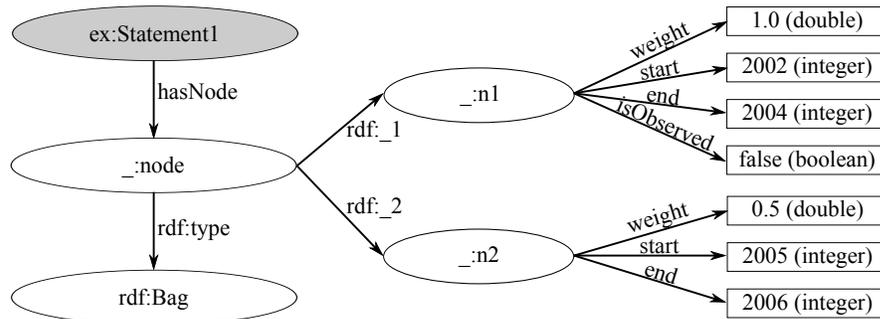
**isObserved** This property can be used to indicate that a statement is observed. We transform the respective statements to Markov Logic using the predicate `tripleO` or `quadO`.

All of those properties are optional. However, some combinations are not allowed. For instance, a statement cannot be observed (`isObserved = true`) and have a weight at the same time. We support only integer values to define the border

<sup>2</sup><https://jena.apache.org/>



**Figure 4.3:** Statements: Direct Annotation.



**Figure 4.4:** Statements: A statement that is annotated with multiple intervals.

points of an interval. Those values can be years but it is also possible to map a date or a timestamp to an integer value. Hence, there is no limitation as long as a document contains only intervals using the same unit (e.g., no mixture of years and timestamps). Moreover, it is possible that a reified triple holds during different periods of time. Therefore, it is possible to introduce an intermediate node of type `rdf:Bag` to which one can add information about different intervals (see Figure 4.4 and Example 11). The bag is added to the node identifying the statement via the property `hasNode`. The information about the different intervals needs to be assigned to disjoint (blank) nodes using the introduced vocabulary. The (blank) nodes need to be added to the bag which is connected to the node representing the respective statement.

**Example 10** *We express a weighted temporal statement in RDF.*

```
John attended University_1.

[ rdf:type      rdf:Statement ;
  rdf:subject  John ;
  rdf:predicate attended ;
  rdf:object   University_1 ;
  weight       "1.0"^^xsd:decimal ;
  start        "2002";
```

```

    end          "2006"
  ].

```

*These RDF triples will be converted to the following ground predicate:*

```
quadW("John", "attended", "University_1", [2002, 2006], 1.0)
```

**Example 11** *We annotate a single RDF triple with multiple intervals in order to define multiple temporal annotated statements.*

```

John attended University_1.

[ rdf:type      rdf:Statement ;
  rdf:subject   John ;
  rdf:predicate attended ;
  rdf:object    University_1 ;
  hasNode       [ a rdf:Bag;
                  rdf:_1 [weight "1.0"^^xsd:decimal ;
                           start  "2002";
                           end    "2004"];
                  rdf:_2 [weight "0.5"^^xsd:decimal ;
                           start  "2005";
                           end    "2006"]
                ]
].

```

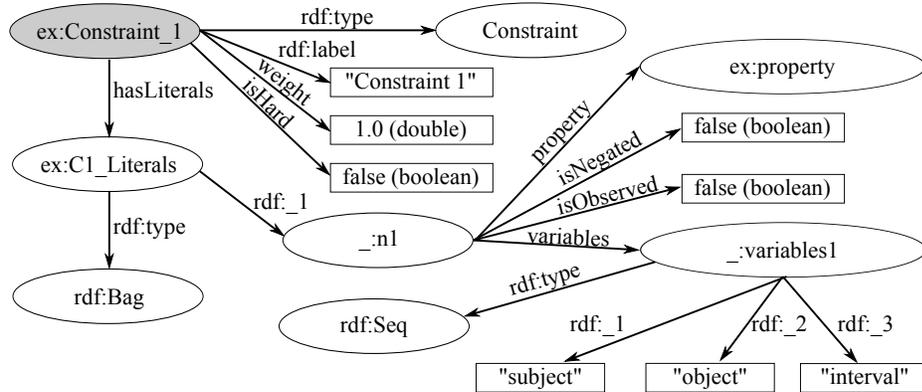
*These RDF triples will be converted to the following ground predicates:*

```
quadW("John", "attended", "University_1", [2002, 2004], 1.0)
quadW("John", "attended", "University_1", [2005, 2006], 0.5)
```

## 4.2 Constraints & Rules

We also decided to model the domain-specific constraints (and rules) in RDF. Hence, it is possible to provide one document that contains the statements as well as the constraints. In general, Markov Logic supports soft constraints and hard constraints. Soft constraints are weighted and it is possible that they are violated in the MAP state. Contrary, hard constraints cannot be violated in the MAP state. The chosen model is aligned to the input of the Markov Logic solver. Thus, this model is suitable to express constraints and rules. A constraint is modeled as a disjunction of literals. Each literal has a property (i.e., the predicate of a RDF triple), a list of variables and can be negated as well as observed.

We illustrate how we construct a constraint in Figure 4.5. A constraint is identified by a node of type `Constraint`. It has a label (`rdfs:label`), a weight (`weight`) as well as the information if it is a hard constraints or a soft constraint (`isHard`). However, the main part of the constraints is a bag of literals



**Figure 4.5:** Constraints: RDF Model

(`hasLiterals`). We convert a constraint to a formula based on the listed basic features and the disjunction of the defined literals.

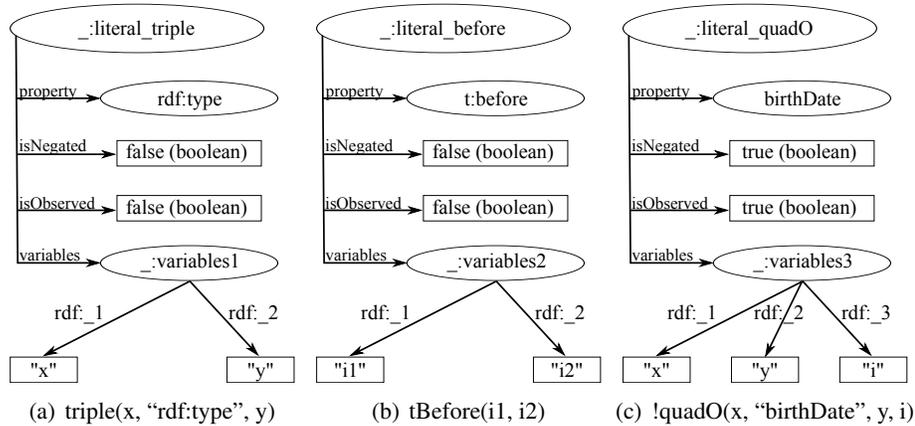
Each literal of a constraint is identified by a separate node that aggregates the information. So, it describes if the literal is negated (`isNegated`) or observed (`isObserved`). The latter feature indicates if the literal needs to get transformed to a `triple` (`quad`) or `tripleO` (`quadO`). Moreover, each literal has a `property` (predicate of a statement) and a sequence of `variables`. The literal gets transformed to a `quadO` if three variables are listed (see Figure 4.6(c)). If a literal has two variables we only transform it to a `tripleO` if the property of the literal is not temporal (see Figure 4.6(a)). If the property is temporal we transform it to the respective Markov Logic predicate that expresses the temporal relation as both variables need to be mapped to intervals (see Figure 4.6(b)). We maintain a list of temporal properties for this purpose. Example 12 illustrates how combine the introduced features in order to define a constraint.

**Example 12** *The following constraint expresses that the `birthDate` of an entity cannot be annotated with the same interval as the `deathDate`.*

```

C1  rdf:type      Constraint;
    rdfs:label   "birthdate and deathdate cannot be annotated
                with the same interval";
    isHard      "true"^^xsd:boolean;
    hasLiterals [
      a rdf:Bag;
      rdf:_1 [ isNegated "true"^^xsd:boolean;
                isObserved "true"^^xsd:boolean;
                property birthDate ;
                variables [ a rdf:Seq;
                            rdf:_1 "x"; rdf:_2 "y"; rdf:_3 "i"]
              ]
    ]

```



**Figure 4.6:** Constraints: Declaration of the different types of literals.

```

    ];
    rdf:_2 [ isNegated "true"^^xsd:boolean;
            property deathDate;
            variables [ a rdf:Seg;
                      rdf:_1 "x"; rdf:_2 "z"; rdf:_3 "i" ]
            ]
  ].

```

We convert this constraint to the following formula:

$$!quadO(x, "birthDate", y, i) \vee !quad(x, "deathDate", z, i).$$

It is a hard constraint that cannot be violated in the cleansed dataset.

### 4.3 Interval Relations

In this section, we explain how we process the intervals that are annotated to statements and give reasons for the chosen implementation. We express the temporal relation that holds between two statements using an interval relation of Allen's interval algebra (see Section 2.1). In general, a temporal statement is annotated with a start point and an end point which set the border points of an interval. Currently, we support only values of type integer but the following observations hold for any comparable data type. We considered three models to process the intervals and to compute the relations:

**M1** We directly assign to each disjoint interval that occurs in the dataset an identifier and compute all interval relations in JAVA. The input of the Markov Logic Network are ground predicates expressing the interval relations.

- M2** We create all possible intervals, i.e., all possible combinations of the border points (start point  $\leq$  end point) of the given intervals and compute the interval relations in JAVA. The input of the Markov Logic Network are ground predicates expressing the interval relations.
- M3** We compute the relations between all border points of the intervals in JAVA using the point algebra and compute the interval relations in Markov Logic.

The major difference between the Model **M1** and the other options is that Model **M1** is restricted to the given intervals, i.e., intervals that are annotated to statements in the input document. Hence, our application is not able to infer additional intervals (see Example 13) using this option. However, we will see that only Model **M1** is suitable for our approach.

**Example 13** *If we consider a knowledge base that contains the following statements:*

```
John attended University {[2009, 2012]}
John attended University {[2012, 2014]}
```

*We can compute the overarching interval [2009, 2014] that comprises (overarches) the other intervals. Using this interval, we can infer the following statement:*

```
John attended University {[2009, 2014]}
```

*This is only possible if our application computes additional intervals and does not only rely on the intervals that are provided in the input dataset.*

However, computing additional intervals increases the problem size (see Example 14). As we are not able to define constraints that limit the number of inferred intervals, e.g., compute only intervals that will be used to annotate statements, our application would compute all possible combination.

**Example 14** *We can compute 10 intervals if the input dataset contains two intervals with four different border points. The intervals [1, 2] and [3, 4] have the following border points: 1, 2, 3, 4. Hence, there exist 10 intervals having these border points: [1, 1], [2, 2], [3, 3], [4, 4], [1, 2], [1, 3], [1, 4], [2, 3], [2, 4] and [3, 4].*

We illustrate the effect on the problem size depending on the number of disjoint intervals ( $n$ ) for the proposed models in Table 4.1. Model **M1** computes the relations between all given intervals. This leads to  $\binom{n}{2} + n = \frac{1}{2}n \cdot (n - 1) + n$  interval relations as the relations are jointly exhaustive and pairwise disjoint. In this

intervals (statements)	border points	possible intervals	Model M1 interval relations	Model M2 interval relations	Model M3 number relations
2	4	10	3	55	16
3	6	21	6	231	36
4	8	36	10	666	64
5	10	55	15	1,540	100
10	20	210	55	22,155	400
25	50	1,275	325	813,450	2,500
50	100	5,050	1,275	12,753,775	10,000
100	200	20,100	5,050	202,015,050	40,000
500	1,000	500,500	125,250	125,250,375,250	1,000,000
1,000	2,000	2,001,000	500,500	2,002,001,500,500	4,000,000
5,000	10,000	50,005,000	12,502,500	1,250,250,037,502,500	100,000,000
10,000	20,000	200,010,000	50,005,000	20,002,000,150,005,000	400,000,000
25,000	50,000	1,250,025,000	312,512,500	781,281,250,937,512,000	2,500,000,000
50,000	100,000	5,000,050,000	1,250,025,000	12,500,250,003,750,000,000	10,000,000,000
100,000	200,000	20,000,100,000	5,000,050,000	200,002,000,015,000,000,000	40,000,000,000
250,000	500,000	125,000,250,000	31,250,125,000	7,812,531,250,093,750,000,000	250,000,000,000
500,000	1,000,000	500,000,500,000	125,000,250,000	125,000,250,000,375,000,000,000	1,000,000,000,000
1,000,000	2,000,000	2,000,001,000,000	500,000,500,000	2,000,002,000,001,500,000,000,000	4,000,000,000,000

**Table 4.1:** Number of interval relations depending on the different models. Having  $n$  different intervals leads to  $\frac{1}{2}n \cdot (n - 1) + n$  interval relations.

case, we interpret  $n$  as the number of statements of the dataset and use the big  $O$  notation to express the upper bounds. This is possible as the number of statements is an upper limit for the number of (disjoint) intervals. However, there can be fewer intervals as it happens that some statements are not annotated with an interval or that different statements are annotated with the same interval. Model **M2** works like the first model but computes all intervals that exist for the given border points. There are  $x = \mathcal{O}(2n)$  different numbers as one interval has up to 2 different border points and all intervals may have different border points. The extracted border points can be used to compute  $y = \binom{x}{2} + x$  intervals and thus  $z = \binom{y}{2} + y$  interval relations (see Example 15).

**Example 15** *If the initial dataset contains 1,000 intervals then there are up to 2,000 different numbers. Hence, we can compute 2,001,000 intervals. This leads to 2,002,001,500,500  $\approx 2.0E + 12$  interval relations.*

Example 15 illustrates that the Model **M2** is not feasible as the interval relations have a huge influence on the problem size. Nevertheless, we continue with Model **M3** as this model allows us to infer the interval relations in Markov Logic. The number of the resulting intervals and interval relations is equal to Model **M2**. However, the input of the Markov Logic Network are the relations among the border points using the point algebra and. In total, we need to define  $\mathcal{O}(4n^2)$  ground predicates expressing the relations of the border points (see Table 4.1). So, the input of the Markov Logic Network is smaller, i.e., less initial ground predicates, while it is also much more complex as we will see in the following.

**Model M1 + M2.** The models **M1** and **M2** and rely on the same concept. We extract the intervals from all temporal annotated statements and map each interval to a unique identifier. However, for model **M2** we extract the border points and create all possible intervals (see Example 14). In the next step step, we calculate all interval relations (see Algorithm 1) according to their characteristics which are illustrated in Table 2.2 in Chapter 2.

The implemented algorithm returns the name of the interval relation that holds between two specific intervals. The respective Markov Logic Network (see Listing 4.1) contains one observed predicate for each interval relation. We do not need any additional constraint as we can directly use the observed predicates in the constraints. In order to initialize the model, we use the identifier of the intervals that we generated in first step. Example 16 gives a brief overview on all steps taken by Model **M1**.

---

**Algorithm 1** Calculates the relation between two intervals.

---

**Require:**  $i1.lowerBound \leq i1.upperBound$

**Require:**  $i2.lowerBound \leq i2.upperBound$

```
1: function CALCULATEINTERVARELATION(i1, i2)
2:   if  $i1.lowerBound = i2.lowerBound \wedge i1.upperBound = i2.upperBound$  then
3:     return tEqual
4:   end if
5:   if  $i1.lowerBound < i2.lowerBound$  then
6:     if  $i1.upperBound < i2.lowerBound$  then
7:       return tBefore
8:     end if
9:     if  $i1.upperBound = i2.lowerBound$  then
10:      return tMeets
11:    end if
12:    if  $i1.upperBound > i2.lowerBound \wedge i1.upperBound < i2.upperBound$  then
13:      return tOverlaps
14:    end if
15:  else if  $i1.lowerBound > i2.lowerBound$  then
16:    if  $i1.upperBound < i2.upperBound$  then
17:      return tDuring
18:    end if
19:    if  $i1.upperBound = i2.upperBound$  then
20:      return tFinishes
21:    end if
22:  end if
23:  if  $i1.lowerBound = i2.lowerBound \wedge i1.upperBound < i2.upperBound$  then
24:    return tStarts
25:  end if
26: end function
```

---

```
// interval relations
*tBefore(Interval, Interval)
*tMeets(Interval, Interval)
*tOverlaps(Interval, Interval)
*tStarts(Interval, Interval)
*tDuring(Interval, Interval)
*tFinishes(Interval, Interval)
*tEqual(Interval, Interval)
```

**Listing 4.1:** Markov Logic Network for M1 & M2.

**Example 16** *Given the following statements:*

```
John attended University {[2009, 2012]}
John attended University {[2012, 2014]}
```

*We extract the intervals [2009, 2012], [2012, 2014] and map them to identifiers:*

```
Interval1 := [2009, 2012]
Interval2 := [2012, 2014]
```

*Using Algorithm 1, we get the interval relation  $tMeets$  and initialize the Markov Logic Network with the following ground predicate:*

```
tMeets(Interval1, Interval2)
```

*After the reasoner terminated, we are able to retrieve the actual intervals as we keep the map during the calculation of the MAP state.*

**Model M3.** The last model (**M3**) uses the Markov Logic reasoner to infer all possible interval relations. However, we need to compute the relations of the border points that can be expressed with the point algebra in JAVA. Hence, the Markov Logic Network contains three observed predicates to model the point algebra relations and seven hidden predicates to model the interval relations (see Listing 4.2).

The first step is to collect all border points of the given intervals and to calculate the relations. This step is illustrated in Example 17:

**Example 17** *Given the following statements:*

```
John attended University {[2009, 2012]}
John attended University {[2012, 2014]}
```

*We extract the border points 2009, 2012, 2014 and map them to identifiers:*

```

// point algebra relations
*tpLarger(n,n)
*tpEquals(n,n)
*tpSmaller(n,n)

// interval relations
tBefore(n,n,n,n)
tMeets(n,n,n,n)
tOverlaps(n,n,n,n)
tStarts(n,n,n,n)
tDuring(n,n,n,n)
tFinishes(n,n,n,n)
tEqual(n,n,n,n)

```

**Listing 4.2:** Markov Logic Network for M3: Predicates

```

// constraints to infer intervals
X v !tpEquals(s1,s2) v !tpEquals(e1,e2) v tEqual(s1,e1,s2,e2) .

X v !tpSmaller(e1,s2) v tBefore(s1,e1,s2,e2) .

X v !tpSmaller(s1,s2) v !tpEquals(e1,s2) v !tpSmaller(e1,e2) v
tMeets(s1,e1,s2,e2) .

X v !tpSmaller(s1,s2) v !tpLarger(e1,s2) v !tpSmaller(e1,e2) v
tOverlaps(s1,e1,s2,e2) .

X v !tpLarger(s1,s2) v !tpLarger(e1,s2) v !tpSmaller(e1,e2) v
tDuring(s1,e1,s2,e2) .

X v !tpLarger(s1,s2) v !tpEquals(e1,e2) v tFinishes(s1,e1,s2,e2) .
X v !tpEquals(s1,s2) v !tpSmaller(e1,e2) v tStarts(s1,e1,s2,e2) .

// restrictions: pairwise disjointness (48 constraints)
!tBefore(s1,e1,s2,e2) v !tBefore(s2,e2,s1,e1) .
!tBefore(s1,e1,s2,e2) v !tMeets(s1,e1,s2,e2) .
!tBefore(s1,e1,s2,e2) v !tMeets(s2,e2,s1,e1) .
...
!tFinishes(s1,e1,s2,e2) v !tEqual(s1,e1,s2,e2) .
!tFinishes(s1,e1,s2,e2) v !tEqual(s2,e2,s1,e1) .

```

**Listing 4.3:** Markov Logic Network for M3: Constraints. (X = tpLarger(s1,e1) v tpLarger(s2,e2))

```
n1 := 2009,    n2 := 2012,    n3 := 2014
```

*We compute the point algebra relations and initialize the Markov Logic Network with the following ground predicates:*

```
tpEquals(n1, n1)    tpEquals(n2, n2)    tpEquals(n3, n3)
tpSmaller(n1, n2)  tpSmaller(n1, n3)  tpSmaller(n2, n3)
tpLarger(n2, n1)   tpLarger(n3, n1)   tpLarger(n3, n2)
```

Based on the calculated point algebra relations, we define rules that infer all possible interval relations (see Listing 4.3). The constraints are comparable to the algorithm that we use for the other models (see Algorithm 1). However, we need to add the restriction that for an interval  $[s, e]$  holds that  $s \leq e$  which is equivalent to  $\text{!tpLarger}(s, e)$ . Moreover, we need constraints that ensure that only one relation holds between two intervals (pairwise disjointness). This is necessary as the respective predicates are hidden. Hence, we need to define constraints that prevent the reasoner from returning arbitrary groundings for the predicates that do violate the characteristics of the relations. We continue Example 17 in order to illustrate the output of the Markov Logic solver (see Example 18).

**Example 18** *We showed in Example 17 how we compute the input of the Markov Logic Network for Model M3. The example ended with the following ground predicates that express the relations of the border points:*

```
tpEquals(n1, n1)    tpEquals(n2, n2)    tpEquals(n3, n3)
tpSmaller(n1, n2)  tpSmaller(n1, n3)  tpSmaller(n2, n3)
tpLarger(n2, n1)   tpLarger(n3, n1)   tpLarger(n3, n2)
```

*By applying the constraints listed in Listing 4.3, we get these ground predicates:*

```
tEqual(n1, n1, n1, n1)  tEqual(n1, n2, n1, n2)  tEqual(n1, n3, n1, n3)
tEqual(n2, n2, n2, n2)  tEqual(n2, n3, n2, n3)  tEqual(n3, n3, n3, n3)
tBefore(n1, n1, n2, n2) tBefore(n1, n1, n2, n3) tBefore(n1, n1, n3, n3)
tBefore(n1, n2, n3, n3) tBefore(n2, n2, n3, n3) tDuring(n2, n2, n1, n3)
tFinishes(n2, n2, n1, n2) tFinishes(n2, n3, n1, n3) tFinishes(n3, n3, n1, n3)
tFinishes(n3, n3, n2, n3) tMeets(n1, n2, n2, n3)  tStarts(n1, n1, n1, n2)
tStarts(n1, n1, n1, n3)  tStarts(n1, n2, n1, n3)  tStarts(n2, n2, n2, n3)
```

*These predicates implicitly contain the following intervals:*

```
[n1, n1], [n2, n2], [n3, n3],
[n1, n2], [n1, n3], [n2, n3]
```

*which can be mapped to:*

```

// observed predicates
*tBefore(Interval, Interval)
*tMeets(Interval, Interval)
...
// hidden predicates
*xBefore(Interval, Interval)
*xMeets(Interval, Interval)
...
// constraints
!tBefore(x, y) v xBefore(x, y).
!tMeets(x, y) v xMeets(x, y).
...

```

**Listing 4.4:** Markov Logic Network used to test Model M1 & M2.

```

[2009, 2009], [2012, 2012], [2014, 2014],
[2009, 2012], [2009, 2014], [2012, 2014]

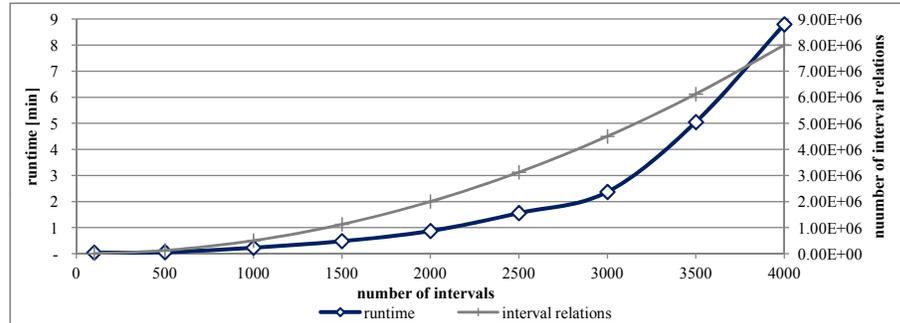
```

*It is possible to define 6 intervals given 3 border points ( $\binom{3}{2} + 3 = 6$ ). Hence, Model **M3** infers all valid intervals.*

We see that all three models rely on computations outside of Markov Logic as number comparison is not supported by Markov Logic. We do also not need to consider the composition table of Allen’s interval algebra (see Table 2.3 in Chapter 2) as the relations between all intervals can be directly computed. However, the constraints given in this table are implicitly considered and not violated. Moreover, the intervals and their relations can be modeled as observed predicates as we decouple them from the statements. This means that the relations of an interval to the other intervals are independent from the correctness of the statements from which the interval was extracted. We benefit from this observation as it allowed us to define a compact Markov Logic Network.

**Experiments.** We evaluated the feasibility of the models **M1** and **M2** as both depend on the same Markov Logic Network. The only difference between the models is that the input of the Markov Logic Network contains more interval relations if we apply Model **M2** (see Table 4.1). However, we focus on the efficiency of the model which depends on the problem size which is related to the number of interval relations. We created a Markov Logic Network to carry out the experiments (see Listing 4.4) that returns the interval relations using hidden predicates that are connected to the observed predicates.

The observed and hidden predicates model the seven Allen algebra interval relations (see Table 2.2 in Chapter 2). We create an increasing number of intervals



**Figure 4.7:** Interval Relations: Runtime. We created a fixed number of intervals and computed the interval relations. The charts shows the effect of an increasing number of interval relations on the runtime.

and compute the interval relations that serve as the input for the Markov Logic Network. The constraints induce that the output of the Markov Logic solver conforms to its input. Hence, it gives us a baseline as temporal RDF(S) reasoning relies on more constraints and may additionally contain weighted statements and weighted constraints. The experiments show that the limit is already reached at  $\approx 4,000$  intervals which require more than 8 million interval relations (see Figure 4.7). We let the reasoner (Markov Logic solver) allocate up to 16 GB RAM (see Chapter 5 for a complete description of the test system) and run always out of memory when further increasing the problem size. So, even though we defined a simple model, we reach the limit at a relatively small number of intervals (statements). This indicates that fine-grained temporal reasoning is hardly possible as maintaining the interval relations requires too many resources. However, the actual limit might be higher as we do not need the hidden predicates and constraints that infer already known information. This frees up memory which was the biggest issues in this test case. But we also need to consider that the complexity of the modeled problem increases due to additional predicates and constraints (see Chapter 3).

The results also show that Model **M2** are not feasible as  $8.0E + 06$  interval relations correspond to  $\approx 50$  intervals in the initial dataset (see Table 4.1). We recognize that we need to compute as much as possible before we execute the Markov Logic solver as it needs more resources and is also slower than a comparable JAVA method. We measured that generating the intervals and calculating the relations took only a few seconds in JAVA compared to nearly 10 minutes that were required to solve the Markov Logic Network for 4,000 intervals. Hence, we conclude that the limit of model **M3** is well below 50 intervals.

So, we decided to select Model **M1** as the other options are not practicable as most of the resources would be used to process the interval relations instead of detecting inconsistencies within the data or inferring new statements. Even the most efficient approach (**M1**) adds  $\mathcal{O}(n^2)$  additional ground predicates to the Markov Logic Network. However, we mitigate these effects by defining observed predicates for the interval relations that cause that the reasoner directly knows which relation holds between two intervals.



## Chapter 5

# Evaluation and Applications

In this chapter, we complement the theory by evaluating the practicality of our approach. Therefore, we apply our application to different datasets and use cases. We focus on the quality of the output of our application in terms of precision, recall and F-measure and also investigate the runtime efficiency. We selected the following use cases:

The first use case (see Section 5.1) is a benchmark that was developed to evaluate the reasoning capabilities of a knowledge base system. It does only require inferring new statements and does not contain weighted statements or constraints. We selected this use case in order to evaluate the basic features of our application. Moreover, it gives us a baseline for more complex use cases.

The second use case (see Section 5.2) bases on facts extracted from DBPedia [Auer et al., 2007; Bizer et al., 2009b; Lehmann et al., 2014]. Based on the derived statements, we create datasets that contain many weighted statements which violate the defined constraints. These datasets simulate datasets created by an open information extraction system [Etzioni et al., 2008, 2011] that have similar characteristics. Using such a dataset was not feasible in the context of this work because it requires developing a framework that makes the data processable by our application. Nevertheless, the focus of this use case is to remove inconsistent statements from a dataset.

The third use case (see Section 5.3) is unrelated to the other use cases. It contains temporal annotated statements that are used to evaluate activity recognition algorithms. We transform the dataset to the RDF data model in order to apply our application. The purpose of this use case is to infer activities based on observa-

	<b>DBP extract</b>	<b>Sensor Data</b>
goal	data cleansing	activity recognition
statements	> 150,000	< 20,000
constraints	10 + RDF(S)	≈ 800 (partially complex)
RDF(S) reasoning	required	disabled

**Table 5.1:** Characteristics of the datasets (DBP = DBPedia).

tions which are temporal annotated. Therefore, it requires many complex weighted constraints that interact with each other. This makes it valuable as the other use cases only rely on a comparable small number of hard constraints (see Table 5.1). Moreover, we show that our approach is not limited to a specific domain.

All sections in this chapter follow the same pattern: They start with a brief introduction to the use case and an explanation why the respective use case is relevant for the evaluation. After that, we outline the characteristics of the dataset and explain the applicable constraints. The next part comprises the description of the experiments and the report of the results. Each section concludes with a discussion of the results.

In the remainder of the introduction to this chapter, we summarize different key figures and measures which we use in following sections.

## Parameters and Key Figures

We keep track of different parameters and key figures for each test case. This includes general information as well as information about the input and output. The attributes, which are described in the following, will be recorded when evaluation mode of our application is enabled.

### General Information

In order to identify a test case we save the input parameters:

<b>sDate</b>	the date and time of the start of the test
<b>sNote</b>	a short note specified by the user
<b>sFile</b>	name of the input file
<b>sRDFReasoning</b>	selected RDF reasoning approach (true, basic (no entailment rules), none (differently typed variables))

### Problem Size

We want to investigate the influence of the problem size on the runtime of the application. Thus, we record the number of statements, constraints and intervals:

<b>nIT, nIQ, nI</b>	number of input triples (nIT), input quads (nIQ) and the total number (triple + quad) of statements (nI)
<b>nConstraints</b>	number of domain specific constraints
<b>nIntervals</b>	number of intervals

### Runtimes

We measure the runtime of the different parts of the application:

<b>tLoadModel</b>	loading the input RDF document
<b>tMLN</b>	extracting the constraints from the RDF document and converting them to the Markov Logic syntax
<b>tDB</b>	extracting the data and converting it to the Markov Logic syntax
<b>tOut</b>	writing the model file (definition of predicates and constraints) and the evidence file (ground values) for the Markov Logic solver to the disk
<b>tMAP</b>	time taken by the Markov Logic solver to compute the MAP state
<b>tRDF</b>	converting the output of the Markov Logic solver (ground predicates) to a RDF document
<b>tEval</b>	computing and storing the information that is part of the evaluation (i.e., all variables described in this section)
<b>tTotal</b>	total runtime (excluding tEval)

### Output

We do not only convert the output of the Markov Logic solver to RDF but also analyze the statements and categorize them. A statement that is part of the MAP state is assumed to be correct as it does not violate any constraints. Statements that were contained in the initial dataset and are not part of the MAP state are considered as wrong statements as they needed to be removed in order to resolve inconsistencies. The statements that got inferred by the reasoner are the new statements. A standard statement (i.e., `triple` or `tripleO`) that was inferred from a temporal annotated statement (i.e., `quad` or `quadO`) will not be considered. Hence, we can determine the following numbers:

<b>nCT, nCQ, nC</b>	number of correct statements (triple, quad, total)
<b>nWT, nWQ, nW</b>	number of wrong statements (triple, quad, total)
<b>nNT, nNQ, nN</b>	number of new statements (triple, quad, total)

### Assessing the Quality of the Output

In order to determine the quality of the output, we compute the precision, the recall and the F-measure. These measures depend on the number of true positives (TP), false positives (FP) and false negatives (FN) [Olson and Delen, 2008]:

$$precision = \frac{TP}{TP + FP} \quad (5.1)$$

$$recall = \frac{TP}{TP + FN} \quad (5.2)$$

$$F - measure = \frac{2 \cdot precision \cdot recall}{precision + recall} \quad (5.3)$$

### System

We executed all tests on a virtual machine running Ubuntu 12.04 LTS (64-bit) that has access to two threads of the CPU (2.4 GHz) and 16 GB RAM. Moreover, we use JAVA<sup>1</sup> 1.7.0\_51, Apache Jena<sup>2</sup> 2.11.1, rockIt<sup>3</sup> 0.4.257, MySQL<sup>4</sup> 5.5.37 and Gurobi<sup>5</sup> 5.6.1.

---

<sup>1</sup><http://java.com/>

<sup>2</sup><https://jena.apache.org/>

<sup>3</sup><https://code.google.com/p/rockit/>

<sup>4</sup><http://www.mysql.com/>

<sup>5</sup><http://www.gurobi.com/>

## 5.1 Standard RDF(S) Reasoning

We independently validate the RDF(S) reasoning capability of our application as it is a core component that also works without temporal annotated statements. Therefore, we use the “Lehigh University Benchmark” (LUBM) [Guo et al., 2005]<sup>6</sup> that was developed for benchmarking Semantic Web knowledge base systems. This dataset does not contain inconsistencies but it requires the application to infer new statements. Moreover, it allows us to test the flexibility of our approach as it forces us to extend the inference rules.

### 5.1.1 Data & Constraints

The benchmark contains an ontology<sup>7</sup> that describes the university domain. It uses the OWL lite standard [Welty and McGuinness, 2004]<sup>8</sup> which requires us not only to map some classes and properties to their comparable counterparts of RDF(S) but also to add additional rules. This is necessary as OWL lite provides more properties as well as the possibility to define classes using complex class expressions. In particular, we define the following statements and constraints to connect the OWL vocabulary to the RDF(S) vocabulary:

- $\text{triple}(\text{owl} : \text{Class}, \text{rdfs} : \text{subClassOf}, \text{rdfs} : \text{Class})$
- $\text{triple}(\text{owl} : \text{DatatypeProperty}, \text{rdfs} : \text{subClassOf}, \text{rdf} : \text{Property})$
- $\text{triple}(\text{owl} : \text{ObjectProperty}, \text{rdfs} : \text{subClassOf}, \text{rdf} : \text{Property})$
- $\text{triple}(\text{owl} : \text{TransitiveProperty}, \text{rdf} : \text{subClassOf}, \text{rdf} : \text{Property})$
- $\text{triple}(\text{owl} : \text{inverseOf}, \text{rdf} : \text{type}, \text{rdf} : \text{Property})$
- $\text{triple}(a, p, b) \wedge \text{triple}(b, p, c) \wedge \text{triple}(p, \text{rdf} : \text{type}, \text{owl} : \text{TransitiveProperty}) \Rightarrow \text{triple}(a, p, c)$
- $\text{triple}(a, p1, b) \wedge \text{triple}(p1, \text{owl} : \text{inverseOf}, p2) \Rightarrow \text{triple}(b, p2, a)$
- $\text{triple}(p1, \text{owl} : \text{inverseOf}, p2) \Rightarrow \text{triple}(p2, \text{owl} : \text{inverseOf}, p1)$

The ontology also uses more complex concepts, e.g., `owl:Restriction` or `owl:intersectionOf`, to define axioms like:

$$\text{GraduateStudent} \sqsubseteq \text{Person} \sqcap \exists \text{takesCourse}.\text{GraduateCourse}$$

We can neither extract those types of statements from the ontology using generic constraints as there are too many scenarios and combinations nor formulate them

<sup>6</sup><http://swat.cse.lehigh.edu/projects/lubm/>

<sup>7</sup><http://swat.cse.lehigh.edu/onto/univ-bench.owl>

<sup>8</sup><http://www.w3.org/TR/2004/REC-owl-guide-20040210/>

correctly with our approach. First, we need to split the class expression into two constraints that are implicitly conjoined:

$$\text{GraduateStudent} \sqsubseteq \text{Person}$$

$$\text{GraduateStudent} \sqsubseteq \exists \text{takesCourse}.\text{GraduateCourse}$$

While the first statement can be expressed with our formalism the second cannot. The direct transformation of it is:

$$\begin{aligned} & \text{triple}(s, \text{"rdf:type"}, \text{"GraduateStudent"}) \\ & \Rightarrow \text{triple}(s, \text{"takesCourse"}, c) \wedge \text{triple}(c, \text{"rdf:type"}, \text{"GraduateCourse"}) \end{aligned}$$

This constraint has a conjunction of literals on the right-hand side of the implication. As they are connected by a variable that does not occur on the left-hand side of the implication, we cannot split this constraint. This forces us to move one literal to the left-hand side of the implication. Moving  $\text{triple}(c, \text{rdf:type}, \text{GraduateCourse})$  would cause a student to take all graduate courses. By moving  $\text{triple}(s, \text{takesCourse}, c)$ , we lose the requirement that a student has to take a course at all. As both approaches are not correct, we decided to assume equivalence between the original class expressions and model only the inverse direction:

$$\begin{aligned} & \text{triple}(s, \text{"takesCourse"}, c) \wedge \text{triple}(c, \text{"rdf:type"}, \text{"GraduateCourse"}) \\ & \Rightarrow \text{triple}(s, \text{"rdf:type"}, \text{"GraduateStudent"}) \end{aligned}$$

The difference to the original axiom is that it assigns the type `GraduateStudent` to every entity that takes a graduate course while it does not force a graduate student to take a graduate course. The reason for this choice is that we model other axioms that define equivalence instead of subsumption in the same way. Moreover, the benchmark does not check if this particular axiom is correctly handled by the reasoner as the data only contains graduate students that visit graduate courses. Nevertheless, we showed that it is possible to extend the constraint set with additional inference rules that do not exceed the expressiveness of our approach.

The authors provide a data generator that creates datasets of arbitrary size using the classes and properties of the ontology. This allows evaluating the scalability of a reasoner. However, only slightly more than 100,000 statements are required to evaluate the correctness of a reasoner. The benchmark contains fourteen SPARQL queries<sup>9</sup> as well as their answers<sup>10</sup>. The queries test a variety of properties of a

<sup>9</sup><http://swat.cse.lehigh.edu/projects/lubm/queries-sparql.txt>

<sup>10</sup><http://swat.cse.lehigh.edu/projects/lubm/answers.zip>

knowledge base system including their reasoning capabilities. Hence, this benchmark is valuable for our evaluation as we can test the correctness and scalability of our RDF(S) reasoner. This is a prerequisite to carry out temporal reasoning.

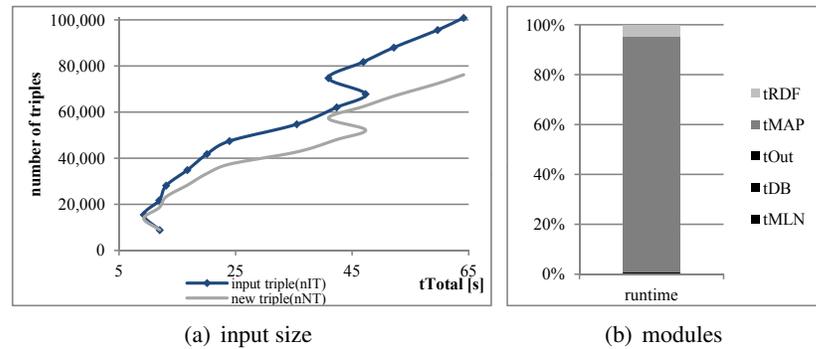
### 5.1.2 Results & Discussion

The data generator creates a series of documents that need to be assembled to one document. Hence, we can not only test if the reasoner works correctly but it also gives us the opportunity to investigate how our well our application scales with an increasing problem size. However, the data required for testing the correctness of the reasoner consists of 15 documents ( $\approx 100,000$  statements). For this document corpus, we observe that the runtime is linearly related to the number of input statements (see Figure 5.1). The number of inferred statements (i.e., new triples) increases as well because all instances get assigned to additional classes. The majority of the runtime ( $\approx 94.5\%$ (60.2s)) is required to compute the MAP state while it takes  $\approx 3.2s$  (4.8%) of the time to create the final document (see Figure 5.1(b)). While we expect that converting the output of the Markov Logic solver to a RDF document only depends on the number of statements, the runtime of the Markov Logic solver can severely increase for other problems, e.g., another constraint set. This problem (RDF(S) reasoning) is not very complex as we only need hard constraints and do not have to consider weighted statements that cause inconsistencies. Furthermore, we see that the other modules of our application have no impact on the total runtime.

We continuously extended the document corpus (see Figure 5.2) in order to reach the maximum number of statements that can be processed by our application. The limit was reached at nearly 1.75 million input statements that could be processed in 220 minutes and led to 1.25 million new statements. While there is a linear relation between the problem size and the runtime for small problems, the runtime increases much faster for bigger problems. Moreover, we see that the runtime varies strongly, i.e., it decreases for larger problems in some cases. This might be caused by optimization techniques (e.g., parallelization) of the Markov Logic solver. We compared those results to a reasoner of a state of the art knowledge base software. Therefore, we selected the system OWLIM lite<sup>11</sup> that provides a build-in in-memory reasoner and created a workflow that is comparable to our application. The workflow includes loading all documents, computing the materialization and exporting the result. Using the OWLIM lite reasoner, we measured that processing the smallest input ( $\approx 10,000$  statements) is 3 times faster and processing the

---

<sup>11</sup>OWLIM lite 5.3 <http://owlim.ontotext.com/display/OWLIMv53>

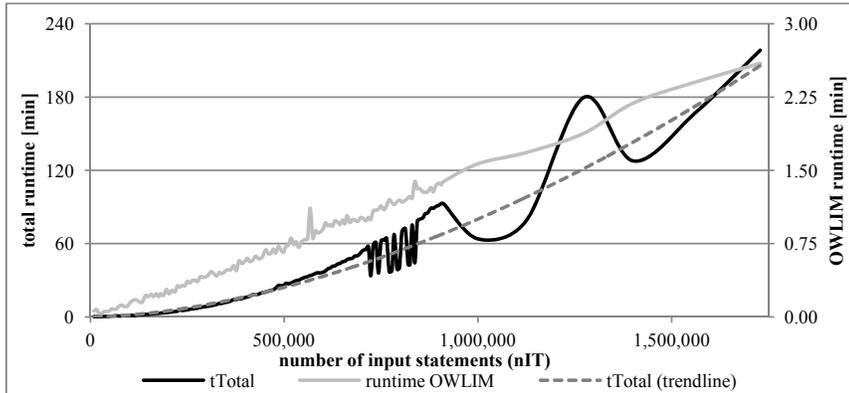


**Figure 5.1:** Reasoning: Runtime: Chart (a) shows that the runtime increases with the number of input statements (i.e., triples). It also shows that the number of inferred statements increases. Diagram (b) shows that computing the MAP state takes 94.5% and creating the final RDF document (i.e., a document containing nearly 180,000 statements) takes 4.8% of the runtime. The basis for this chart is the data which is required to evaluate the results of the SPARQL queries.

largest input, that can be processed by our application ( $\approx 1.75$  million statements), is 84 times faster. The factor increases as the OWLIM lite reasoner scales much better as the relation between the runtime and the number of input statements remains linear. Hence, a state of the art reasoner is not only much faster but is also able to process much more data. However, our approach is more general as it is also applicable to a probabilistic knowledge base.

In order to validate the correctness of our reasoner, we set up a *OpenRDF Sesame*<sup>12</sup> triplestore that allows creating and querying repositories containing RDF data. We created one repository for the original dataset and another one for the document created by our application that contains the facts of the initial dataset and all inferred statements. We executed all 14 queries against both repositories and can report that the original dataset answers only 4 of the 14 queries correct while the other repository answers all queries complete and correct. Hence, the RDF(S) reasoning part of our application works as intended.

<sup>12</sup>Sesame 2.7.11 <http://www.openrdf.org/>



**Figure 5.2:** RDF(S) Reasoning: We compared our application to the reasoner of OWLIM lite. Our system is much slower and does not scale with an increasing problem size. The limit is reached at  $\approx 1.75$  million input statements.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7
<b>O</b>	4/4	0/0	6/6	0/34	0/719	0/7790	0/67
<b>I</b>	4/4	0/0	6/6	34/34	719/719	7790/7790	67/67

	Q8	Q9	Q10	Q11	Q12	Q13	Q14
<b>O</b>	0/7790	0/208	0/4	0/224	0/15	0/1	5916/5916
<b>I</b>	7790/7790	208/208	4/4	224/224	15/15	1/1	5916/5916

**Table 5.2:** Number of correct answers when querying the original dataset (**O**) and the dataset containing all inferred statements (**I**). Both datasets return none wrong answer but only the latter answers all queries completely.

## 5.2 Linked Open Data - DBPedia Extract

DBPedia is the center of the Linked Open Data cloud. It covers many domains as its content is extracted from Wikipedia's info boxes and categories [Auer et al., 2007; Bizer et al., 2009b; Lehmann et al., 2014]. In order to apply our approach, we need to define domain specific constraints. Hence, we need to focus on a small section as we are not able to define constraints that cover all domains contained in DBPedia. We decided to build the dataset around a certain group of persons and their relations among each other. Unfortunately, DBPedia does not provide temporal annotated statements. So, we need to extract the temporal information from dates which are given for most persons as birth date and death date. In particular, we focus on scientists as there are a few properties that connect many instances of this class. This domain also contains many universities for which a founding date is available that can be used to create temporal annotated statements.

### 5.2.1 Data

In order to create the dataset, we need to collect statements from DBPedia (version: 3.9). Therefore, we execute the following steps:

1. We select all statements having one of the properties listed in Table 5.3. Those properties primarily connect instances of type `Person`. However, the domain of the properties `academicAdvisor`, `doctoralAdvisor`, `doctoralStudent` and `notableStudent` is further restricted to instances of the class `Scientist` which is a subclass of the class `Person`. The properties `influenced` and `influencedBy` do not have any domain or range restrictions. They connect not only persons but also entities of other classes like programming languages. Despite the fact that we want focus on persons, we did not apply a filter to exclude the unrelated statements as we want to increase the chance that our application detects erroneous statements.
2. We take the available birth dates and death dates of all persons into account in order to create temporal annotated statements. For this purpose, we use the properties which are listed in Table 5.4. The provided information must at least contain a sequence of four digits that represents the year as we only use the year in the annotation. If more information is provided (e.g., day and month) we do not use it in the annotation but give the statement a higher weight (i.e., we decrease the weight if this information is not available).
3. For each person, we derive the attended universities (property: `almaMater`) from DBPedia if this information is available. So, we get 7,793 statements

property/class	count
dbpedia-owl:academicAdvisor	825
dbpedia-owl:doctoralAdvisor	4,618
dbpedia-owl:doctoralStudent	5,838
dbpedia-owl:notableStudent	865
dbpedia-owl:influenced	11,463
dbpedia-owl:influencedBy	23,827
dbpedia-owl:Person	26,074
dbpedia-owl:Scientist	7,699

**Table 5.3:** DBPedia: Number of selected statements and relevant instances.

property/class	count
dbpedia-owl:birthDate	14,710
dbpprop:birthDate	12,166
dbpprop:dateOfBirth	12,303
dbpedia-owl:deathDate	9,720
dbpprop:deathDate	7,266
dbpprop:dateOfDeath	6,925
dbpedia-owl:almaMater	7,793
dbpedia-owl:University	988
dbpprop:established	920
dbpedia-owl:foundingDate	49

**Table 5.4:** DBPedia: Additional properties and instances.

having the property `almaMater` (see Table 5.4) pointing to 988 universities (see Table 5.4).

4. For each university, we consider the available founding dates (property: `established` or `foundingDate`) and annotate the statements with the respective year.

We assign to statements having a property described in the DBPedia ontology (namespace prefix: `dbpedia-owl`) the weight 1.0 and to statements having a property with the namespace prefix `dbpprop` the weight 0.75. Moreover, temporal annotated statements that are not based on a complete date (e.g., only the year) get their weight decreased by 0.05.

## 5.2.2 Constraints

The next step is to define a set of constraints that allows us to detect erroneous statements in the dataset. First, we introduce constraints that exploit the disjointness between certain classes. Second, we define constraints that must hold in the selected dataset, i.e., domain-specific constraints.

We use the property `owl:disjointWith` to express disjointness between the classes `Person` and `University`. Moreover, statements having properties without domain or range restrictions (e.g., `influenced` or `influencedBy`) introduced additional classes (e.g., `Language`). Most of those classes are disjoint with the class `Scientist` (at least in our dataset) which allows us to add the respective disjointness assertions. Therefore, we need the following constraints that ensure the characteristics of the property `owl:disjointWith`:

$$\text{triple}(c1, \text{"owl : disjointWith"}, c2) \quad \Rightarrow \quad \text{triple}(c2, \text{"owl : disjointWith"}, c1)$$

$$\text{triple}(c1, \text{"owl : disjointWith"}, c2) \wedge \text{triple}(x, \text{"rdf : type"}, c1) \Rightarrow \neg \text{triple}(x, \text{"rdf : type"}, c2)$$

$$\begin{aligned} \text{triple}(c1, \text{"owl : disjointWith"}, c2) \wedge \text{triple}(cs1, \text{"rdfs : subclassOf"}, c1) \\ \Rightarrow \text{triple}(cs1, \text{"owl : disjointWith"}, c2) \end{aligned}$$

These constraints should cover all scenarios but we do not show that they are complete. Nevertheless, the created dataset allows defining the following domain-specific constraints:

- Our dataset contains statements having a property that connects persons, e.g., `academicAdvisor`, `doctoralStudent`, etc. (see Table 5.3). These statements can only be true if the respective entities have lived during an overlapping period. The exception are the properties `influenced` and `influencedBy` that do also allow that the influencer has died before the other person was born. Hence, it is sufficient that the influencer was born before the influenced person died.
- A person can only have visited a university (alma mater) that was established before her/his death.
- A person has at most one birth date and at most one death date.
- A university has at most one founding date.

In order to define the constraints, we need to combine different properties that describe the same aspect of an entity (e.g., the birth date). Hence, we create the three properties `birthDateN`, `deathDateN` and `foundingDateN` that serve as super-properties (see Table 5.5).

So, instead of using every single property we can just use the respective super-property which reduces the number of required constraints. For instance, if we

<b>birthDateN</b>	<b>deathDateN</b>	<b>foundingDateN</b>
dbpedia-owl:birthDate	dbpedia-owl:deathDate	dbpprop:established
dbpprop:birthDate	dbpprop:deathDate	dbpedia-owl:foundingDate
dbpprop:dateOfBirth	dbpprop:dateOfDeath	

**Table 5.5:** DBPedia: New super-properties and their sub-properties.

want to combine the birth date and the death date of a person in a constraint, we can just define one constraint instead of nine constraints covering all combinations. Hence, we benefit from the reasoning capabilities of our approach. In particular, it is enough to extend the underlying ontology with the respective axioms. Replacing the properties of each statement is not necessary as this is done by the reasoner. Moreover, we create the two property classes `TimeFunctionalP` and `OverlappingLifeP` that comprise properties having certain characteristics (see Table 5.6).

<b>TimeFunctionalP</b>	<b>OverlappingLifeP</b>
birthDateN	dbpedia-owl:academicAdvisor
deathDateN	dbpedia-owl:doctoralAdvisor
foundingDateN	dbpedia-owl:doctoralStudent
	dbpedia-owl:notableStudent

**Table 5.6:** DBPedia: New property classes and their instances.

Using the introduced classes and properties, we define domain-specific constraints which we will introduce in the remainder of this section.

Instances of the class `TimeFunctionalP` are properties that connect an entity to a date. The restriction is that one property (e.g., `birthDateN`) can only be part of statements, describing a single entity, that are annotated with the same interval (e.g., a person has only one birth date). Assigning a super-property (i.e., `birthDateN`, `deathDateN`, `foundingDateN`) to this class (`TimeFunctionalP`) ensures that the condition holds for all of its sub-properties. Hence, it is not required to modify the statements in the dataset.

$$\text{triple}(p, \text{"rdf:type"}, \text{"TimeFunctionalP"}) \wedge \text{quad}(x, p, t1, i1) \wedge \text{quad}(x, p, t2, i2) \Rightarrow tEqual(i1, i2)$$

Persons that are connected by a property that is an instance of the class `OverlappingLifeP` must live/have lived during an overlapping time period. We ensure this by checking in both directions if a person was born before the other person died:

$$\begin{aligned} & \text{triple}(x, p, y) \wedge \text{triple}(p, \text{"rdf : type"}, \text{"OverlappingLifeP"}) \wedge \\ & \quad \text{quad}(x, \text{"birthDateN"}, t1, i1) \wedge \text{quad}(y, \text{"deathDateN"}, t2, i2) \\ & \qquad \qquad \qquad \Rightarrow \text{tBefore}(i1, i2) \end{aligned}$$

$$\begin{aligned} & \text{triple}(x, p, y) \wedge \text{triple}(p, \text{"rdf : type"}, \text{"OverlappingLifeP"}) \wedge \\ & \quad \text{quad}(y, \text{"birthDateN"}, t1, i1) \wedge \text{quad}(x, \text{"deathDateN"}, t2, i2) \\ & \qquad \qquad \qquad \Rightarrow \text{tBefore}(i1, i2) \end{aligned}$$

We use the introduced super-property `foundingDateN` to check if a university was established before the death of the persons who visited the respective university:

$$\begin{aligned} & \text{triple}(x, \text{"dbpedia-owl : almaMater"}, u) \wedge \\ & \quad \text{quad}(u, \text{"foundingDateN"}, e, i1) \wedge \text{quad}(x, \text{"deathDateN"}, dd, i2) \Rightarrow \text{tBefore}(i1, i2) \end{aligned}$$

A person has to be born before she/he dies:

$$\text{quad}(x, \text{"t : birthDate"}, t1, i1) \wedge \text{quad}(x, \text{"t : deathDate"}, t2, i2) \Rightarrow \text{tBefore}(i1, i2)$$

Additionally, we add constraints that ensure the characteristics of the properties `influenced` or `influencedBy`:

$$\begin{aligned} & \text{quad}(x, \text{"t : birthDate"}, t1, i1) \wedge \text{quad}(y, \text{"t : deathDate"}, t2, i2) \wedge \\ & \quad \text{triple}(x, \text{"dbpedia-owl : influenced"}, y) \Rightarrow \text{tBefore}(i1, i2) \end{aligned}$$

$$\begin{aligned} & \text{quad}(x, \text{"t : birthDate"}, t1, i1) \wedge \text{quad}(y, \text{"t : deathDate"}, t2, i2) \wedge \\ & \quad \text{triple}(y, \text{"dbpedia-owl : influencedBy"}, x) \Rightarrow \text{tBefore}(i1, i2) \end{aligned}$$

So, we defined seven constraints that must hold in this dataset. While we do not need weighted constraints, we profit from the flexibility and reasoning capability of our approach. That is, we are able to create new classes and properties that reduce the number of required constraints which makes it more comfortable to maintain them.

### 5.2.3 Experiments

The created dataset is suitable to carry out two types of experiments. In the first part of the experiments, we evaluate how well our approach detects wrong statements in the original dataset. We apply our application directly on the derived data and manually evaluate the output with a focus on the removed statements. Hence, we investigate if the removed statements are actual wrong. The second part of the experiments builds on the first part as we remove all statements that caused inconsistencies in the original dataset. This results in a consistent dataset (with respect to the defined constraints) to which we add wrong statements. This part of the experiments is necessary as the initial dataset contains only a small number of wrong statements. Hence, we investigate the influence of an increasing percentage of wrong statements in the dataset on the results in the second part of the experiments. We conclude this section with a brief report on the runtime of our application. However, we will not investigate if the inferred statements are correct as the constraints are designed to detect inconsistencies in the dataset.

#### Part 1: Original Dataset

Our application detects 359 (48 triples and 311 quads) wrong statements in the dataset. This means that less than 0.5% of the statements cause inconsistencies. In the following, we separately report on the results of both groups of statements (triples and quads) as they are marked as wrong for different reasons. Moreover, we classify the statements of both groups into additional categories and report the number of statements (count) and the precision of each category. The results reported in this section exclusively refer to the statements that got identified as wrong statements by our application.

**Triples.** The application declared 48 triples as wrong statements for the following reasons (grouped by the respective property / property class):

**OverlappingLifeP (count: 33 / precision: 1.0):** The most common problem was that statements violate the range restrictions of properties that are part of this class (count: 29) (see Table 5.6). For example, a `University` instead of a `Person` was listed as an `academicAdvisor` of a `Scientist`. In the other cases, an entity that has lived at a different time was assigned as `doctoralStudent` or `doctoralAdvisor` (4). Those cases can be further classified as follows: The assigned entity has the same name as the correct person (2), is unrelated (1), had a connection (e.g., same research area) but died before the subject of the statement was born (1).

**rdf:type (count: 6 / precision: 0.44):** The violation of domain and range restrictions of properties can cause that our application drops the type assignment instead of the property assertion if this leads to a state with a higher weight (MAP state). We observed that some universities are part of multiple statements having a property that only connects persons (e.g., `doctoralAdvisor`, `doctoralStudent`). Moreover, our application drops the type assignment of a university if it has less properties that indicate via their domain or range restriction that it is of type `University` (e.g., `almaMater`).

**influenced / influencedBy (count: 9 / precision: 1.0):** Our application detected statements in which the subject and object were interchanged (6), a wrong entity having the same name was assigned (1) and unrelated persons were connected.

So, we achieve a precision of 91.67%(44/48) for triples. The most frequent problem was that statements violate domain and range restriction of properties (35/48). Other frequent issues are interchanged subject and object (6/48) and wrong entity resolution (3/48), i.e., assignment of a different entity having the same name.

**Quads.** Our application identified 311 temporal annotated statements that cause inconsistencies according to the defined constraints. Of these statements express 9 statements the founding date of a university and 302 statements provide the birth date or death date of a person. The statements declaring the founding date are all correct which means that the precision of our application is 0%. The statements got detected as some alumni died before the university was established. However, it is often the case that the respective persons studied at the university but the resource refers to a reestablished university having the same name or one that resulted from the union of other universities. Hence, it is necessary to check if there is an entity describing the original university in order to solve the problem. The second group contains all statements stating the birth date or date death of a person. Our constraints ensure that an entity can have at most one of each. However, the statements got removed from the dataset for the following reasons:

**properties: influenced / influencedBy (count: 8 / precision: 0.0):** As already mentioned, it happens that the subject and object of these properties are interchanged. This causes a violation of the constraints if the respective entities lived at distanced times. The introduced inconsistency cannot be correctly resolved if this happens multiple times for a single entity. In such a scenario, our application removes the birth date or death date instead of the wrong statement.

In the remaining cases (294), the statements got removed as the dataset contained multiple different birth dates or death dates for a person. This violates the constraint that a person has at most one birth date and at most one death date. As the provided dates are mostly close to each other, i.e., less than 5 years difference, our application makes the decision only based on the other dates (i.e., birth dates and death dates) that are given for the respective person. The decision can be reasoned or random:

**reasoned (count: 136 / precision: 0.86):** We derived the dates using various properties having different weights (see Table 5.4). If some of those properties agree on a date, our application makes a founded decision as the weights of those statements sum up and are then higher than the weight of the wrong statement. In other cases, the correct death date is also given as birth date (or vice versa). Even if the birth dates would have the same weight, our application makes the correct decision as it also considers that a person must be born before she/he can die. Hence, it makes a decision that keeps a birth date and a death date for the person as this leads to a state with a higher total weight.

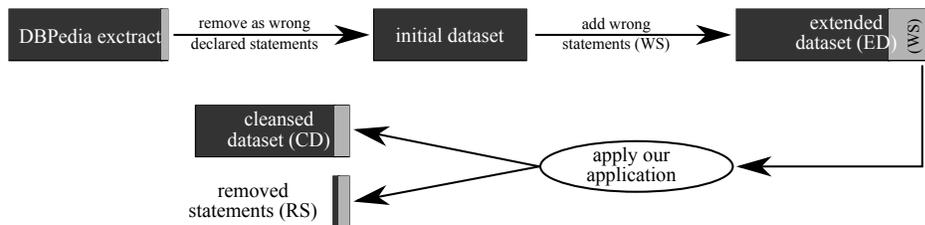
**random (count: 158 / precision: 0.59):** In some cases, there is no evidence that makes one date superior to another date. In those scenarios, the application selects randomly one statement. As the alternatives are mostly very close and often before the 20<sup>th</sup> century it is even for a domain expert hard to make the correct decision.

Overall, the precision is 67.88% (205/302) for the detected birth dates and death dates. The reason for the low precision is that the decision was often made between two close dates. As this dataset does not allow fine-grained temporal reasoning, it is not possible to improve the results. However, we can report that our application acted as intended in all situations. It highlights parts of the dataset that contain erroneous statements in 100% of cases. So, the system indicates with a very high precision which areas of the dataset need to be inspected by a domain expert.

## Part 2: Adding additional wrong Statements

We reported in the first part of the experiments (previous section) that the data derived from DBpedia does not contain many wrong statements. This forces us to generate additional statements, which are supposedly wrong, in order to evaluate how well our application handles a higher percentage of wrong statements. The goal is to simulate a dataset containing many incorrect statements like a dataset collected by an open information extraction system that crawls data on the Web. Therefore, we carry out the following steps (see Figure 5.3):

1. We remove all statements that cause inconsistencies from the initially derived DBPedia extract. This leads to a dataset, called initial dataset, which is consistent with respect to the defined constraints. Hence, we assume that the F-measure of this dataset is 1.0 as it contains all statements (recall = 1.0) and because it is free from inconsistencies (precision = 1.0).
2. We generate wrong statements (WS) and add them to the initial dataset. This leads to the extended dataset (ED) which has a recall of 1.0. The precision depends on the number of added wrong statements (WS) as we vary the share of WS.
3. We apply our application to the extended dataset (ED). The output of the application comprises the removed statements (RS) as well as the cleansed dataset (CD). In the first part of the experiments, we exclusively focused on the removed statements (RS) as we do not know the precision of the DBPedia extract. Additionally, the improvement could not be noticeable as only a very small number of statements got removed, i.e., less than 0.5%. However, in this part of the experiments we can also evaluate the cleansed dataset (CD) as we know the measures of the extended dataset (ED).



**Figure 5.3:** DBPedia: Overview on the process of the second part.

Before we start to explain the experiments and report the results of this part, we outline how we generate the additional statements in order to obtain the extended dataset (ED) based on the initial dataset.

**Generating wrong statements.** We extend the initial dataset with temporal statements as well as with connections between resources (e.g., persons, universities). All temporal statements in our dataset are annotated with a year which can be a birth year or death year of a person or a founding year of a university. This gives us various options to create wrong statements based on the correct statements. In particular, we use the following techniques, having different probabilities to be selected, to compute a wrong year:

**mixed digits (20%):** We randomly mix the digits of the original year while we ensure that the resulting year is not smaller than 1,000. So, a small as well as a big difference to the original year is possible (e.g., 1954 → 1945 or 1954 → 1549).

**swapped digits (60%):** We swap only two digits while considering two independent criteria. In 50% of the cases, we ensure that the first digit remains in its place in order to minimize the error. In 70% of the cases, we ensure that the swapped digits are neighboring in order to simulate typos.

**small error (20%):** We add (or subtract) a small number (1 – 20) to the original value in order to simulate a minor error.

Beside this, we randomly select persons and assign them a birth year or death year in the range from 1,500 to 2,100 as most of the correct data falls in this interval. This ensures that persons that do not have such information in the original dataset can get one in the extended dataset (ED). So, we are covering a wide range of errors, i.e., from minor typos to completely wrong information. Moreover, we add additional links between resources. Therefore, we randomly select two entities without considering their types. Based on the property distribution in the original dataset, we use either a property that forces that the entities have lived during an overlapping time period or we set the second entity as alma mater of the first entity. We restrict the procedure to these two types of properties as the other properties do not interact with many constraints. Overall, we introduce various errors:

- The created statements can be wrong as we do not check if the selected entities match the domain and range restrictions of the selected property.
- If we connect two persons it is also possible that they have lived at distanced time periods. This violates the overlapping life property.
- If we set the alma mater of a person it is possible that she/he died before the university was established.

The effect of the latter two points gets amplified by the generated temporal statements. Hence, we created a realistic use case that allows us to investigate if the application is able to detect those errors based on the defined constraints.

In order to evaluate how well our approach handles erroneous data, we add presumably wrong statements (WS) to the initial dataset as explained. We vary the share of wrong statements as well as the weight of the added wrong statements in the dataset. In general, we do not assign weights to statements that are contained

in the terminological part of the ontology as we assume that they are correct. In the remainder of this section, we report the results of the following aspects of the experiments:

**Baseline.** We assign to all statement that are contained in the initial dataset (ID) the weight 1.0 while the generated statements get a random weight between 0.0 (exclusive) and 0.1. Hence, the generated wrong statements (WS) should be removed from the dataset.

**Increasing share of wrong statements.** We assign to all statements a weight between 0.0 (exclusive) and 1.0 and report the effects of an increasing percentage of wrong statements.

**Decreasing weights of the wrong statements.** We assign to all statement that are contained in the initial dataset (ID) a random weight between 0.0 (exclusive) and 1.0 and to the generated statements a random weight between 0.0 (exclusive) and a varying upper bound.

**Temporal statements vs. relations among entities.** We report if there are differences between these two types of added statements as they interact with different rules.

We focus on the precision, recall and F-measure of the removed statements (RS) but also state the effect on the complete dataset by reporting  $\Delta$  F-measure. We define  $\Delta$  F-measure as the F-measure of cleansed dataset (CD) minus the F-measure of the extended dataset (ED). In order to complement these values, we report the precision and the recall of the extended dataset (ED) and the cleansed dataset (CD) in tables. This is necessary as it helps to understand  $\Delta$  F-measure. The reported measures (precision, recall and F-measure) of the extended dataset (ED) and the cleansed dataset (CD) exclude statements that were not considered when generating the additional statements, e.g., the terminological part of the dataset. However, we consider all statements when assessing the quality of the removed statements (RS).

In order to compute the precision and the recall of a dataset (e.g., the cleansed dataset), we define the true positives (TP) as the statements that were also part of the initial dataset, the false positives (FP) as the (wrong) statements which we added to the dataset and got not removed from the dataset and the false negatives (FN) as the statements that were part of the initial dataset and got removed by our application.

In order to compute the precision and the recall of the removed statements (RS), we define the true positives (TP) as the statements which we added to the initial dataset, the false positives (FP) as the statements that were part of the initial dataset and got removed by our application and the false negatives (FN) as statements which we added to the initial dataset and got not removed by our application.

The goal of our approach is to improve the precision while retaining a high recall of the complete dataset. Therefore, it is important that the precision of the removed statements is also high, i.e., we remove actual wrong statements. In general, it makes sense to apply our approach when  $\Delta$  F-measure has a positive value. However, this value can be low when we add a small number of wrong statements because the data quality of the extended dataset (ED) is already high. We repeated the experiments multiple times as we create the additional statements randomly.

**Baseline.** First, we determine the best possible results that can be achieved for the created dataset (ED). Therefore, we give all wrong statements a much lower weight (weight between 0.0 and 0.1) than the correct statements (weight = 1.0). This setting should allow our application not only to detect all statements that violate the constraints but to remove the added (wrong) statements (WS) from the dataset in conflicting situations. The experiments show that the precision is close to 100% while the recall is above 75% for the removed statements (RS) (see Table 5.7). Hence, the recall of the cleansed dataset (CD) remains close to 100% and its precision increases. This leads to a better F-Measure (absolute improvement 0.4 – 23.1 percentage points). The degree of the improvement depends on the amount of wrong statements in the extended dataset as the correctness of the removed statements seems to be independent from the share of wrong statements (WS). This underlines the fact that the distribution of the added statements corresponds to the original dataset and that the introduced inconsistencies are equally distributed.

However, the recall is well below 100% which means that many added statements do not violate any constraints. There are several explanations for this observation:

- When we add a connection between two entities it is not unlikely that they have lived at the same time.
- Assigning an additional university as alma mater to a person does not lead to a conflict if the university was founded before the death of that person.

	0.01 WS			0.10 WS			0.25 WS		
	P	R	F	P	R	F	P	R	F
ED	0.99	1.0	0.995	0.909	1.0	0.952	0.800	1.0	0.889
CD	0.998	1.0	0.999	0.976	1.0	0.988	0.942	1.0	0.970
$\Delta$	0.008	0.0	0.004	0.067	0.0	0.035	0.142	0.0	0.081
RS	1.0	0.753	0.859	1.0	0.752	0.858	1.0	0.755	0.860

	0.50 WS			0.75 WS			1.00 WS		
	P	R	F	P	R	F	P	R	F
ED	0.667	1.0	0.8	0.572	1.0	0.727	0.5	1.0	0.667
CD	0.894	1.0	0.944	0.852	1.0	0.920	0.814	1.0	0.898
$\Delta$	0.227	0.0	0.144	0.280	0.0	0.193	0.314	0.0	0.231
RS	1.0	0.764	0.866	1.0	0.768	0.869	1.0	0.772	0.871

**Table 5.7:** DBpedia: Additional Statements: Baseline for varying percentages of wrong statements (abbreviations: p = precision, r = recall, F = F-measure, ED = extended dataset, CD = cleansed dataset, RS = removed statements, WS = wrong statements).

- While the constraints respect the domain and range of the properties, it is still possible that the reasoner assigns an entity to an additional class if that does not violate the class disjointness axioms. Many instances are only assigned to the class `Person` and to none of its subclasses. Hence, further specifying the type of the instance does not lead to inconsistencies.
- Moreover, when we assign a death date to a person that is still alive or does not have a death date in the initial dataset it is possible that this error cannot be detected.

These points primarily indicate limitations of the used dataset and not of our application. Most of those issues would not be as significant in a more connected and fine-grained dataset that relies on a series of interacting constraints.

**Increasing share of wrong statements.** The next part of the experiments is more realistic as we randomly assign to all statements a weight in the range from 0.0 to 1.0. This leads to many situations in which the reasoner does not necessarily keep the correct statement when it is involved in a conflict. For instance, we added temporal annotated statements that differ only by a smart margin from the correct value. The reasoner takes the statement with the higher weight as our dataset is not very fine-grained in such scenarios. However, this effect is weakened for the birth dates and death dates as those dates are often provided by multiple properties. This allows the reasoner to select a statement using a weighted majority vote. Overall, we expect a lower precision and in consequence of that also a lower recall compared to the baseline.

WS	removed statements (RS)			ED	cleansed dataset (CD)			$\Delta$ F-m.
	precision	recall	F-measure		precision	recall	F-measure	
0.01	0.799	0.629	0.704	0.990	0.996	0.998	0.997	0.002
0.1	0.801	0.643	0.713	0.909	0.965	0.984	0.974	0.022
0.25	0.812	0.648	0.720	0.800	0.916	0.962	0.939	0.050
0.5	0.824	0.654	0.729	0.667	0.843	0.930	0.884	0.084
0.75	0.829	0.653	0.731	0.572	0.776	0.899	0.833	0.106
1.0	0.836	0.654	0.734	0.500	0.716	0.872	0.786	0.119

**Table 5.8:** DBPedia: Varying fraction of added wrong statements (WS). Precision, recall and F-measure is given for the removed statements (RS) and for the cleansed dataset (CD). The precision of the extended dataset (ED) is also given while its recall is always 1.0.  $\Delta$  F-measure indicates the improvement of the F-measure of the complete dataset.

We increase the fraction of wrong statements to verify our assumptions (see Table 5.8). The results are worse than the baseline as the precision dropped significantly from 100% to  $\approx 80\%$ . With respect to an increasing fraction of wrong statements (WS), we observe that the precision improves slightly from 79.9% to 83.6%. This indicates that the dataset contains many very small connected subgraphs. Hence, it is possible that the additional statements are placed in independent areas. The introduced inconsistencies are then resolved with a comparable precision. If more than one wrong statement is placed in a connected subgraph it is possible (but not always the case) that the conflict can be resolved with a higher precision as shown in the following examples (see Example 19 & 20).

**Example 19** *Considering the following set of statements that contains one wrong statement that is involved in a conflict ( $c$  = correct,  $w$  = wrong statement):*

```
c: tripleW(Lucia_Caporaso, rdf:type, Person, 0.24)
c: tripleW(Joe_Harris, doctoralStudent, Lucia_Caporaso, 0.61)
w: tripleW(Dolf_Sternberger, randomAlmaMater, Lucia_Caporaso, 0.97)
```

*The range of the property of the wrong statement does not match the type of Lucia\_Caporaso. In consequence of that, the application drops both correct statements as the wrong statement has a higher weight. Thus, the type of the entity Lucia\_Caporaso changes to University which is disjoint with Person. Only the following statement remains in the dataset:*

```
w: triple(Dolf_Sternberger, randomAlmaMater, Lucia_Caporaso)
```

*Another wrong statement is added to this conflict:*

```
c: tripleW(Lucia_Caporaso, rdf:type, Person, 0.24)
c: tripleW(Joe_Harris, doctoralStudent, Lucia_Caporaso, 0.61)
w: tripleW(Dolf_Sternberger, randomAlmaMater, Lucia_Caporaso, 0.97)
w: tripleW(Joe_Cutler, randomOverlap, Lucia_Caporaso, 0.49)
```

Now, it is better not to change the type of the entity *Lucia\_Caporaso*. Hence, none correct statement but one wrong statement gets removed from the dataset:

```
c: triple(Lucia_Caporaso, rdf:type, Person)
c: triple(Joe_Harris, doctoralStudent, Lucia_Caporaso)
w: triple(Joe_Cutler, randomOverlap, Lucia_Caporaso)
```

*In summary, the application removed two correct statements from the dataset in the first case. By adding an additional statement to the dataset, the application removes one wrong statement and keeps all correct statements. Hence, precision and recall of the removed statements (RS) improve.*

Example 19 also indicates that it is more likely to detect added statements having the property `almaMater` and violating the range restriction of this property when the dataset contains a higher share of wrong statements. It is probable that the object of the added statement is of type `Person` as the initial dataset contains much more instances of the class `Person` than instances of the class `University`. Hence, the chance that the type of the entity gets changed to the class `University` decrease when more statements are added. This is caused by the fact that most of the statements contained in the initial dataset use properties which have the domain and/or range restriction `Person` and that we keep this ratio in the extended dataset (ED).

**Example 20** *Considering the following set of statements that contains one wrong statement ( $c$  = correct,  $w$  = wrong statement):*

```
c: quadW(Ferdinand_Cohn, birthDate, 1828-01-24, [1828,1828], 0.42)
c: quadW(Ferdinand_Cohn, deathDate, 1898-06-25, [1898,1898], 0.05)
c: tripleW(Georg_Lunge, doctoralAdvisor, Ferdinand_Cohn, 0.02)
w: quadW(Ferdinand_Cohn, birthDate, generated, [1952,1952], 0.56)
```

*The application keeps only the generated birth year of *Ferdinand\_Cohn* as this leads to the consistent dataset with the highest weight. The sum of the weights of the correct statements is smaller than the weight of the wrong statement. Thus, the wrong statement will not be removed from the dataset. The correct birth date gets removed as a person has at most one birth date. The correct death date gets removed as it has to be after the birth date. *Georg\_Lunge* lived from 1828 to 1898. Hence, *Ferdinand\_Cohn*, whose birth year got changed to 1952, cannot be his doctoral advisor. So, only the wrong statement remains:*

```
w: quad(Ferdinand_Cohn, birthDate, generated, [1952,1952])
```

*Another wrong statement is added to this conflict:*

```

c: quadW(Ferdinand_Cohn, birthDate, 1828-01-24, [1828,1828], 0.42)
c: quadW(Ferdinand_Cohn, deathDate, 1898-06-25, [1898,1898], 0.05)
c: tripleW(Georg_Lunge, doctoralAdvisor, Ferdinand_Cohn, 0.02)
w: quadW(Ferdinand_Cohn, birthDate, generated, [1952,1952], 0.56)
w: quadW(Ferdinand_Cohn, birthDate, generated, [1540,1540], 0.58)

```

*In consequence of that, it is enough to change the birth year of Ferdinand\_Cohn. So, he can be the doctoral advisor of Georg\_Lunge and also his correct death date causes no inconsistencies:*

```

c: triple(Georg_Lunge, doctoralAdvisor, Ferdinand_Cohn)
w: quad(Ferdinand_Cohn, birthDate, generated, [1540,1540])
c: quad(Ferdinand_Cohn, deathDate, 1898-06-25, [1898,1898])

```

*In summary, the application removed three correct statements from the dataset in the first case. By adding an additional statement to the dataset, the application removes only one correct statement and also one wrong statement. Hence, precision and recall of the removed statements (RS) improve.*

Example 20 also shows that adding many statements to an entity via a functional property (e.g., `birthDate`) has a positive effect on the precision and recall of the removed statements (RS). The cleansed dataset contains only one fact using a functional property per entity. Thus, the wrong statements get removed with a high precision and also with a high recall.

Independently from the share of wrong statements, the recall remains around 65% (see Table 5.8) which is just 10 percentage points lower than the baseline. The reason for this reduction is the lower precision. It causes that correct statements instead of the added statements (WS) got removed from the dataset in conflicting situations. Overall, the F-measure of the cleansed datasets ( $\Delta$  F-measure) improves in all test cases. Independently from the share of wrong statements in the dataset, the improvements of  $\Delta$  F-measure for the different test cases are roughly half as large as the reported baseline of  $\Delta$  F-measure.

**Decreasing weights of the wrong statements.** In the previous experiment, we assigned to all statements a weight within the same range. Now, we make the wrong statements more explicit to the reasoner by limiting their weights. Hence, the correct statements have a higher average weight. This simulates a scenario in which the correct statements are provided by more trustworthy sources or that multiple sources agree on a fact. The precision as well as the recall increase by further limiting the weight of the wrong statements (see Table 5.9). The absolute number of removed statements is constant as the same amount of inconsistencies gets detected. Those inconsistencies can be resolved with a higher precision when the

limit	removed statements (RS)			cleansed dataset (CD)			$\Delta$ F-measure
	precision	recall	F-measure	precision	recall	F-measure	
1.0	0.824	0.654	0.729	0.843	0.930	0.884	0.084
0.7	0.898	0.693	0.782	0.862	0.961	0.909	0.109
0.5	0.939	0.718	0.814	0.874	0.977	0.922	0.122
0.3	0.970	0.740	0.840	0.884	0.989	0.933	0.133
0.1	0.992	0.756	0.858	0.891	0.997	0.941	0.141
baseline	1.0	0.764	0.866	0.894	1.0	0.944	0.144

**Table 5.9:** DBPedia: Limiting the weights of the added wrong statements (WS). The basis for these results is a dataset with 50% additional wrong statements (see Tables 5.7 & 5.8). Hence, the precision of the extended dataset is 0.667, the recall is 1.0 and the F-measure is 0.800. The F-measure of the dataset improves ( $\Delta$  F-measure) by decreasing the weight of the wrong statements.

WS	removed statements (RS)					
	p(r)	p(t)	r(r)	r(t)	F(r)	F(t)
0.01	0.832	0.781	0.515	0.731	0.636	0.755
0.1	0.818	0.790	0.532	0.739	0.644	0.764
0.25	0.835	0.798	0.530	0.750	0.648	0.773
0.5	0.848	0.810	0.528	0.763	0.651	0.786
0.75	0.853	0.816	0.518	0.771	0.644	0.793
1.0	0.858	0.824	0.510	0.778	0.640	0.800

**Table 5.10:** DBPedia: Measures ( $p$  = precision,  $r$  = recall,  $F$  = F-measure) for the temporal statements (t) and the relations among entities (r) for a varying fractions of added wrong statements (WS).

weight of the added (wrong) statements (WS) is lower. Hence, the recall increases as more wrong statements get removed. The precision and the recall is very close to the baseline if the wrong statements (WS) have a weight between 0.0 and 0.1. The results do not match the baseline because 10% of the correct statements have a weight in the same range as the wrong statements.

**Temporal statements vs. relations among entities.** Finally, we investigate how well the different types of statements get handled in our dataset. Therefore, we distinguish between temporal statements and relations among entities. The recall is much higher for temporal annotated statements (see Table 5.10). The reason for this is that most dates are reported by multiple properties and that the temporal properties are functional, e.g., a person has at most one birth date and a university has at most one founding date. Hence, it is very likely that a conflict gets detected and it is also reasonable the respective measure improve when we increase the share of wrong statements. Contrary, the recall of the statements describing relations among entities is much lower as not all wrong statements violate the defined

	DBP	ID	0.1 WS	0.25 WS	0.5 WS	0.75 WS	1.0 WS
<b>nI</b>	195,055	194,692	207,177	225,358	255,148	283,891	313,667
<b>tTotal [min]</b>	6.80	6.64	7.54	8.46	10.14	12.98	18.34
<b>nIntervals</b>	818	809	1,559	1,936	2,234	2,397	2,508

**Table 5.11:** DBPedia: Runtime of the different datasets (DBP = DBPedia extract, ID = initial dataset, WS = wrong statements).

constraints. We listed reasons for this observation on page 73. The precision is similar for both types of statements. Thus, the F-measure of the temporal annotated statements is more than 10 percentage points higher.

### Runtime

The total time required to process the DBPedia extract is just below 7 minutes (see Table 5.11). However, the runtime decreased slightly ( $\approx 10$  seconds) after we removed the 359 statements causing inconsistencies and applied the application to the initial dataset. This indicates that solving the conflicts takes additional time. The difference would be bigger if the DBPedia extract would have contained a higher degree of wrong statements. However, the runtime increases as we add wrong statements that cause a bigger and more complex problem. Moreover, we can report that this use case is more difficult than standard RDF(S) reasoning (see Section 5.1.2) as it requires not only to apply the rules to infer statements but also to resolve conflicts. Additionally, the occurrence of temporal annotated statements and weighted statements makes this use case more complex. It takes  $\approx 7$  minutes (compared to  $\approx 3.75$ ) when the dataset contains 200k statements and  $\approx 18.3$  minutes (compared to  $\approx 9.5$ ) when it contains 313k statements to compute the MAP state. However, the results show that our application is able to process a dataset containing at least 2,500 intervals as well as over 300k weighted statements that cause many conflicts in a reasonable amount of time.

### 5.2.4 Discussion

We used a dataset derived from DBPedia in order to evaluate if our approach correctly resolves inconsistencies in a temporal probabilistic knowledge base. We applied our application to the original dataset in the first part of the experiments. Despite the fact that not all removed statements are erroneous statements, we can report that our application acts as intended in all situations. This leads to the conclusion that it is necessary that a domain expert checks areas of the dataset in which our applications detects inconsistencies. However, even an unsupervised approach leads to reasonable results. The DBPedia extract contains only a very small share

(< 0.5%) of statements that violate the defined constraints. Thus, we extended the dataset with generated statements in order to investigate how well our application handles a higher degree of wrong statements. Moreover, we varied the weight of the added statements in order to make them more explicit to the reasoner. The results of the experiments indicate that our application detects erroneous statements with a very high precision if their weight is lower than that of the correct statements. The precision drops if all statements have a similar weight as this leads to situations in which the reasoner cannot make a founded decision. This is caused by the fact that neither the dataset nor the defined constraints allow fine-grained reasoning. However, the F-measure of the dataset also improves when all statements have a similar weight. Furthermore, the experiments show that our application is still able to achieve good results when the dataset contains a high percentage of wrong statements. In fact, the results are to a certain degree independent from the share of wrong statements in the dataset. This shows that the application is scalable with respect to an increasing amount of inconsistencies in the dataset. Moreover, it is necessary that the interaction of erroneous statements with other statements leads to inconsistencies according to the defined constraints. This is not always given in the used dataset as the connectivity is relatively low and the data is not very fine-grained. In consequence of that, the recall of the removed statements is well below 100% for all test cases. We were only able to define a few hard constraints. However, the number of required constraints would be much higher if we do not exploit the RDF(S) reasoning capability of our approach. The observed run-times indicate that our application can handle more complex use cases (e.g., higher connectivity and also soft constraints) of at least comparable size. So, we showed that our approach improves the precision while hardly decreasing the recall of a probabilistic knowledge base containing many temporal facts.

## 5.3 Sensor Data

Another use case provides a sensor data dataset that is used to evaluate activity recognition algorithms. At first view, this use case seems to be unrelated to our approach but its characteristics make it valuable for our evaluation. It contains temporal annotated statements that can be validated using temporal constraints. Moreover, an ontology that describes the relations between the statements by defining weighted axioms was proposed by Helaoui et al. [2013]. The axioms contained in this ontology can be transformed to temporal constraints. Beside the data and the rules, there is also a gold standard that can be used for objective assessment. So, this dataset does not only fulfill all requirements to apply our application but also allows us to show that our approach is domain independent.

### 5.3.1 Dataset & Constraints

The chosen dataset was collected in a real-life scenario in the context of the EU research project “Activity and Context Recognition with Opportunistic Sensor Configurations”<sup>13</sup>. The data was collected in a smart room simulating a studio flat that was equipped with multiple sensors that recorded the activities of users carrying out morning routines with the focus on maximizing the activity primitives [Lukowicz et al., 2010]. The sensors are able to detect the locomotion (e.g., stand, lie) and the body gesture (e.g., move, release) of the user as well as the interaction with certain objects (e.g., a knife, a bottle). The combination of body gestures and objects leads to atomic gestures like “move knife”. A timestamp and the duration got stored for each detected atomic gesture and locomotion. Moreover, several atomic gestures got aggregated to a complex activity. However, we are using the annotated dataset<sup>14</sup> introduced by Helaoui et al. [2013] that contains two additional intermediate levels. This dataset comprises the sensor data of three users (S10, S11 and S12) that execute three different routines (ADL1 – 3).

The dataset comprises four different levels to which the gestures and activities are assigned according to their duration and dependencies on other activities. Higher level activities have a longer duration because they are composed of multiple lower level activities. The different levels of the dataset have the following characteristics:

---

<sup>13</sup><http://www.opportunity-project.eu>

<sup>14</sup><http://webmind.dico.unimi.it/care/annotations.zip>

**Atomic gestures (AG, Level 4)** have a very short duration and cannot be decomposed in simpler ones (e.g., reach knife). However, it is possible that a subject executes different gestures in parallel as her/his hands can independently interact with objects.

**Manipulative gestures (MG; Level 3)** last only very few seconds and depend on atomic gestures (e.g., fetch knife). Multiple Level 3 gestures can be executed in parallel.

**Simple activities (SA, Level 2)** are temporal sequences of manipulative gestures (e.g., prepare salami includes reach knife, cut salami and release knife). However, a specific simple activity can depend on different manipulative gestures. The typical duration is a few seconds.

**Complex activities (CA, Level 1)** are concurrent executions of simple activities and can last from a few minutes to hours (e.g., sandwich time can depend on prepare salami and other activities).

Hence, the activities are modeled in a hierarchical structure which includes temporal dependencies. An overview on the activities existing in the dataset is presented in Table 5.12. The relations between the activities are modeled in a multilevel activity recognition ontology<sup>15</sup> based on the idea that more complex activities are based on simpler one. Atomic gestures (Level 4) got directly inferred from the data of the sensors based on interactions with certain objects. Hence, these activities are observed which means that statements using Level 4 activities are explicitly true or false. Manipulative gestures depend on atomic gestures and the semantic context of the activities (e.g., an object can be moved to put it down or to fetch it). Therefore, they [Helaoui et al., 2013] model the gestures in a probabilistic ontology and resolve possible inconsistencies by computing the most probable consistent ontology using the reasoner ELOG [Noessner and Niepert, 2011]. This ontology is then used to infer the manipulative gestures by standard subsumption and equivalence reasoning. Similarly, they computed the axioms for the simple activities, which depend on a sequence of manipulative gestures, and complex activities, which depend on simple activities. Finally, they assign manually weights (i.e., confidence scores) to the axioms based on common sense knowledge and observation of the data. The confidence scores are in the range from 0.15 to 1.00 which makes it possible to interpret them as a probability or weight of a constraint.

The resulting axioms can be used to extract rules that allow inferring activities on higher levels. Thus, we will provide an example for each level and explain

<sup>15</sup>[http://webmind.dico.unimi.it/care/multilevel\\_activities.owl](http://webmind.dico.unimi.it/care/multilevel_activities.owl)

	<b>Locomotion, Gesture, Activity</b>
<b>Locomotion</b>	Lie, Sit, Stand, Walk
<b>Level 4</b>	Open, Close, Reach, Release: Dishwasher, Door(1,2), Drawer(1,2,3), Fridge
	Move, Reach, Release: Bottle, Bread, Cheese, Milk, Salami, Sugar, Chair, LazyChair, WashableObject(Cup, Glass, Knife(Cheese, Salami), Plate, Spoon)
	BiteBread, CutBread, CutSalami, SipCup, SipGlass, Spread-Cheese, StirSpoon, UnlockDoor(1,2), LockDoor(1,2), ReleaseSwitch, ReachSwitch, CleanTable, ReachTable, ReleaseTable
<b>Level 3</b>	Putdown, Fetch: Bottle, Bread, Cheese, Milk, Salami, Sugar, Chair, LazyChair, WashableObject (Cup, Glass, Knife(Cheese, Salami), Plate, Spoon)
	Open, Close: Dishwasher, Door(1,2), Drawer(1,2,3), Fridge
	InteractwithChair, InteractwithLazychair, SwitchSwitch, CleanTable
<b>Level 2</b>	Get, Putaway: Bottle, Milk, Cheese, Salami, Bread
	PrepareSalami, PrepareCheeseSandwich, PutSugar, GetKnifeSalami, GetKnifeCheese, GetPlate, EatBread, DrinkfromCup, DrinkfromGlass, LieonLazychair
<b>Level 1</b>	Cleanup, CoffeeTime, SandwithTime

**Table 5.12:** Sensor Data: Overview on the activities.

how we interpret it. An activity recognition system can only rely on the locomotion and the atomic gestures as its purpose is to infer the activities of the levels 1–3.

**Level 3:** Manipulative gestures can be directly inferred from atomic gestures. The following example expresses that `FetchKnifeSalami` can be inferred from the atomic gesture `ReachKnifeSalami`.

$$\begin{aligned}
 \text{MGFetchKnifeSalami} \sqsubseteq & \text{ManipulativeGesture} \\
 & \sqcap \exists \text{hasMGAActor} . (\text{Person} \\
 & \sqcap \exists \text{hasAtomicGesture} . \text{ReachKnifeSalami}) \\
 & [\textit{confidence} : 0.9]
 \end{aligned} \tag{5.4}$$

**Level 2:** Simple activities are defined as a sequence of lower level activities, mostly Level 3 activities. For example, the sequence of the manipulative gesture `FetchKnifeSalami`, the atomic gesture `CutSalami` and the manipulative gesture `PutdownKnifeSalami` identifies the simple activity `PrepareSalami`. Note that the order of the activities is important in order to match the axiom. They use the property `hasOrder` in order to express the sequence in the class description. An activity related to a higher order has to occur directly before the activity having the next lower order as the property models the memory of a subject.

$$\begin{aligned}
 \text{PrepareSalami} \sqsubseteq & \text{SimpleActivity} \\
 & \sqcap \exists \text{hasMemory} . (\text{Memory} \sqcap \exists \text{hasOrder} = 1 \\
 & \sqcap \exists \text{hasMG} . \text{PutdownKnifeSalami}) \\
 & \sqcap \exists \text{hasMemory} . (\text{Memory} \sqcap \exists \text{hasOrder} = 2 \\
 & \sqcap \exists \text{hasAG} . \text{CutSalami}) \\
 & \sqcap \exists \text{hasMemory} . (\text{Memory} \sqcap \exists \text{hasOrder} = 3 \\
 & \sqcap \exists \text{hasMG} . \text{FetchKnifeSalami}) \\
 & [\textit{confidence} : 1.0]
 \end{aligned} \tag{5.5}$$

**Level 1:** Complex activities follow from simple activities. For instance, the simple activity `PrepareSalami` happens during the complex activity `SandwichTime`.

$$\begin{aligned}
 \text{SandwichTime} \sqsubseteq & \text{ComplexActivity} \\
 & \sqcap \exists \text{hasCAAActor} . (\text{Person} \\
 & \sqcap \exists \text{hasSimpleActivity} . \text{PrepareSalami}) \\
 & [\textit{confidence} : 0.9]
 \end{aligned} \tag{5.6}$$

Hence, the ontology provides two different types of constraints. The activities of Level 3 and Level 1 can be directly inferred by applying a simple rule, e.g.:

$$\text{PrepareSalami} \Rightarrow \text{SandwichTime} \quad (5.7)$$

The rules related to simple activities (Level 2) are more complex as they rely on a specific sequence of activities, e.g.:

$$(\text{FetchKnifeSalami}, \text{CutSalami}, \text{PutdownKnifeSalami}) \Rightarrow \text{PrepareSalami} \quad (5.8)$$

Additionally, the ontology provides class disjointness axioms that define which activities cannot happen at the same time. In general, this comprises different activities on the same level that describe contrary interactions with the same object (e.g., fetch/putdown (Level 3) or get/put away (Level 2)). Moreover, only one complex activity can hold at a time. So, this dataset provides a large set of constraints which makes it a valuable use case for our application. We will describe how we express this use case with our formalism in the next section.

### 5.3.2 Data Model

In the previous section, we outlined the characteristics of the dataset. The next step is it to transform it to a model that fits our application. This includes two parts: First, we introduce a RDF model for the data. Second, we explain how we transform the constraints. However, in order to provide a better understanding of the chosen constraint design, we need to explain some preprocessing steps beforehand.

#### RDF Model

We follow the approach of Helaoui et al. [2013] and introduce five object properties to connect a person (i.e., a subject) to a locomotion, a gesture or an activity (see Table 5.13<sup>16</sup>). Additionally, we annotate each statement with an interval, i.e., the points in time when the statement holds. Thus, each statement describing an activity of a user is annotated with at least one interval.

The following RDF statements express that the subject S10 executes the simple activity “prepare salami” twice during the complex activity “sandwich time” ( $t1 < t2 < t3 < t6 < t7 < t10$ ):

```
sdOnt:S10 sdOnt:hasLevel1Activity sdOnt:CASandwichTime. {[t1,t10]}
sdOnt:S10 sdOnt:hasLevel2Activity sdOnt:SAPprepareSalami. {[t2,t3], [t6,t7]}
```

<sup>16</sup>“sdOnt” is an arbitrary namespace.

Property	Domain	Range
sdOnt:hasLocomotion	Person	locomotion
sdOnt:hasGesture		atomic gesture
sdOnt:hasLevel3Activity		manipulative gesture
sdOnt:hasLevel2Activity		simple activity
sdOnt:hasLevel1Activity		complex activity

**Table 5.13:** Sensor Data: Properties used to model the activities.

This representation is sufficient for all Level 1–3 activities. Contrary, we need to explicitly model statements describing the atomic gestures (`sdOnt:hasGesture`) or the state of locomotion (`sdOnt:hasLocomotion`) as observed statements. This is necessary as those statements represent the input information of the activity recognition and we do not want to infer statements using these properties. The following example expresses that the user S10 “reaches the salami knife” twice while standing:

```
sdOnt:S10 sdOnt:hasLocomotion sdOnt:Stand.           {[t1,t10,true]}
sdOnt:S10 sdOnt:hasGesture sdOnt:MGRreachKnifeSalami. {[t2,t3,true],
                                                         [t6,t7,true]}
```

Hence, this use case relies on different possibilities to express temporal annotated statements supported by our formalism. It shows that all of the respective features are useful and enable a wide range of possible applications.

### Data Preprocessing

The used dataset splits up the recorded data into four files. Each file describes the recorded activities at a different level. The three files describing the activities of Level 3, Level 2 and Level 1 provide the gold standard. These files contain the start and end time of an executed activity in milliseconds:

```
1213421 1258153 CACleanup
1280587 1307487 CARelaxing
```

The fourth file contains the atomic gestures and the states of locomotion of a user and serves as input of the activity recognition. The format of this file is different from the gold standard. It provides the start time of a gesture/locomotion in milliseconds, the duration of the gesture/locomotion in seconds, the locomotion, the gesture of the left hand as well as the gesture of the right hand:

1247921	10.131	Stand	null	CleanTable
1258154	0.759	Stand	null	null
1258921	1.782	Walk	null	null
1260721	1.089	Stand	null	null
1261821	3.729	Walk	null	null
1265587	0.231	Walk	ReachDoor2	null
1265821	0.330	Stand	ReachDoor2	null

This small excerpt does already show some problems which make preprocessing necessary. First, the duration of the gestures varies considerably and is very precise. This is problematic as we would end up with too many different intervals. Moreover, it can falsify the evaluation as gestures that last longer have a stronger influence on the result despite the fact that the constraints do not consider the length of a gesture. Another problem is that gestures are split into multiple intervals when the state of locomotion changes. This is inconvenient as the locomotion is not used in the respective constraints. In order to circumvent those difficulties, we decided to group all gestures that start within one second. Moreover, we group all sequences that do not contain a gesture. This step reduces the number of intervals without the loss of meaningful information. The result of these preprocessing steps is an ordered sequence of  $n$  intervals that contain multiple activities which we can enumerate from 1 to  $n$ . The previous example can be transformed to this representation:

1	1	Stand	null	CleanTable
1	1	Stand	null	null
2	2	Walk	null	null
2	2	Stand	null	null
2	2	Walk	null	null
3	3	Walk	ReachDoor2	null
3	3	Stand	ReachDoor2	null

The second column contains the end of the interval instead of the duration. We use intervals with the same start value and end value as we follow an event-based approach in which the duration of an interval does not matter. It is also not problematic that there are blocks with no gestures and different states of locomotion. In fact, we ignore those blocks when we assess the quality of our approach as they do not provide relevant information. The aggregation can cause that multiple gestures are assigned to the same interval. This is not problematic as long as the gestures are not mutual exclusive but it needs to be considered while designing the constraints. However, even the original data describes up to two activities at the same time as gestures are recorded for both hands. Thus, the aggregation does not introduce a

new difficulty but it increases the likelihood of such a scenario. Finally, we transform the data to temporal annotated RDF triples (`true` associated to an interval indicates that the respective statement is observed):

```
sdOnt:S10 sdOnt:hasLocomotion sdOnt:Stand.
                                     {[1,1,true], [2,2,true],[3,3,true]}
sdOnt:S10 sdOnt:hasLocomotion sdOnt:Walk.      {[2,2,true]}
sdOnt:S10 sdOnt:hasGesture     sdOnt:MGCleanTable. {[1,1,true]}
sdOnt:S10 sdOnt:hasGesture     sdOnt:MGCleanTable. {[3,3,true]}
```

Please note that preprocessing the data also enables that we can compare our results to the results of Helaoui et al. [2013] as their application processes the data in a similar way.

### Constraints

In order to detect activities based on the sensor data, i.e., locomotion and atomic gestures, we need to model rules that describe the relations between the activities on the different levels. Therefore, we primarily rely on the constraints detected by Helaoui et al. [2013] but also add additional rules. Moreover, we add constraints that are required by our approach like the aggregation of interval relations.

**Aggregation of interval relations.** The interval relations introduced by Allen [1983] are too specific in this context. Thus, we need to introduce two new relations that aggregate some of those. The first relation is called `sdOntDuring` and is used to compare the interval of a lower level activity to the interval of a higher level activity. The higher level interval should contain the intervals of all related lower level activities. Hence, it aggregates the relations `tDuring`, `tStarts`, `tFinishes` and `tEqual` (see Section 4.3). The second relation is called `sdOntBefore` and is used to model a sequence of activities which is required to infer Level 2 activities. It aggregates the relations `tBefore`, `tMeets`, `tOverlaps` and `tEqual`. So, we need eight constraints of the following form:

$$tBefore(i1, i2) \Rightarrow sdOntBefore(i1, i2)$$

One could argue that a few aggregations are not required and cause an unreasonable overhead. We can weaken this argument as we only have to deal with a small number of disjoint intervals ( $< 500$ ) which makes it completely unproblematic to ground all of those constraints. Moreover, it shows the benefits of the possibility to aggregate intervals. For instance, without this feature we would end up with a higher number of complex constraints. For example, a rule that models

the sequence of four activities needs to describe three interval relations (i.e.,  $i1 - i2$ ,  $i2 - i3$ ,  $i3 - i4$ ). If we were not able to introduce the new relation `sdOntBefore`, we would end up with  $3^4 = 81$  constraints instead of one constraint using the new interval relation. This causes not only much more overhead than computing the new relations but it may even lead to a model that is too big to be processed by a Markov Logic solver. It also is more convenient to maintain a smaller number of constraints. Hence, we need to rely on the interval relations `sdOntBefore` and `sdOntDuring`.

**Simple constraints.** The first type of activity recognition rules models the dependencies between Level 4 and Level 3 as well as between Level 2 and Level 1 (see Equation 5.7). The correct model of those would be:

$$\begin{aligned} & \text{quad}(s, \text{"hasL2A"}^{17}, \text{"PrepareSalami"}, i1) \wedge \text{sdOntDuring}(i1, i2) \\ & \Rightarrow \text{quad}(s, \text{"hasL1A"}, \text{"SandwichTime"}, i2) [\textit{weight} : 90] \end{aligned}$$

However, we can simplify this formula as we only rely on the intervals provided by the sensor data because we are not able to infer overarching intervals (see Section 4.3). In consequence of that, we can directly assign the interval of the lower level activity to the higher level activity.

$$\begin{aligned} & \text{quad}(s, \text{"hasL2A"}, \text{"PrepareSalami"}, i1) \\ & \Rightarrow \text{quad}(s, \text{"hasL1A"}, \text{"SandwichTime"}, i1) [\textit{weight} : 90] \end{aligned}$$

It is noteworthy that we use the predicate `quad0` for statements having the property `sdOnt:hasGesture` or `sdOnt:hasLocomotion` to prevent inferring statements that use these properties.

---

<sup>17</sup>We abbreviate the properties listed in Table 5.13 in order to keep the formulas shorter.

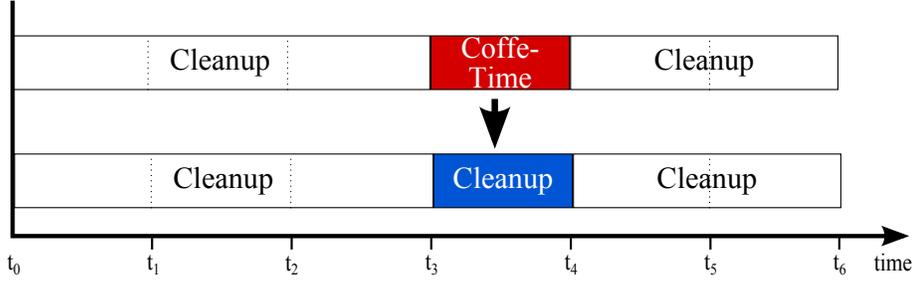
**Complex constraints.** The rules that are required to infer activities on Level 2 are more complex (see Equation 5.8). We model them in the following way:

$$\begin{aligned}
& \text{quad}(s, \text{"hasL3A"}, \text{"FetchKnifeSalami"}, i1) \wedge \\
& \quad \text{quadO}(s, \text{"hasAG"}, \text{"CutSalami"}, i2) \wedge \\
& \text{quad}(s, \text{"hasL3A"}, \text{"PutdownKnifeSalami"}, i3) \wedge \\
& \quad \text{sdOntBefore}(i1, i2) \wedge \text{sdOntBefore}(i2, i3) \\
& \quad \Rightarrow \\
& \text{quad}(s, \text{"hasL2A"}, \text{"PrepareSalami"}, i1) \wedge \\
& \text{quad}(s, \text{"hasL2A"}, \text{"PrepareSalami"}, i2) \wedge \\
& \quad \text{quad}(s, \text{"hasL2A"}, \text{"PrepareSalami"}, i3) \\
& \quad [weight : 100]
\end{aligned}$$

This rule does not exactly correspond to the intention of the initial axiom. One problem is that activities on the left-hand side of the rule must not only be in a specific sequence but also must directly follow each other, i.e., the sequence has to be cohesive. In theory, we could use the interval relation  $t_{Meets}$  to model this but this would not work very well in practice. This has several reasons:

1. It is possible that multiple activities of a sequence are assigned to the same interval.
2. One of the activities partially overlaps within another activity of the respective sequence.
3. The sequence can be interrupted by short phase for which no gesture or an unrelated gesture is registered.

Hence, we cannot use the relation  $t_{Meets}$  which is the only relation that considers the temporal distance between two intervals. The chosen rule design addresses these problems by using an aggregated interval relation. However, it has the disadvantage that it also matches activities that are unrelated as the rules do not consider their distance of time. We address this issue by applying our application only to statements covering a small time frame (see Section 5.3.3). As already mentioned in the context of the simple constraints, we cannot infer overarching intervals. So, we cannot formulate a constraint that introduces an interval  $i4$  that overarches the other intervals (i.e.,  $i1, i2, i3 \text{ sdOntDuring } i4$ ) and infer the axiom



**Figure 5.4:** Sensor Data: Continuous complex activities. This figure illustrates that a sequence of complex activities can be interrupted by another activity for a short period of time. The constraints counteract such scenarios.

`quad(s, hasL2A, PrepareSalami, i4).`

In order to transform the previous constraint to the conjunctive normal form, we need to split it up into three rules as we cannot have a conjunction of literals on the right-hand side of implication. Each of the resulting constraints has the same left-hand side but has only one literal from the right-hand side. The weight of each constraint is one-third of the weight of the original formula ( $100/3 = 33.33$ ).

**Continuous complex activities.** Complex activities represent the highest level to describe the activity of a subject. They are characterized by the execution of multiple simple activities which causes durations in the range from a few minutes to a few hours. The previously introduced rules do not consider the context in which a simple activity is carried out. Hence, it is possible that only one simple activity leads to a different complex activity than the previous and following one (see Figure 5.4). In order to block the respective rule, we introduce constraints that ensure a continuous assignment of a complex activity:

$$\begin{aligned} & \text{quad}(s, \text{"hasL1A"}, \text{"SandwichTime"}, i2) \wedge \text{sdBefore}(i1, i2) \\ & \Rightarrow \text{quad}(s, \text{"hasL1A"}, \text{"SandwichTime"}, i1) [\textit{weight} : 30] \end{aligned}$$

$$\begin{aligned} & \text{quad}(s, \text{"hasL1A"}, \text{"SandwichTime"}, i1) \wedge \text{sdBefore}(i1, i2) \\ & \Rightarrow \text{quad}(s, \text{"hasL1A"}, \text{"SandwichTime"}, i2) [\textit{weight} : 30] \end{aligned}$$

These constraints express that all upcoming respectively all previous intervals should have the same complex activity. We decided against using the interval relation `tMeets` as the relation `sdOntBefore` makes the intention of the constraints more obvious and would also work for the original (not preprocessed) dataset. In order to restrict the influence of the relation `sdOntBefore`, we will use a window within we predict the activities (see Section 5.3.3). Moreover, we could have replaced the complex activity by a variable but decided to define two rules for each complex activity as there are only four of them.

**Disjoint activities.** The next set of constraints ensures that certain activities cannot happen at the same time. This comprises certain pairs of activities, e.g., “open door” and “close door”, but also all complex activities are mutually exclusive. The respective rules have the following form:

$$\begin{aligned} \text{quad}(s, \text{“hasL1A”}, \text{“SandwichTime”}, i1) \wedge \text{sdOntDuring}(i1, i2) \\ \Rightarrow \text{!quad}(s, \text{“hasL1A”}, \text{“Cleanup”}, i2) \end{aligned}$$

In fact, we express that the property `sdOnt:hasLevel1` activity is functional with respect to a specific point in time by introducing constraints that make all complex activities mutually exclusive. We considered adding similar soft constraints between all activities on each level as it is unlikely that a subject executes multiple activities of the same level at the same time. However, this would lead to more than 2.000 additional constraints. Instead, we introduce another set of constraints which we explain in the following.

**No activity.** The final set of constraints is tailored to our approach. In order to detect wrongly recognized activities, we need violated constraints. The previously introduced constraints (beside the disjointness constraints) can only infer activities but do not directly express which activities cannot follow from certain lower level observations. Even the rules that ensure disjointness are not able to remove all activities in an interval. Hence, we introduce for each activity a rule that is always violated. The union of all constraints of this type express that a subject does nothing if there is no evidence for a specific activity. So, we are able to detect all statements that do not follow from an activity recognition rule:

$$\text{!quad}(s, \text{“hasL1A”}, \text{“SandwichTime”}, i1) [\textit{weight} : 1.1]$$

Level	Activities	Disjoint Pairs	Constraints
3	42	19	38
2	21	5	10
1	4	6	12
$\Sigma$	67	30	60

**Table 5.14:** Sensor Data: Number of activities per level and number of disjoint activities. Hence, we need 67 “no activity” constraints and 60 hard constraints to model mutual exclusive activities.

**Number of constraints and weights.** As previously mentioned, we define for each possible activity a constraint having a weight of 1.1 that contradicts the activity. This leads to 67 “no activity” soft constraints (see Table 5.14). In order to detect inconsistent statements, we give all statements a weight of 1.0. Hence, each statements has a weight of  $-0.1$  excluding the other constraints. This gives most of the power to the 637 activity recognition constraints (i.e., simple and complex constraints). Table 5.15 shows how the constraints are distributed in terms of their length. Actually, the most rules are simple (length = 1) but when we transform all rules to the conjunctive normal form rules relying on a sequence of activities require a similar number of constraints. The weights of the activity recognition rules range from 10 to 100 with an average of 68.62 and a standard deviation of 27.28 which make them more relevant than the “no activity” rules. Moreover, we define 8 constraints that ensure that complex activities are not interrupted by single activities that do not correspond to the overarching complex activity. Each rule of these rules has a weight of 30 which ensures that one complex activity, surrounded by two different identical complex activities, gets their weight reduced by 60 which is just below the average weight of the activity recognition rules. However, the weights of all constraints depend only on common knowledge and observation of the data. Hence, learning them could lead to better results. We also create 60 hard constraints to model mutual exclusive activities (see Table 5.14) and 8 hard constraints that define the used temporal relations. So, we model this use case with a total of 780 constraints of various complexity and type. We deactivate the RDF(S) reasoning for this use case as none of the constraints relies on it.

### 5.3.3 Experiments

We introduced a RDF model for a sensor data dataset that can be used for activity recognition in the previous section. Due to the design of the constraints, we can either follow an approach that infers all the activities based on the input data,

Length	Count	Constraints
1	175	175
2	135	270
3	44	132
4	15	60
$\Sigma$	–	637

**Table 5.15:** Sensor Data: Number of the activity recognition rules grouped by the length of the sequence of the lower level activities. The weights of those rules range from 10 to 100 with an average of 68.62 and a standard deviation of 27.28.

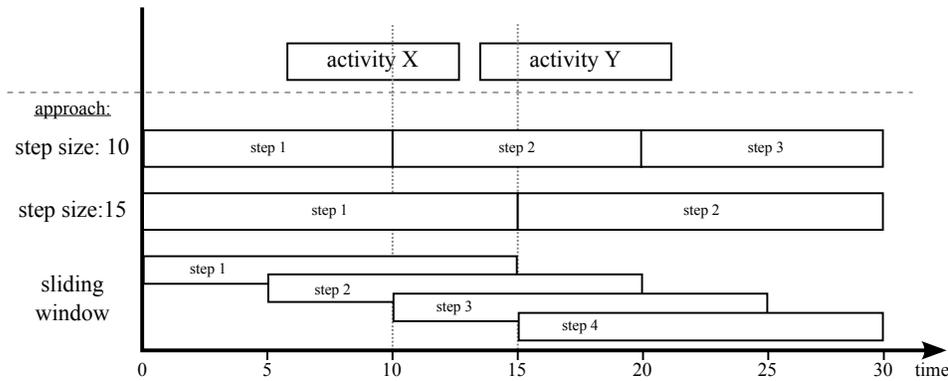
	S10	S11	S12
ADL1	366	464	465
ADL2	321	347	386
ADL3	271	404	379

**Table 5.16:** Sensor Data: Number of intervals per test case.

i.e., statements describing the atomic gesture and the locomotion of the subject, or detect statements that violate the constraints in a document that already contains activities of the higher levels. In the first scenario, the input of our application contains no statements describing activities of Level 3 and higher. In the second scenario, the input of our application contains the observed atomic gestures and the different states of locomotion as well as all possible high level activities (Level 3 – 1) for each interval. Both approaches should lead to same results but possibly different runtimes. Hence, we follow the latter approach as this corresponds to the intention of our application which is to remove erroneous statements from a dataset. In particular, we create for each interval of the respective test case all possible activities (67 per interval (see Table 5.14)). The different test cases average 378 intervals (see Table 5.16). This leads to over 25,000 potentially wrong statements per test case. In combination with the 780 constraints, the problem is too big to be solved at once. Thus, we apply a domain-specific segmentation into subproblems.

### Segmentation

We rely on a domain-specific approach to split a test case into smaller subproblems as it is not feasible to compute the complete solution at once due to the high number of conflicting constraints. In order to determine the activities, we only need to



**Figure 5.5:** Sensor Data: A lower step size leads to a higher fragmentation which hinders the activity recognition. Increasing the step size decreases the chance of fragmentation but may lead to worse results. Hence, we also apply a sliding window approach in order to use a small step size and decrease the chance of fragmentation. This figure does only illustrate the idea as all pictured step sizes are rather small.

consider the time frame in which the activity is actually carried out (see Figure 5.5). Hence, there is no benefit to compute all activities of a test case at once. We also expect better results by computing the activities for smaller time frames as this fits better to the design of our rules. Therefore, we validate the activities step by step in disjunctive subproblems (e.g., 1 – 10, 11 – 20, ...). This approach introduces the problem that activities get fragmented into different subproblems. For instance, if we choose a step size of five and the average length of a simple activity is three then there is a 40% chance that the activity does not fall in a single step. Thus, the steps cause borders that hinder the correct detection of simple activities. We can circumvent this problem by determining the activities in a sliding window (e.g., 1 – 10, 2 – 11, ...). However, this approach is not very efficient as we compute the activities for each interval multiple times. Moreover, it takes too much time to compute the optimal step size, which we need to do first, using the sliding window approach. Selecting the best step size is not trivial:

**small step size:** A smaller step size fits better to the design of our constraints as we cannot model cohesive sequences of activities. Hence, a small step size decreases the probability that a constraint matches unrelated activities. However, fragmentation of activities is probable and can cause that some activities cannot be recognized. Moreover, a too small step size leads to a longer total runtime as the application executes identical parts more often (e.g., loading the constraints).

**large step size:** A larger step size has the advantage that it is less probable that activities get fragmented. It might be a problem that a step contains multiple simple activities as the respective rules could match unrelated lower level activities. However, this is not necessarily a problem if the section comprises simple activities that rely on different lower level activities. A large step size ensures that the time range of a complex activity is not interrupted by a different complex activity. It also causes that phases in which no complex activity hold get an activity assigned which might reduce the precision.

Hence, we need to test different step sizes in order to see which leads to the best results. Once we have determined a good step size, we will apply the sliding window approach. In particular, we want to answer the following questions:

- **SD-Q1:** Which step size leads to the best result in terms of precision, recall and F-measure?
- **SD-Q2:** What is the effect of the step size on the total runtime? How fast get the subproblems solved?
- **SD-Q3:** How differentiates the runtime between the “remove wrong” and the “infer new” statements approach?
- **SD-Q4:** Can the sliding window approach improve the results?

## Results

**SD-Q1: Precision and Recall.** We removed the inconsistent statements (activities) from a document using different step sizes (5, 10, 15, 20, 25, 50, 75, 100, 150, 200) and computed the respective precision, recall and F-measure for each level (see Figure 5.6). In general, we observe that all measures decrease for high step sizes ( $> 75$ ). This corresponds to our assumption that the constraints work best for smaller step sizes. We also notice that small step sizes ( $< 10$ ) lead to worse results than medium step sizes. Even though we get the best results for the lowest level (Level 4) using a small step size, higher levels do not profit from it due to a high degree of fragmentation. The precision and recall decreases continuously for Level 3. This is surprising as the respective statements directly depend on the observed Level 4 activities. Hence, the higher level activities implicitly affect the activity detection on Level 3 due to the equivalence  $(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$ . Thus, we get the best results for step sizes in the range from 10 to 30 for which fragmentation of activities is less probable but the F-measure of Level 3 is still high. However, we achieve the best results for Level 1 for relatively large intervals. This shows that the “continuous complex activities” constraints have a positive effect and work as

intended as a phase where a complex activity holds does not get interrupted by another complex activity. However, those constraints have a negative effect on the results of the lower levels as the observed atomic gestures and directly inferable activities do not correspond to the set Level 1 activity. Hence, the predictions for Level 3 and Level 2 are not as accurate anymore.

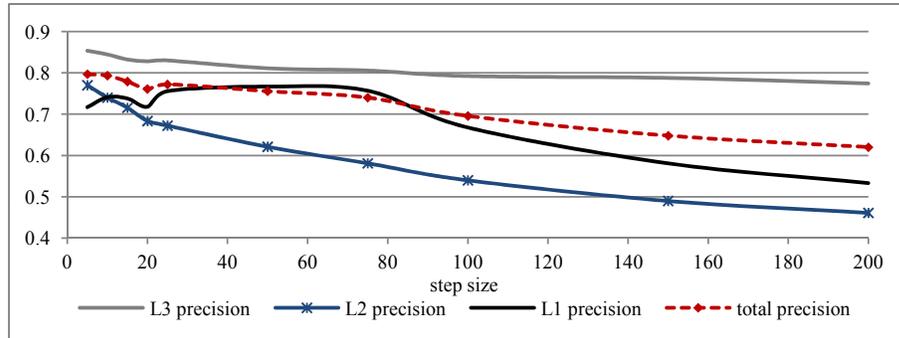
**SD-Q2: Runtime.** We recorded the time that is required to solve one step as well as the total runtime (see Figure 5.7(a)<sup>18,19</sup>). For step sizes smaller than 100 we observed that the time that is required to solve one step is nearly constant ( $\approx 55$  seconds). This is contrary to the expectation as the number of input statements increases linearly (see Figure 5.7(b)). The reason for this is that rockIt needs most of the time for processing the constraints and computing the groundings. In particular, rockIt creates one database table for each constraint and each predicate which sums up to round about 800 database tables in our use case. The actual computation of the MAP state, i.e., solving of integer linear programs, takes only a fraction of the runtime. Hence, the total runtime decreases while we increase the step size for step sizes smaller than 100 because processing a test case using a larger step size requires less steps. However, the computation of the MAP state for larger step sizes takes much more time than initializing the Markov Logic solver. This causes that the total runtime as well as the time to solve one step increases for higher step sizes. We see that the runtime rapidly increases for high step sizes. This indicates that it is not feasible to determine all activities of a test case at once. The reason for this observation are the limitations of the Markov Logic solver, which are caused by the complexity of the problem, as all other parts of the application take together approximately 0.5% of the runtime of the Markov Logic solver (see Figure 5.7(c)). The diagram shows it only for the step size 5 but we measured similar results for small and medium step sizes while the ratio drops to 0.1% for the step size 200. However, this is not an issue as we measure the shortest overall runtime for step sizes that lead to the best activity recognition results.

**SD-Q3: Create vs. Remove.** The design of the constraints allows to detect inconsistent statements as well as to infer new statements. Both approaches should lead to the same results but different runtimes. We conducted the previous experiments providing all possible activities for each interval. Thus, the application could only detect inconsistent (wrong) statements (see Figure 5.7(b)) which is basically a

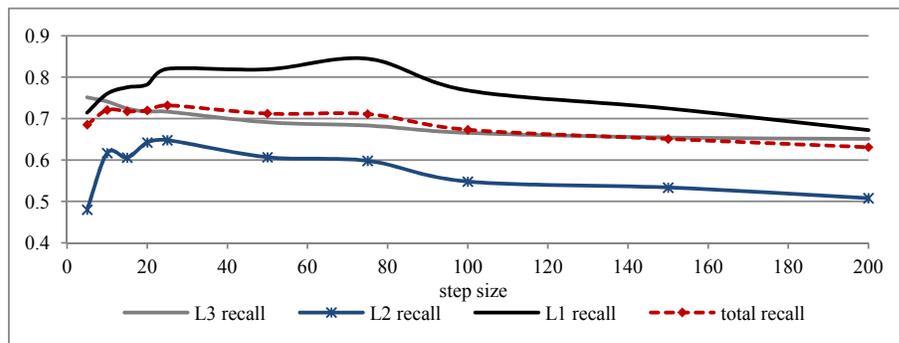
---

<sup>18</sup>We excluded the test case S10-ADL2 as it had a much higher runtime for higher step sizes than all other test cases.

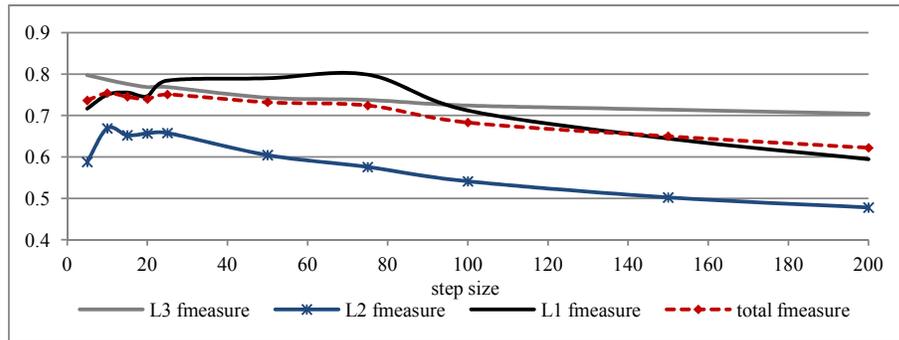
<sup>19</sup>We considered only complete/full steps when computing the average runtime per step size. Hence, we ignored the last step for most test cases. However, we relied on all steps to compute the runtime totals.



(a) precision

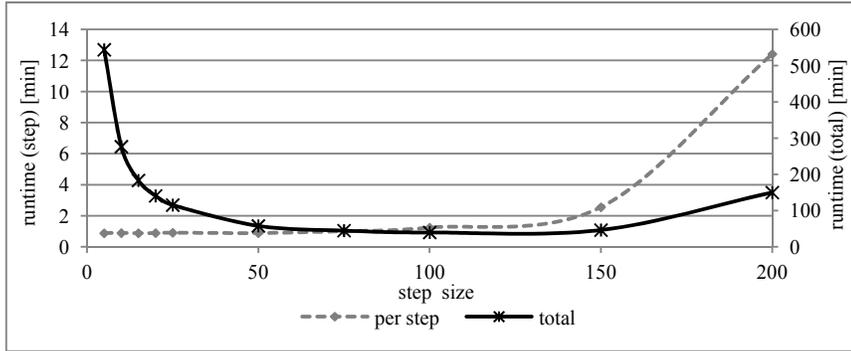


(b) recall

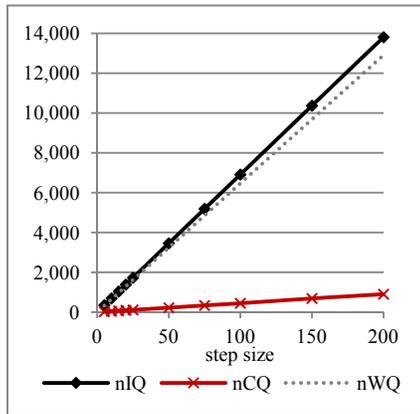


(c) F-measure

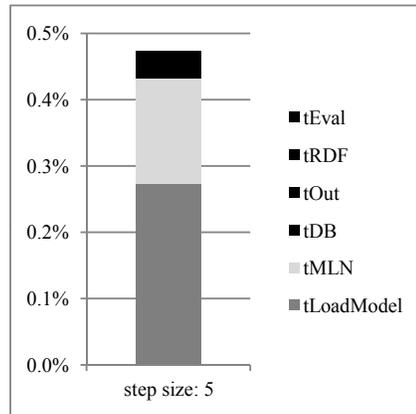
**Figure 5.6:** Sensor Data: Precision, Recall & F-measure depending on different step sizes. All three measures decrease for higher step sizes. Overall, the best result can be achieved for step sizes in the range from 10 to 30. Abbreviations: L3 = Level3, L2 = Level2, L1 = Level1.



(a) runtime per step & total runtime



(b) problem size

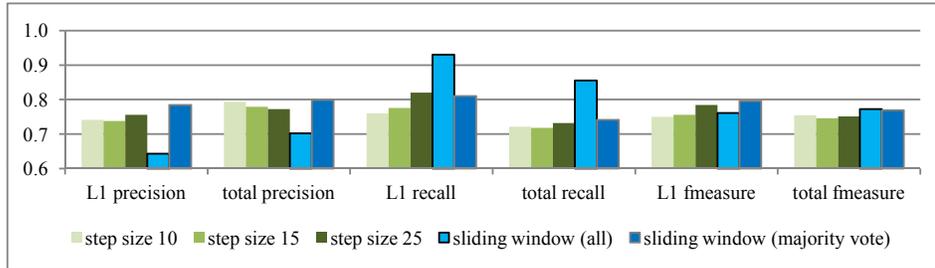


(c) modules

**Figure 5.7:** Sensor Data: The diagram (a) shows the time that is required to solve a single subproblem (left y-axis) as well as the total runtime (right y-axis) for different step sizes (x-axis). The effect of higher step sizes on the total runtime is undervalued in the diagram as the last step can be much smaller than the selected step size (see Table 5.16). The diagram (b) illustrates the problem size for different step sizes. Diagram (c) shows how much time the modules of our application need in relation to the computation of the MAP state (tMAP).

pure data cleansing approach. In order to determine the runtime differences and to validate our assumption that both approaches lead to the same results, we executed all test cases for the different step sizes only providing the states of locomotion and atomic gestures (Level 4). We can confirm that both approaches lead to the same results due to the “no activity” constraints that negate all activities. With respect to the runtime, we cannot measure noticeable differences. For small and medium step sizes ( $\leq 100$ ) we record that the “create statements” approach is on average 1% faster (standard deviation = 3%) than the “remove statements” approach. This supports our previous observation that most of the time is required to initialize the Markov Logic solver. For large step sizes, the “create statements” approach is 5% slower (standard deviation = 30%). However, at least the results for large step sizes are not representative as the sample size is too small. For small step sizes, which lead to the better activity recognition results, we can report that there is no noticeable difference between both approaches.

**SD-Q4: Sliding Window.** Finally, we apply the sliding window approach with the goal to improve the activity recognition. Based on the previous results, we decided to select the step size 15 and slide it 5 steps each time (e.g., 1 – 15, 6 – 20, 11 – 25, ...). This ensures that we consider each activity sequence that is distributed over 11 time points. So, we are in a range where we achieve a higher precision compared to other step sizes and expect to increase the recall by decreasing the chance of fragmentation. Moreover, the step size is long enough to detect Level 1 activities accordingly. Compared to the standard approach, we expect at least to retain the precision of step size 10 and to achieve the recall of step size 30. The chosen approach requires us to combine multiple predictions for each interval as the windows are overlapping. Therefore, we can basically follow an approach that leads to a high recall or one that improves the precision. As we combine only up to three predictions (i.e., overlap of three windows) there is no room to balance between both possibilities. So, we get the highest recall when we take all predictions (see Figure 5.8). The recall for Level 1 raises to over 90% but we do not check if mutual exclusive activities are assigned to the same interval. In order to improve the precision, we consider only activities that occur in the majority of the predictions, e.g., an activity has to occur in at least two of the three overlapping windows. We also take into account that it is possible that none activity is detectable for some intervals. Besides, the majority vote approach ensures that the constraints that enforce mutual exclusive activities are not violated in the final prediction. The resulting precision is very similar to the standard approach for comparable step sizes which indicates that the borders introduced by the steps only affect the recall. Thus, it is reasonable that the recall increases and that the



**Figure 5.8:** Sensor Data: The sliding window approach with a window length of 15 that is slid by 5 compared to the closest step sizes (10, 15 and 25) for Level 1 (L1) and all levels (total). The highest recall is achieved when considering all recognized activities while selecting the activities by majority vote leads to a higher precision.

	step size 10	step size 15	sliding window
total runtime [min]	277	183	535
runtime / step [s]	53.2	52.6	53.1

**Table 5.17:** Sensor Data: The runtime of the sliding window approach compared to the standard approach. The total runtime is higher as more steps are required.

F-measure improves. However, the sliding windows approach takes much more time as more steps are required (see Table 5.17).

### 5.3.4 Discussion

We showed that the formalism developed in this work is not limited to a specific domain (e.g., Linked Data). In particular, we modeled sensor data in order to detect activities based on temporal relations to other activities. Therefore, we needed to rely on different features supported by our application. This includes the aggregation of interval relations as the standard ones are too specific for this use case or the possibility to make certain statements observed (e.g., Level 4 activities). This does not only fit to the characteristics of those statements but improves the activity recognition and reduces the runtime. We defined close to 800 domain-specific constraints of various types (i.e., different complexity and weights). In particular, constraints that describe direct consequences ( $A \Rightarrow B$ ), temporal sequences of events ( $(A, B) \Rightarrow C$ ) and properties that are functional with respect to a certain point in time. We also needed to apply a domain-specific subproblem generation

	Level 3	Level 2	Level 1
<b>precision</b>	-0.02	-0.12	-0.13
<b>recall</b>	-0.08	0.25	0.16
<b>F-measure</b>	-0.07	0.13	0.05

**Table 5.18:** Sensor Data: Comparison to the results of Helaoui et al. [2013]. We subtracted their results from our sliding window approach (majority vote) results. Hence, a positive value indicates that our approach achieved better results.

due the high number and the characteristics of the constraints. The large number of constraints makes the model too big to solve a test case at once. Additionally, smaller time frames (i.e., subproblems) fit better to the design of the constraints as we are not able to model a cohesive sequence of events and the lack of the possibility to infer overarching intervals. We observed that generating too small subproblems has a negative effect on the runtime as long as initializing the Markov Logic solver takes much more time than computing the MAP state. We can report that the limitations with respect to the input size are only caused by the Markov Logic solver. Hence, improvements in this research area make our approach more viable.

We compared our results to the work of Helaoui et al. [2013], as the used constraints are based on their ontology, in order to assess the practicality of our approach. First of all, we need to admit that Helaoui et al. [2013] introduced an application that predicts the activities in real-time while the forecast of our approach is at least delayed by the step size of the sliding window. Their approach is also faster as they take only a small number of facts into account (at most the last 5 seconds) and predict the activities for the different levels one after another (Level 4  $\rightarrow$  Level 3, Level 3  $\rightarrow$  Level 2, ...). Due to the comparable rule set, we expected to achieve similar results. Overall, we get a worse precision on all levels while the recall is noticeable higher for the upper two levels (see Table 5.18). In consequence of that, the F-measure is worse for Level 3 and better for Level 2 and Level 1. The primary reason for this is that we added additional constraints. In order to improve the precision of the predictions for Level 3, we defined constraints that infer Level 3 activities based on sequences of Level 4 activities (comparable to the constraints that are used to infer simple activities (Level 2)). Another or additional way to improve the recall as well as the precision would be the tracking of the status of different objects (e.g., is an object still in the fridge). A higher precision would automatically lead to a higher recall as the decision is often made between two mutual exclusive activities. However, this is not supported by our formalism. For

Level 2, we added constraints that led to a higher recall but worse precision. For instance, if the ontology of Helaoui et al. [2013] contains the constraint  $(A,B,C) \Rightarrow D$  we would additionally add constraints that cover parts of the initial constraint like  $(A,C) \Rightarrow D$  in some cases. To improve the results of Level 1, we added rules that connect Level 4 directly to Level 1 which reduces the precision but also causes a better recall. These additional rules might violate the concept of the levels as they skip intermediate levels but also the gold standard contains intervals for which no activities for the intermediate levels are named. Hence, we needed to add those constraints to be able to predict the respective activities without inferring activities for the intermediate levels. So, all changes made to the constraint set of Helaoui et al. [2013] are reasonable and allow us to compare the results. We achieved considerable results that underline that our formalism and application does not only allow modeling various domains but can actually improve the data quality of the respective datasets. However, it should be possible to improve the results if we learn the weights of the used constraints. Currently, the weights of the constraints depend only on common knowledge and manual observation of the dataset which makes this idea promising.



## Chapter 6

# Related Work

In this chapter, we summarize approaches that are related to our work. In the first part of this chapter, we give a brief overview on existing approaches to model temporal and probabilistic statements using frameworks associated with the Semantic Web. In the second part of this chapter, we present existing approaches to carry out reasoning for such datasets. Moreover, we outline related areas that use Markov Logic in order to deal with temporal data and uncertainty.

### **Part 1: Models for temporal & probabilistic Data**

The standard RDF data model does neither support probabilistic nor temporal facts. We circumvent this limitation by associating the respective information, i.e., validity time and confidence, with facts using reification. A similar approach was proposed by Gutierrez et al. [2005, 2007] who incorporated temporal reasoning into RDF. Therefore, they annotate standard RDF triples with time points or intervals using reification with their own vocabulary.

Lopes et al. [2010] and Zimmermann et al. [2012] developed a RDF based annotation framework for representing, reasoning and querying data on the Semantic Web. In particular, they support statements annotated with probabilities and temporal information. They also introduce operators which consider the probabilities as well as the validity times in order to infer additional statements. Moreover, they extend the RDFS entailment rules with these operators and present an extension of SPARQL for querying RDF with annotations. So, in contrast to our approach, they infer statements that are annotated with probabilities and arbitrary intervals. This is not supported by our application as we are neither able to perform calculations on intervals nor able to compute the probabilities of the inferred statements. However, the data representation model fits our requirements.

Udrea et al. [2006] proposed an approach that integrates probabilities into RDF (pRDF) as well as a more comprehensive approach that focuses on a broader range of annotations, i.e., uncertainty, temporal aspects and provenance, in RDF (aRDF) [Udrea et al., 2010]. Their approach bases on RDF but they extend the syntax and semantics of RDF in order to define a framework that can be used to reason about combinations of time, uncertainty and provenance. Motik [2012] presented a logic-based approach to extend RDF and OWL with temporal reasoning that builds on first-order theories. The approach also includes an extension of SPARQL and optimized entailment algorithms. Moreover, Analyti and Pachoulakis [2012] published a survey on models and query languages of temporal annotated RDF statements. Thereby, they distinguish between works that introduce a new model theory and works that extend RDF simple entailment and RDFS entailment.

The focus of our work was not on developing a data model for temporal and probabilistic facts. However, this topic is relevant for other researchers which makes it possible that a standard for publishing such datasets on the Semantic Web will be defined and implemented in the future. This makes it possible that many datasets that use such a framework will be published. Despite the fact that we use a model that will not adhere to a new standard in order to express the statements, our approach can be applied to datasets using another format after minor adjustments in order to improve the data quality. However, many approaches require computations on intervals (e.g., overlap of two intervals) and determining the probabilities of the inferred statements. These features are not part of our approach.

## **Part 2: Related Reasoning Approaches**

We give an overview on related reasoning concepts in the following. Therefore, we focus on approaches that can process comparable datasets and approaches that use Markov Logic.

### **Part 2.1: Reasoning for temporal Knowledge Bases**

Batsakis et al. [2011] and Anagnostopoulos et al. [2013] extend OWL 2.0 in order to enable temporal reasoning for supporting temporal queries. Therefore, they also explore possibilities to annotate statements with validity times. Moreover, they consider the point algebra as well as Allen's interval algebra which also allow them to infer new facts as well as to detect inconsistencies. The approach of Batsakis et al. [2011] realizes reasoning, i.e., consistency checking and inference over temporal relations, by defining a set of SWRL<sup>1</sup> rules that are compatible to reasoner

---

<sup>1</sup><http://www.w3.org/Submission/SWRL/>

that support DL-safe rules (e.g., Pellet<sup>2</sup>). They also implemented the reasoning system CHRONOS [Anagnostopoulos et al., 2013] that achieves a better performance than the first implementation as it is tailored to their approach. In particular, they separate temporal reasoning from semantic reasoning in order to propose an optimized algorithm. Semantic reasoning is still carried out by Pellet. Temporal reasoning is realized with a path consistency algorithm [Christodoulou et al., 2012] that computes all compositions of existing relations until an inconsistency is detected or a fixed point is reached. Their approach also includes a SPARQL-like temporal query language that incorporates Allen’s interval algebra. So, in contrast to our approach they rely on OWL instead of RDF(S). Moreover, their system is only suited to detect if a knowledge base is inconsistent but it cannot resolve the existing conflicts.

Researcher of the Max-Planck Institute for Informatics (Saarbrücken, Germany) published several papers that are closely related to the content of this work. Their publications cover all steps from extracting facts from the Web to the definition of a temporal-probabilistic knowledge. In particular, they also propose different methods to carry out reasoning in such databases considering first-order logic formulas. All approaches that we describe in the following address the issue that knowledge bases containing temporal statements extracted from the Web are erroneous as the temporal extraction algorithms do not achieve 100% precision. Hence, the precision of a database can be increased by taking temporal constraints into account [Wang et al., 2010a; Dylla et al., 2011, 2013].

**Histogram-based probabilistic knowledge base.** Wang et al. [2010a] present a histogram-based model for time-aware reasoning in probabilistic knowledge bases. The introduced knowledge base consists of facts and first-order logic inference rules. The facts can be encoded in a directed labeled graph (like RDF) and are associated with histograms that capture the validity of a fact at different time points, i.e., their probability distributions. In order to create the histograms, they collect all time points occurring in the dataset and transform them to a discrete series of ordered time points. Therefore, it is necessary that the temporal information is given at a fixed granularity (e.g., years). Hence, based on the ordered time points it is possible to state the weight or probability of a validity interval of a statement in the histogram. They also consider that the initial dataset contains statements to which different validity times and probabilities are assigned. Hence, it is necessary to combine histograms of statements that express the same fact, i.e., facts having

---

<sup>2</sup><http://clarkparsia.com/pellet/>

the same none-temporal part. Therefore, they distinguish between event relations and state relations. Facts of an event relation are only true at a specific point in time while facts having a state relation can be valid at different points in time. This distinction affects the weight distribution in the histogram of a fact as well as coalescing and slicing of intervals when aggregating histograms. For instance, slicing of intervals associated with facts having a state relation is not allowed. However, by merging the histograms they obtain for each statement a probability distribution that indicates when then statement is valid.

In order to apply reasoning, they support first-order logic inference rules. In particular, they focus on first-order formulas that are disjunctions of literals having at most one positive literal (Horn clause [Horn, 1951]). Hence, the rules are comparable to Datalog inference rules. Rules without a positive literal are used to define integrity constraints. Rules with a positive literal can be rewritten as implications and allow to infer new knowledge or to answer queries. They use the temporal relations of Allen's interval algebra to express the logical dependencies of statements. Their system answers queries by considering all possible worlds that do not violate the constraints. Therefore, they use the histograms of the statements to calculate the world with the highest probability.

They implemented their approach as an extension of the RDF reasoning framework URDF [Theobald et al., 2010]. Moreover, they used the temporal knowledge base T-YAGO [Wang et al., 2010b] in order to evaluate their approach. In particular, they investigated the overhead of time-aware query processing compared to a time-oblivious setting. The experiments indicate that the time-histograms cause only a light overhead to a comparable probabilistic setting that does not consider time.

**Resolving conflicts using scheduling.** Dylla et al. [2011] proposed a declarative reasoning framework to resolve temporal conflicts in RDF knowledge bases. Therefore, they define a knowledge base that contains (weighted and temporal) facts and (temporal) consistency constraints. The facts can be annotated with an interval that indicates when it is valid. In order to express the temporal relations of the intervals, they use the relations *before*, *overlap* and *disjoint*. The constraints are expressed in first-order logic. Thus, they define first-order predicates for the temporal relations as well as for all properties occurring in the RDF database (extensional relations). Moreover, they introduce predicates for arithmetic relations, e.g., equal or not equal. In particular, they support only formulas that can be written as a disjunction of literals with at most one positive literal. Hence, the formulas

can be written as implications. The left-hand site of the implication must contain two extensional relations and at most a (non-temporal) arithmetic relation while the right-hand site of the implications is either *false* or a temporal relations. Moreover, they classify the supported constraints as follows: Disjointness constraints ensure that an entity does not occur in two statements having the same extensional relation (predicate) that are annotated with overlapping intervals. Precedence constraints restrict that a specific facts has to be invalid before another fact can be valid. Mutual exclusion constraints express which statements are always (regardless of time) in conflict with each other.

They propose an approach to resolve the conflicts at query time as they consider a knowledge base containing a huge number of facts as well as changing constraints. Thus, they argue that only a dynamic approach is feasible in such a scenario. Hence, the reasoner has to infer the consistent world with the maximum weight at query time. They [Dylla et al., 2011] show that finding a subset of consistent facts contains the Maximum Weight Independent Set problem which is NP-hard [Godsil et al., 2001]. For this reason they propose an approximation heuristic which resolves the conflicts using a scheduling algorithm. Therefore, they map the facts to scheduling jobs and the consistency constraints to scheduling machines. Computing the maximum-weight feasible schedule corresponds to a consistent subset of facts. In particular, they create a constraints graph covering the logical dependencies of the extensional relations. These graphs are covered by machine graphs which represent scheduling machines. At query time they compute a set of facts comprising the matches of the query as well as their closure of conflicting facts. Based on this set, they separately resolve the conflicts by applying the scheduling algorithm for each entity. This is possible as the extensional predicates (representing relations) in a constraint share a variable representing the entity. Hence, they can determine the set of statements that is associated with an entity and relevant to the query. Their experiments only show that their approach performs superior to other heuristics that are related to the Maximum Weight Independent Set problem in terms of runtime and approximation quality.

**A temporal-probabilistic database model.** Dylla et al. [2013] present a temporal-probabilistic database model that allows building high-precision knowledge bases containing facts obtained from information extraction methods. The database contains facts that can be associated with weights as well as with temporal intervals. It supports temporal deduction rules and temporal consistency constraints which are both given as first-order formulas. Deduction rules help to reduce the incompleteness while consistency constraints are required to detect and to resolve incon-

sistencies in the database. Moreover, they extend their model with a query engine that supports queries that are written as a conjunction of literals.

However, the core of their model contains the temporal rules and temporal constraints. Temporal deduction rules are represented as logical implications that are comparable to Datalog rules. The left-hand site of the implication contains a conjunction of at least one positive and optional negative literals that can be grounded with temporal annotated facts. Moreover, it contains a conjunction of arithmetic predicates ( $=$ ,  $\neq$  and temporal relations (Allen' interval algebra)) having arguments that occur in the non-arithmetic predicates of the rule. The right-hand site of the implication contains a single literal that represents the inferred fact. This fact is annotated with an interval whose bounds depend on arbitrary bounds of the intervals occurring on the left-hand site of the implication. Hence, they infer new knowledge and are also able to compute new intervals on demand. By applying the rule it is possible that parts of a validity interval of a fact can be deduced by different rules which lead to duplicated facts in the database. Therefore, they group all intervals of facts which have the same non-temporal arguments.

Temporal consistency constraints are required to detect inconsistencies and are used to condition the marginal probabilities of facts contained in the database. In general, the constraints can be written as a negated conjunction of literals. It has to contain arithmetic relations as well as predicates that can be grounded with temporal facts. In order to calculate the confidence of a fact, they apply an approach that is comparable to the work of Koch and Olteanu [2008] which also relies on consistency constraints and is executed during the query processing. The confidence of a fact depends on the confidence of its lineage (e.g., the facts that are used to deduce it) as well as the confidence of the grounded constraints.

They evaluate their approach using a small dataset (272 entities and 1,827 temporal facts) as well as YAGO2 [Hoffart et al., 2011, 2013] which contains a large amount of temporal annotated statements. However, YAGO2 is primarily used to investigate runtimes of different parts of their application for different query types. The results obtained from the experiments with the smaller dataset are more relevant as it also contains a high degree of facts ( $\approx 700$ ) that violate at least one constraint. Hence, they use this dataset in order to show that their approach is able to extract the correct statements from the dataset. The results indicate that their approach outperforms other systems relying on Markov Logic solvers or integer linear program solvers in terms of quality and runtime. Moreover, they apply their system with and without temporal constraints. Adding constraints causes a slightly higher runtime but also better results as the focus shifts from recall to precision.

**Discussion.** The presented approaches [Wang et al., 2010a; Dylla et al., 2011, 2013] have much in common with our work but there are also some differences. They rely on datasets containing weighted temporal annotated statements that contain a certain degree of errors. Hence, the processable data is comparable to the datasets which we considered in this work. In order to detect inconsistencies, they also define constraints using first-order logic formulas and the temporal relations of Allen’s interval algebra. They follow different approaches to resolve the inconsistencies but all rely on determining the consistent world (subset of facts) that has the highest probability. This is comparable to computing the MAP state in a Markov Logic Network. Moreover, Wang et al. [2010a] and Dylla et al. [2013] also consider inferring new facts by combining histograms or by applying first-order logic deduction rules. Thereby, they infer any interval based on the bounds of the intervals occurring in the original dataset. Our approach does not support this feature as we are not able to infer new intervals on demand. Hence, these approaches are better suited to reduce the incompleteness of a knowledge base. Another difference is that their approaches contain a query engine which is important as at least some parts of their algorithms resolve conflicts dynamically at query time. They argue that this is necessary in order to be able to handle large knowledge bases that contain changing constraints. Therefore, they consider only a subset of the data in order to compute the set of consistent facts that is relevant to the query. In order to achieve reasonable response times, Dylla et al. [2011] proposes an approximation algorithm. In contrast, our approach considers all statements in the knowledge bases and determines the world with the highest weight (MAP state). Moreover, they do not support weighted constraints and do also not exploit the benefits of relying on RDF(S) reasoning.

## Part 2.2: Reasoning using Markov Logic

**Temporal reasoning.** In the context of temporal reasoning, Markov logic is used in the field of event recognition. The focus of this area is inferring higher level events based on lower level events. Artikis et al. [2012] provide an overview on logic-based event recognition algorithms. In particular, they review approaches that rely on Markov Logic in order to process incomplete, inconsistent and erroneous data. Markov Logic handles these issues as it combines standard first-order logic and uncertainty. Skarlatidis et al. [2011] extend the Event Calculus [Kowalski and Sergot, 1989] with probabilistic reasoning using Markov Logic. The Event Calculus is a descriptive framework that allows expressing the effects of events. The ontology of the Event Calculus consists of time points, events and fluents. Events can occur at specific time points in order to initiate or to terminate fluents which are properties that change over time. Hence, domain-specific constraints

define when a fluent holds or does not hold. The constraints rely on the predicates of the calculus (e.g., *happens*, *holdsAt*, *initiates*, *terminates*). However, they face the problem that the state of a fluent does not change if no event occurs (*law of inertia*). Due to the open world semantics of first-order logic, they must explicitly define constraints that ensure when a fluent is not instantiated and when it is not terminated. Moreover, they need to restrict their approach to the discrete version of the Event Calculus [Mueller, 2008], which is equivalent for discrete time domains, because the original version leads to too large Markov Logic Networks. Their experiments underline that relying on soft constraints improves the event recognition [Skarlatidis et al., 2011]. So, their approach also uses Markov Logic in order to carry out temporal reasoning. They rely on the Event Calculus while we use Allen’s interval algebra. Moreover, it is notable that they faced the same issues as we did.

**MAP Inference in Log-Linear Description Logics.** Niepert et al. [2011] define log-linear description logics as a family of probabilistic logics that combines log-linear models [Koller and Friedman, 2009] and description logics [Baader and Nutt, 2003]. In particular, they focus on characteristics of the description logic  $\mathcal{EL}^{++}$  [Baader et al., 2005, 2008] that is also the basis of the Web Ontology Language OWL2+EL [Grau et al., 2012]<sup>3</sup> which is used in various domains of the Semantic Web. It is designed for ontologies that contain a very large number of classes and/or properties. OWL2 [Welty and McGuinness, 2004]<sup>4</sup> is linked to RDF as many OWL 2 documents are available in RDF or can be mapped to it<sup>5</sup>. Moreover, log-linear models are the basis of Markov Logic which makes their approach similar to our approach.

The semantics of log-linear description logics are defined by a log-linear probability distribution over coherent ontologies. Based on a deterministic CBox  $\mathcal{C}^D$ , that is assumed to be coherent, and an uncertain CBox  $\mathcal{C}^U$ , that contains real-valued weighted axioms, they compute the probability of another CBox  $\mathcal{C}'$  which is greater than zero if it is coherent and entails the deterministic CBox  $\mathcal{C}^D$ . A CBox is the constraint box that contains a finite set of general concept inclusion axioms (class axioms) and role inclusion axioms (property axioms). A typical inference task is to determine the most probable CBox  $\mathcal{C}'$  (MAP state). In order to do this, they convert  $\mathcal{EL}^{++}$  axioms to first-order logic as follows ( $r, s, r_i (1 \leq i \leq 3)$ )

<sup>3</sup>[http://www.w3.org/TR/owl2-profiles/#OWL\\_2\\_EL](http://www.w3.org/TR/owl2-profiles/#OWL_2_EL)

<sup>4</sup><http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/>

<sup>5</sup><http://www.w3.org/TR/2012/REC-owl2-mapping-to-rdf-20121211/>

denote properties/roles,  $C_1, C_2$  denote basic concept descriptions,  $D$  denotes basic concept descriptions or the bottom concept):

$$\begin{aligned}
C_1 \sqsubseteq D &\mapsto \text{sub}(C_1, D) \\
C_1 \sqcap C_2 \sqsubseteq D &\mapsto \text{int}(C_1, C_2, D) \\
C_1 \sqsubseteq \exists r.C_2 &\mapsto \text{rsub}(C_1, r, C_2) \\
\exists r.C_1 \sqsubseteq D &\mapsto \text{rsub}(C_1, r, D) \\
r \sqsubseteq s &\mapsto \text{psub}(r, s) \\
r_1 \circ r_2 \sqsubseteq r_3 &\mapsto \text{pcom}(r_1, r_2, r_3)
\end{aligned}$$

They implemented this theory in the reasoning system ELOG [Noessner and Niepert, 2011]. It transforms the problem to a Markov Logic Network and computes the MAP using rockIt. So, their system has much in common with the approach presented in this work as both focus on optimizing elements of the Semantic Web using Markov Logic. However, there are notable differences: ELOG is restricted apply to the terminological part of an ontology while our approach focuses on the assertional knowledge. Moreover, they only use the  $\mathcal{EL}^{++}$  logic while we rely on RDF(S) and temporal relations.



## Chapter 7

# Conclusion

The conclusion of this work is split in two parts. In Section 7.1, we summarize our approach and give the answers to the research questions. In Section 7.2, we are concerned with future work with respect to possible extensions and improvements.

### 7.1 Summary

In this section, we summarize this work and present the answers to the research questions stated in Section 1.3.

**RQ1** We proposed a Markov Logic based approach which is suitable to carry out temporal RDF(S) reasoning (see Chapter 3). In particular, we developed a flexible formalism to express temporal and non-temporal statements. Statements can be weighted, unweighted or observed which allows to state the confidence of a fact and to control whether certain facts are part of the output of the reasoning process. The supported types of statements are not only sufficient to express all kinds of facts that we target with our application but are also required to define entailment rules and constraints. It is possible to define any rule or constraint that can be written as a disjunction of literals having universally quantified variables. Our application contains the RDF(S) entailment rules which are required to infer statements based on the terminological knowledge associated with a dataset. We also incorporated Allen's interval algebra which allows expressing relations between temporal statements. Markov Logic is suitable for our approach as it supports weighted and hard (unweighted) constraints. Thus, it fulfills all requirements with respect to the essential types of statements and constraints. In order to detect and to resolve conflicts in datasets, we rely on Markov Logics' MAP inference. The MAP state is the most probable consistent set of statements with respect to the defined constraints

and rules. Hence, it allows removing erroneous statements from the dataset. We integrated the Markov Logic solver *rockIt* in our application but it is mentionable that our formalism does not rely on this specific system. Hence, it is possible to use our formalism with other Markov Logic solvers that support MAP inference.

**RQ2** We do not only incorporate the RDF(S) entailment rules but do also use the RDF data model in order to express the statements and constraints of a knowledge base. Therefore, we use reification to annotate statements with temporal information or confidence values as standard RDF statements are only triples. Moreover, we introduced a concept which is suitable to maintain domain-specific rules and constraints that relies on the RDF(S) vocabulary. Hence, all building blocks of our approach can be modeled with RDF (see Chapter 4).

**RQ3** The scalability of our approach primarily depends on the scalability of the used Markov Logic solver. One issue is that a high number of soft constraints in combination with weighted statements have a negative effect on the runtime. It is even possible that the Markov Logic solver does not terminate if the problem is too complex. Another issue is the representation of the relations of the temporal intervals that are annotated to the statements. We decided to introduce observed predicates for the relations of Allen's interval algebra and to initialize the Markov Logic Network with all ground predicates covering the relations of all intervals occurring in the dataset. This requires many system resources as the number of the required ground predicates is quadratic with respect to the existing intervals. For instance, 5,000 intervals do already lead to over 10,000,000 ground predicates. Nevertheless, this model does not affect the scalability if a dataset contains less than 2,500 intervals. For example, this is satisfactory if only years are annotated to the statements. However, the scalability of *rockIt* is sufficient to apply our approach to various use cases. In particular, we applied our application to datasets having more than 10,000 weighted statements and close to 800 weighted constraints as well as to datasets having more than 1,500,000 unweighted statements and over 20 hard constraints. The computation time was approximately 12 minutes in the first scenario and 3 hours in the second scenario.

**RQ4** In order to demonstrate the practicality of our approach, we presented an extensive evaluation of all features of our approach. We used a benchmark for Semantic Web knowledge base systems (Lehigh University Benchmark (LUBM)) in order to evaluate the RDF(S) reasoning capability of our approach (see Sec-

tion 5.1). The results indicate that our approach correctly infers new statements based on the RDF(S) entailment rules and some additional rules. However, state of the art reasoners that are dedicated to this task are faster and scale much better. For instance, in contrast to our approach such reasoners do not support a probabilistic setting. In the second part of the experiments (see Section 5.2), we investigated the capability of our application to detect erroneous statements in a dataset containing many weighted and temporal statements which we derived from facts that we got from DBPedia. The application improves the data quality by removing wrong statements but we recommend that a domain expert reviews those statements in order to increase the precision. Our application always acts as intended but it is possible that a dataset contains inconsistencies that cannot be correctly resolved given the evidence. We extended the dataset with generated (wrong) statements as we wanted to apply our approach to a dataset that contains a high degree of wrong statements. Our application also achieves reasonable results if the dataset contains many inconsistencies while the runtime increases only slightly. It is also observable that the precision increases if we decrease the weights of the generated statements as this enables that the conflicts can be resolved with a higher precision. Thus, the results underline that our application is suitable to remove erroneous statements from datasets with a high precision. We also applied our application to a dataset that is used to recognize activities (see Section 5.3) in order to demonstrate that our approach is compatible with any use case that provides temporal annotated facts and constraints. Moreover, this dataset is useful as it requires a large number of weighted constraints. We needed to split the input of our application as the complete dataset was not processable at once as it required too many system resources. However, generating smaller subproblems had a positive effect on the results as some activities could be detected with a higher precision. Overall, we achieved a worse precision but a better recall and F-measure than an application that is dedicated to this task. However, the respective application is more specialized and infers the activities faster. Nevertheless, the experiments underline that our approach can be applied to many domains.

## 7.2 Future Work

In the following, we outline ideas for possible improvements of our approach.

**Extension of the rule set.** Currently, we include only the RDF(S) entailment rules in order to give our approach a basic logical expressiveness. However, it might be worth to consider extending this rule set. Some features of the Web

Ontology Language (OWL), e.g., equality and inequality, might be beneficial in some use cases and can be expressed with our formalism.

**Inferring intervals.** Our approach is not able to infer new intervals as this would require solving too big Markov Logic Networks (see Section 4.3). Nevertheless, we think that this feature is helpful in order to close gaps in a knowledge base. In particular, we suggest to coalesce adjacent intervals annotated to the same non-temporal fact ( $[1, 3], [3, 5] \Rightarrow [1, 5]$ ), to compute the intersection of intervals ( $[1, 3], [2, 4] \Rightarrow [2, 3]$ ) and to infer new intervals based on the border points of other intervals ( $[1, 2], [8, 9] \Rightarrow [1, 9]$ ). Those features will extend the possibilities to infer new knowledge based on existing statements. However, our application computes all intervals relations before the execution of the Markov Logic solver which makes it not possible to infer new intervals on demand.

**Detection of subproblems.** The current representation of the interval relations in the Markov Logic Network is quadratic. Thus, the application requires much memory to maintain them. The experiments indicate that the limit is reached at less than 5,000 different intervals (i.e.,  $\approx 10$  million interval relations) if the process can use up to 16 GB memory. However, this number might be too small in order to carry out fine-grained temporal reasoning. Given the proposed approach, the best solution is to detect independent subproblems that contain a smaller number of intervals and statements.

**User interface.** The experiments indicate that it is advisable that a domain expert checks if the removed statements are actual wrong statements. Our application works as intended but yet it is possible that wrong statements have a higher weight than correct statements. Hence, we suggest developing a user interface that facilitates the revision of removed statements by domain experts.

**Additional experiments.** The primary application area of our approach is to improve the data quality of datasets derived from open information extraction systems. However, in the context of this work it was not feasible to carry out experiments with such a dataset. Hence, the next step is to create a dataset that contains weighted and temporal statements derived from the Web and to extend the experiments.

# Bibliography

- J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- E. Anagnostopoulos, S. Batsakis, and E. G. Petrakis. Chronos: A reasoning engine for qualitative temporal information in owl. *Procedia Computer Science*, 22: 70–77, 2013.
- A. Analyti and I. Pachoulakis. A survey on models and query languages for temporally annotated rdf. *International Journal of Advanced Computer Science & Applications*, 3(9):28–35, 2012.
- A. Artikis, A. Skarlatidis, F. Portet, and G. Paliouras. Logic-based event recognition. *The Knowledge Engineering Review*, 27:469–506, 12 2012.
- S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *The Semantic Web*, pages 722–735. Springer, 2007.
- F. Baader and W. Nutt. Basic description logics. In *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95. Cambridge University Press, 2003.
- F. Baader, S. Brandt, and C. Lutz. Pushing the  $\mathcal{EL}$  envelope. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05*, volume 5, pages 364–369, 2005.
- F. Baader, S. Brandt, and C. Lutz. Pushing the el envelope further. In *In Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*. Citeseer, 2008.
- S. Batsakis, K. Stravoskoufos, and E. G. Petrakis. Temporal reasoning for supporting temporal queries in owl 2.0. In *Knowledge-Based and Intelligent Information and Engineering Systems*, pages 558–567. Springer, 2011.

- T. Berners-Lee, J. Hendler, O. Lassila, et al. The semantic web. *Scientific American*, 284(5):28–37, 2001.
- C. Bizer, T. Heath, and T. Berners-Lee. Linked data-the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009a.
- C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia-a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009b.
- D. Brickley and R. Guha. RDF vocabulary description language 1.0: RDF schema. W3C recommendation, W3C, Feb. 2004. <http://www.w3.org/TR/2004/REC-rdf-schema-20040210/>.
- A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI 2010)*, 2010.
- G. Carothers and E. Prud’hommeaux. RDF 1.1 turtle. W3C recommendation, W3C, Feb. 2014. <http://www.w3.org/TR/2014/REC-turtle-20140225/>.
- G. Christodoulou, E. G. Petrakis, and S. Batsakis. Qualitative spatial reasoning using topological and directional information in owl. In *IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 1, pages 596–602, Nov 2012.
- P. Domingos and D. Lowd. Markov logic: An interface layer for artificial intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 3(1): 1–155, 2009.
- M. Dylla, M. Sozio, and M. Theobald. Resolving temporal conflicts in inconsistent rdf knowledge bases. In *14. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW)*, pages 474–493, 2011.
- M. Dylla, I. Miliaraki, and M. Theobald. A temporal-probabilistic database model for information extraction. *Proceedings of the VLDB Endowment*, 6(14):1810–1821, 2013.
- O. Etzioni, M. Banko, S. Soderland, and D. S. Weld. Open information extraction from the web. *Communications of the ACM*, 51(12):68–74, 2008.
- O. Etzioni, A. Fader, J. Christensen, S. Soderland, and M. Mausam. Open information extraction: The second generation. In *IJCAI*, volume 11, pages 3–10, 2011.

- M. R. Genesereth and N. J. Nilsson. *Logical foundations of artificial intelligence*, volume 9. Morgan Kaufmann Los Altos, CA, 1987.
- C. D. Godsil, G. Royle, and C. Godsil. *Algebraic graph theory*, volume 207. Springer New York, 2001.
- B. C. Grau, A. Fokoue, I. Horrocks, B. Motik, and Z. Wu. OWL 2 web ontology language profiles (second edition). W3C recommendation, W3C, Dec. 2012. <http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/>.
- Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158–182, 2005.
- C. Gutierrez, C. Hurtado, and A. Vaisman. Temporal rdf. In *The Semantic Web: Research and Applications*, pages 93–107. Springer, 2005.
- C. Gutierrez, C. A. Hurtado, and A. Vaisman. Introducing time into rdf. *IEEE Transactions on Knowledge and Data Engineering*, 19(2):207–218, 2007.
- S. Harris and A. Seaborne. SPARQL 1.1 query language. W3C recommendation, W3C, Mar. 2013. <http://www.w3.org/TR/2013/REC-sparql11-query-20130321/>.
- P. Hayes. RDF semantics. W3C recommendation, W3C, Feb. 2004. <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>.
- R. Helaoui, D. Riboni, and H. Stuckenschmidt. A probabilistic ontological framework for the recognition of multilevel human activities. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 345–354. ACM, 2013.
- J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. De Melo, and G. Weikum. Yago2: exploring and querying world knowledge in time, space, context, and many languages. In *Proceedings of the 20th international conference companion on World wide web*, pages 229–232. ACM, 2011.
- J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013.
- A. Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(01):14–21, 1951.

- C. Koch and D. Olteanu. Conditioning probabilistic databases. *Proceedings of the VLDB Endowment*, 1(1):313–325, 2008.
- D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. MIT Press, 2009.
- R. Kowalski and M. Sergot. A logic-based calculus of events. In *Foundations of Knowledge Base Management*, pages 23–55. Springer, 1989.
- J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 2014.
- G. Ligozat and J. Renz. What is a qualitative calculus? a general framework. In *PRICAI 2004: Trends in Artificial Intelligence*, volume 3157 of *Lecture Notes in Computer Science*, pages 53–64. Springer, 2004.
- X. Ling and D. S. Weld. Temporal information extraction. In *AAAI*, 2010.
- N. Lopes, G. Lukácsy, A. Polleres, U. Straccia, and A. Zimmermann. A general framework for representing, reasoning and querying with annotated semantic web data. Technical report, DERI, 2010.
- P. Lukowicz, G. Pirkl, D. Bannach, F. Wagner, A. Calatroni, K. Förster, T. Holczek, M. Rossi, D. Roggen, G. Tröster, et al. Recording a complex, multi modal activity data set for context recognition. In *Proceedings of ARCS '10 - 23th International Conference on Architecture of Computing Systems*, pages 161–166. VDE VERLAG GmbH, 2010.
- P. N. Mendes, H. Mühleisen, and C. Bizer. Sieve: linked data quality assessment and fusion. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 116–123. ACM, 2012.
- B. Motik. Representing and querying validity time in rdf and owl: A logic-based approach. *Web Semantics: Science, Services and Agents on the World Wide Web*, 12:3–21, 2012.
- B. Motik, P. Patel-Schneider, and B. Parsia. OWL 2 web ontology language structural specification and functional-style syntax (second edition). W3C recommendation, W3C, Dec. 2012. <http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/>.

- E. T. Mueller. Event calculus. *Handbook of knowledge representation*, 3:671–708, 2008.
- M. Niepert, J. Noessner, and H. Stuckenschmidt. Log-linear description logics. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pages 2153–2158. AAAI Press, 2011.
- J. Noessner and M. Niepert. Elog: a probabilistic reasoner for owl el. In *Web Reasoning and Rule Systems*, pages 281–286. Springer, 2011.
- J. Noessner, M. Niepert, and H. Stuckenschmidt. Rockit: Exploiting parallelism and symmetry for map inference in statistical relational models. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 2013.
- D. L. Olson and D. Delen. *Advanced data mining techniques*. Springer, 2008.
- J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.
- M. Richardson and P. Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.
- S. Riedel. Improving the accuracy and efficiency of map inference for markov logic. In *Proceedings of the 24th Annual Conference on Uncertainty in AI (UAI '08)*, pages 468–475, 2008.
- D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1): 273–302, 1996.
- A. Skarlatidis, G. Paliouras, G. A. Vouros, and A. Artikis. Probabilistic event calculus based on markov logic networks. In *Rule-Based Modeling and Computing on the Semantic Web*, pages 155–170. Springer, 2011.
- P. P. Talukdar, D. Wijaya, and T. Mitchell. Coupled temporal scoping of relational facts. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 73–82. ACM, 2012a.
- P. P. Talukdar, D. Wijaya, and T. Mitchell. Acquiring temporal constraints between relations. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 992–1001. ACM, 2012b.
- M. Theobald, M. Sozio, F. Suchanek, and N. Nakashole. Urdf: Efficient reasoning in uncertain rdf knowledge bases with soft and hard rules. Technical report, Max Planck Institute Informatics (MPI-INF), 2010.

- O. Udrea, V. Subrahmanian, and Z. Majkic. Probabilistic rdf. In *Information Reuse and Integration, 2006 IEEE International Conference on*, pages 172–177. IEEE, 2006.
- O. Udrea, D. R. Recupero, and V. Subrahmanian. Annotated rdf. *ACM Transactions on Computational Logic (TOCL)*, 11(2):10, 2010.
- M. B. Vilain and H. A. Kautz. Constraint propagation algorithms for temporal reasoning. In *AAAI*, volume 86, pages 377–382, 1986.
- Y. Wang, M. Yahya, and M. Theobald. Time-aware reasoning in uncertain knowledge bases. In *Proceedings of the Fourth International VLDB workshop on Management of Uncertain Data (MUD 2010)*, pages 51–65, 2010a.
- Y. Wang, M. Zhu, L. Qu, M. Spaniol, and G. Weikum. Timely yago: harvesting, querying, and visualizing temporal knowledge from wikipedia. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 697–700. ACM, 2010b.
- C. Welty and D. McGuinness. OWL web ontology language guide. W3C recommendation, W3C, Feb. 2004. <http://www.w3.org/TR/2004/REC-owl-guide-20040210/>.
- A. Yates, M. Cafarella, M. Banko, O. Etzioni, M. Broadhead, and S. Soderland. Texrunner: open information extraction on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 25–26. Association for Computational Linguistics, 2007.
- A. Zimmermann, N. Lopes, A. Polleres, and U. Straccia. A general framework for representing, reasoning and querying with annotated semantic web data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 11:72–95, 2012.

# Appendix A

## Markov Logic Model

### A.1 Basic Model

```
// observed predicates
*tripleW(r,r,r,float_)
*quadW(r,r,r,Interval,float_)
*tripleO(r,r,r)
*quadO(r,r,r,Interval)
*literal(r)

// hidden predicates
triple(r,r,r)
quad(r,r,r,Interval)

// soft constraints with individual weights
weight: !tripleW(s,p,o,weight) v triple(s,p,o)
weight: !quadW(s,p,o,i,weight) v quad(s,p,o,i)

// hard constraints
!quadO(s,p,o,i) v quad(s,p,o,i).
!quad(s,p,o,i) v triple(s,p,o).
!tripleO(s,p,o) v triple(s,p,o).
```

**Listing A.1:** Basic RDF Model

```
*tBefore(Interval,Interval)
*tMeets(Interval,Interval)
*tOverlaps(Interval,Interval)
*tStarts(Interval,Interval)
*tDuring(Interval,Interval)
*tFinishes(Interval,Interval)
*tEqual(Interval,Interval)
```

**Listing A.2:** Predicates of Allen's Interval Algebra

## A.2 RDF(S) Entailment Rules

```

// rdf1 + rdf2
!triple(s,p,o) v triple(p,"rdf:type","rdf:Property").
// rdf2 - not added - rdfs1 is sufficient

// rdfs1
!triple(s,p,o) v !literal(o) v triple(o,"rdf:type","rdfs:Literal").

// rdfs 2+3 domain + range
!triple(s,p,o) v !triple(p,"rdfs:domain",x) v triple(s,"rdf:type",x).
!triple(s,p,o) v !triple(p,"rdfs:range",x) v triple(o,"rdf:type",x).
!quad(s,p,o,i) v !triple(p,"rdfs:domain",x) v quad(s,"rdf:type",x,i).
!quad(s,p,o,i) v !triple(p,"rdfs:range",x) v quad(o,"rdf:type",x,i).

// rdfs 4 resource
!triple(s,p,o) v triple(s,"rdf:type","rdfs:Resource").
!triple(s,p,o) v triple(o,"rdf:type","rdfs:Resource").

// rdfs 5-7 subProperty
!triple(a,"rdfs:subPropertyOf",b) v !triple(b,"rdfs:subPropertyOf",c)
  v triple(a,"rdfs:subPropertyOf",c).

!triple(p,"rdf:type","rdf:Property") v triple(p,"rdfs:subPropertyOf",p).

!triple(a,"rdfs:subPropertyOf",b) v !triple(s,a,o) v triple(s,b,o).
!triple(a,"rdfs:subPropertyOf",b) v !quad(s,a,o,i) v quad(s,b,o,i).

// rdfs 8-11 subClass
!triple(c,"rdf:type","rdfs:Class")
  v triple(c,"rdfs:subClassOf","rdfs:Resource").

!triple(a,"rdfs:subClassOf",b) v !triple(x,"rdf:type",a)
  v triple(x,"rdf:type",b).

!triple(a,"rdfs:subClassOf",b) v !quad(x,"rdf:type",a,i)
  v quad(x,"rdf:type",b,i).

!triple(c,"rdf:type","rdfs:Class") v triple(c,"rdfs:subClassOf",c).

!triple(a,"rdfs:subClassOf",b) v !triple(b,"rdfs:subClassOf",c)
  v triple(a,"rdfs:subClassOf",c).

// rdfs 12+13
!triple(p,"rdf:type","rdfs:ContainerMembershipProperty")
  v triple(p,"rdfs:subPropertyOf","rdfs:member").

!triple(u,"rdf:type","rdfs:Datatype")
  v triple(u,"rdfs:subClassOf","rdfs:Literal").

```

**Listing A.3:** RDF(S) Entailment Rules

## A.3 RDF and RDFS Vocabulary

```
// rdf
triple(rdf:nil,rdf:type,rdf:List)
triple("rdf:nil","rdf:type","rdf:List")
triple("rdf:rest","rdfs:range","rdf:List")
triple("rdf:rest","rdfs:domain","rdf:List")
triple("rdf:rest","rdf:type","rdf:Property")
triple("rdf:Alt","rdfs:subClassOf","rdfs:Container")
triple("rdf:Alt","rdf:type","rdfs:Class")
triple("rdf:object","rdfs:range","rdfs:Resource")
triple("rdf:object","rdfs:domain","rdf:Statement")
triple("rdf:object","rdf:type","rdf:Property")
triple("rdf:first","rdfs:range","rdfs:Resource")
triple("rdf:first","rdfs:domain","rdf:List")
triple("rdf:first","rdf:type","rdf:Property")
triple("rdf:Bag","rdfs:subClassOf","rdfs:Container")
triple("rdf:Bag","rdf:type","rdfs:Class")
triple("rdf:type","rdfs:domain","rdfs:Resource")
triple("rdf:type","rdfs:range","rdfs:Class")
triple("rdf:type","rdf:type","rdf:Property")
triple("rdf:Property","rdfs:subClassOf","rdfs:Resource")
triple("rdf:Property","rdf:type","rdfs:Class")
triple("rdf:value","rdfs:range","rdfs:Resource")
triple("rdf:value","rdfs:domain","rdfs:Resource")
triple("rdf:value","rdf:type","rdf:Property")
triple("rdf:PlainLiteral","rdfs:subClassOf","rdfs:Literal")
triple("rdf:PlainLiteral","rdf:type","rdfs:Datatype")
triple("rdf:List","rdfs:subClassOf","rdfs:Resource")
triple("rdf:List","rdf:type","rdfs:Class")
triple("rdf:XMLLiteral","rdfs:subClassOf","rdfs:Literal")
triple("rdf:XMLLiteral","rdf:type","rdfs:Datatype")
triple("rdf:subject","rdfs:range","rdfs:Resource")
triple("rdf:subject","rdfs:domain","rdf:Statement")
triple("rdf:subject","rdf:type","rdf:Property")
triple("rdf:Seq","rdfs:subClassOf","rdfs:Container")
triple("rdf:Seq","rdf:type","rdfs:Class")
triple("rdf:Statement","rdfs:subClassOf","rdfs:Resource")
triple("rdf:Statement","rdf:type","rdfs:Class")
triple("rdf:predicate","rdfs:range","rdfs:Resource")
triple("rdf:predicate","rdfs:domain","rdf:Statement")
triple("rdf:predicate","rdf:type","rdf:Property")
```

**Listing A.4:** RDF Vocabulary

```

// rdfs
triple("rdfs:Container","rdfs:subClassOf","rdfs:Resource")
triple("rdfs:Container","rdf:type","rdfs:Class")
triple("rdfs:seeAlso","rdfs:domain","rdfs:Resource")
triple("rdfs:seeAlso","rdfs:range","rdfs:Resource")
triple("rdfs:seeAlso","rdf:type","rdf:Property")
triple("rdfs:Resource","rdf:type","rdfs:Class")
triple("rdfs:subPropertyOf","rdfs:domain","rdf:Property")
triple("rdfs:subPropertyOf","rdfs:range","rdf:Property")
triple("rdfs:subPropertyOf","rdf:type","rdf:Property")
triple("rdfs:ContainerMembershipProperty","rdfs:subClassOf","rdf:Property")
triple("rdfs:ContainerMembershipProperty","rdf:type","rdfs:Class")
triple("rdfs:Class","rdfs:subClassOf","rdfs:Resource")
triple("rdfs:Class","rdf:type","rdfs:Class")
triple("rdfs:subClassOf","rdfs:domain","rdfs:Class")
triple("rdfs:subClassOf","rdfs:range","rdfs:Class")
triple("rdfs:subClassOf","rdf:type","rdf:Property")
triple("rdfs:comment","rdfs:range","rdfs:Literal")
triple("rdfs:comment","rdfs:domain","rdfs:Resource")
triple("rdfs:comment","rdf:type","rdf:Property")
triple("rdfs:Literal","rdfs:subClassOf","rdfs:Resource")
triple("rdfs:Literal","rdf:type","rdfs:Class")
triple("rdfs:member","rdfs:range","rdfs:Resource")
triple("rdfs:member","rdfs:domain","rdfs:Resource")
triple("rdfs:member","rdf:type","rdf:Property")
triple("rdfs:Datatype","rdfs:subClassOf","rdfs:Class")
triple("rdfs:Datatype","rdf:type","rdfs:Class")
triple("rdfs:range","rdfs:domain","rdf:Property")
triple("rdfs:range","rdfs:range","rdfs:Class")
triple("rdfs:range","rdf:type","rdf:Property")
triple("rdfs:isDefinedBy","rdfs:domain","rdfs:Resource")
triple("rdfs:isDefinedBy","rdfs:range","rdfs:Resource")
triple("rdfs:isDefinedBy","rdfs:subPropertyOf","rdfs:seeAlso")
triple("rdfs:isDefinedBy","rdf:type","rdf:Property")
triple("rdfs:domain","rdfs:domain","rdf:Property")
triple("rdfs:domain","rdfs:range","rdfs:Class")
triple("rdfs:domain","rdf:type","rdf:Property")
triple("rdfs:label","rdfs:range","rdfs:Literal")
triple("rdfs:label","rdfs:domain","rdfs:Resource")
triple("rdfs:label","rdf:type","rdf:Property")

```

Listing A.5: RDFS Vocabulary