MODELING AND QUERYING PROBABILISTIC RDFS DATA WITH CORRELATED TRIPLES USING BAYESIAN NETWORKS

CHI CHEONG SZETO

Ph.D

The Hong Kong Polytechnic University

2014

The Hong Kong Polytechnic University
Department of Computing

Modeling and Querying Probabilistic RDFS Data with Correlated Triples Using Bayesian Networks

Chi Cheong Szeto

A thesis submitted in partial fulfilment

of the requirements for the degree of

Doctor of Philosophy

August 2013

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_______________________________________(Signed)

Chi Cheong Szeto

Abstract

Resource Description Framework (RDF) is a World Wide Web Consortium (W3C) data model for the Semantic Web. RDF data are RDF triples, and an RDF triple is a triple (subject, property, object). RDF Schema (RDFS) extends RDF by providing a vocabulary to describe application-specific classes and properties, class and property hierarchies, and which classes and properties are used together. RDFS reasoning leverages the vocabulary to derive additional RDF triples from the data.

In recent years, probabilistic models for RDF have been proposed to better represent the real-life information, which is full of uncertainties. Existing models either have limited capabilities to model correlated data or ignore the semantics of the data. We argue that being able to model correlated RDF data is necessary. First, RDF data using the RDFS vocabulary are correlated. Second, correlated data occur in practice. Hence, we introduce a probabilistic model called probabilistic RDFS (pRDFS), which encodes statistical relationships among correlated RDF triples and satisfies the RDFS semantics. Representing and performing probabilistic inference on correlated data are expensive. We use Bayesian networks to represent the correlated data and probabilistic logic sampling to perform approximate inference.

Since there may exist some truth value assignments that violate the RDFS semantics, we devise a consistency checking algorithm for pRDFS. The algorithm checks that the probabilities of all inconsistent truth value assignments for the correlated RDF triples are zeros. It is executed once on static data. For data that are frequently updated, we propose an incremental approach that provides fast rechecking each time the data are updated.

SPARQL is a W3C query language for RDF. The pattern of a SPARQL query is a conjunction of triple patterns, and a triple pattern is an RDF triple any member of which can be replaced with a variable. A solution to the query is the bindings of the query variables such that the query pattern matches the data or the data derived through the RDFS reasoning. We extend the query by including truth values in the triple patterns to match the uncertain data. Apart from the bindings of the query variables, an answer to the extended query includes the probability of the bindings, which is equal to the probability of the matched data. pRDFS fully specifies the probability distribution of declared data, but not derived data. A single probability value may not be able to specify the probability of the matched data containing derived data, and we show how to compute the probability bounds of the matched data in this case.

Finally, we present an experimental evaluation of the running time performance of our proposed algorithms with respect to the data size, the percentage of uncertain data, the size of correlated data (by varying the number of nodes in a Bayesian network), and the complexity of the probability distributions (by varying the degree of nodes in a network). The algorithms were tested on the Berlin SPARQL Benchmark, the Lehigh University Benchmark, and random uncertain data.

List of Publications

SZETO, C.C., HUNG, E., AND DENG, Y. Modeling and querying probabilistic RDFS data sets with correlated triples. In *APWeb* (2011), X. Du, W. Fan, J. Wang, Z. Peng, and M. A. Sharaf, Eds., vol. 6612 of *Lecture Notes in Computer Science*, Springer, pp. 333–344. (Best Student Paper Award)

SZETO, C.C., HUNG, E., AND DENG, Y. SPARQL query answering with RDFS reasoning on correlated probabilistic data. In *WAIM* (2011), H. Wang, S. Li, S. Oyama, X. Hu, and T. Qian, Eds., vol. 6897 of *Lecture Notes in Computer Science*, Springer, pp. 56–67.

Acknowledgements

I would like to take this opportunity to express my sincere gratitude to my chief supervisor Dr. Edward Hung for giving me the opportunity to do research in semantic web and data mining, the useful discussions, encouragement, guidance, and support. I am indebted to my co-supervisor Dr. Korris Chung for his valuable advice and checking my thesis. Finally, I would like to thank my parents for their understanding and support. Without the assistance of these people, I would not have been able to finish this thesis.

Contents

List of Figures

List of Tables

List of Algorithms

Notations

Symbol	**Description**
$\mathbb{C}$	set of classes
$(D, \mathcal{P}, \theta)$	pRDFS theory
D	set of schema and instance triples
D_{ext}	instance of a pRDFS theory
DJ(x, X)	disjunction of all justifications for an RDF triple x in data X
$\mathcal{F}$	propositional formula
G	graph pattern
G_{ext}	extended graph pattern
H	set of schema triples
$\mathbb{I}$	set of individuals
JUST(x, X)	set of all justifications for an RDF triple x in data X
JUST$_{\text{min}}(x, X)$	set of all minimal justifications for an RDF triple x in data X
$\mathbb{L}$	set of literals
M^{op}	sequence of solutions after applying SPARQL operator op
N_{ideg}	in-degree of nodes
N_j	number of minimal justifications
N_{node}	number of nodes in a Bayesian networks
N_{odeg}	out-degree of nodes
N_{sol}	number of solutions
p	percentage of uncertain data
$\mathcal{P}$	set of probability distributions
P_X	probability distribution over data X
$\mathbb{P}$	set of properties
$Pr(x)$	probability of x
$Pr_{\text{lower}}(x)$	lower probability of x
$Pr_{\text{upper}}(x)$	upper probability of x

q	percentage of derived data
Q	SPARQL query
Q_{ext}	extended SPARQL query
R	set of instance triples
rdfs-closure(X)	RDFS closure of data X
T, **F**	true, false
$\mathbb{U}$	set of URI references
V	set of query variables
W	RDFS interpretation
$\mathbb{W}$	set of all RDFS interpretations
$\lvert X \rvert$	number of elements in the set X
Γ_μ	set of alternative matched data of the solution μ
μ_i^{op}	ith solution in the sequence of solutions M^{op}
τ	truth value assignment
θ	function mapping an RDF triple to its probability distribution

Chapter 1

Introduction

1.1 Motivation of Research

Resource Description Framework (RDF) [39] is a basic data model for the Semantic Web [7], the contents of which are machine-readable. An RDF dataset is a set of RDF triples, and an RDF triple is a triple (subject, property, object), where the subject is a resource, the object is a resource or a literal, and the property (also called predicate) specifies the relationship between the subject and the object. Uniform Resource Identifier (URI) references [6] are used to name the resources and the properties of the RDF triples.

RDF Schema (RDFS) [11] is an extension of RDF. It provides a vocabulary to describe application-specific classes and properties, class and property hierarchies, and which classes and properties are used together. This defines the meaning of data and enables machines to perform automated reasoning on the data. Table 1.1 shows five RDF triples and their meanings, where abbreviations are used for URI references and

Symbol	RDF triples	Meaning
d_1	(University, rdf:type, rdfs:Class)	University is a class.
d_2	(degreeFrom, rdf:type, rdf:Property)	degreeFrom is a property.
d_3	(degreeFrom, rdfs:range, University)	The range of degreeFrom is a University.
d_4	(John, degreeFrom, PolyU)	John has a degree from PolyU.
d_5	(PolyU, rdf:type, University)	PolyU is a University.

Table 1.1: RDF triples and their meanings.

the words with prefixes rdf: and rdfs: are in the RDF and RDFS vocabularies respectively. In this example, RDF triples d_3 and d_4 entail RDF triple d_5 by RDFS reasoning.

In recent years, RDF has become popular. The inter-linked RDF datasets published following the Linked Data practices [5] form the Web of Data. As of August 2013, there are 62 billion RDF triples from 870 datasets [2]. The contents of these datasets are diverse. They include data extracted from Wikipedia [1], bioinformatics data [4], book data [9], music, television and radio programmes [40], etc.

Many applications produce uncertain data. For example, an automatic data integration system [23] may return a ranked list of alternative RDF triples for the same piece of information because of the conflicting data obtained from different data sources. A score is computed for each alternative based on the accuracy, reliability, and interdependence of the data sources, and it reflects the degree of belief that the alternative is the correct one. In automatic information extraction [31], a system may return a list of alternative RDF triples and their probabilities of being true for each piece of extracted information because of the imperfect extraction process. In sensor networks [22], data from the sensors are approximate as they are acquired at discrete time and space. Moreover, they are correlated. Temperature data from sensors in close proximity to each other are likely to be correlated. Temperature and voltage data from a single sensor could also be correlated.

Probabilistic models [27, 28, 59, 37, 44] have been proposed for the uncertain RDF data. However, they either have limited capabilities to model correlated data [59, 37] or ignore the semantics of the data [27, 28, 44]. We argue that being able to model correlated RDF data is necessary. First, RDF data using the RDFS vocabulary are correlated. Assume that the RDF triples in Table 1.1 are uncertain. If both d_3 and d_4 are true, then d_5 is also true according to the RDFS semantics. Hence, the probability that d_3 = true, d_4 = true, and d_5 = false should be zero. RDF triples d_3, d_4, and d_5 are correlated. Second, correlated data occur in practice. The data generated in the applications mentioned earlier are correlated.

The work described in this thesis is about modeling uncertain correlated RDF data. It also examines the issues arising from the RDFS semantics like the consistency of the uncertain data and the probability calculation of derived data.

1.2 Contributions

The first contribution of this thesis is a probabilistic model for RDF which models the uncertainties of correlated RDF data. Representing and performing probabilistic inference on correlated data are expensive. We use Bayesian networks to represent the correlated data and probabilistic logic sampling to perform approximate inference.

The second contribution is a consistency checking algorithm which checks if the probability distribution of the correlated RDF data satisfies the RDFS semantics. Moreover, for data that are frequently updated, we provide an incremental consistency checking algorithm which performs fast rechecking each time the data are updated.

The third contribution is a query evaluation algorithm. We extend the query to the uncertain data, and an answer to the extended query includes the solutions to the query pattern and the probabilities of the solutions. For solutions that contain derived data, their probabilities cannot be specified using a single probability value because the probability distribution of the derived data is not fully specified. In this case, the probability bounds of the solutions are computed.

The forth contribution is the notion of minimal justifications for RDF triples, which is used to reduce the computations of both the consistency checking and query evaluation algorithms.

The final contribution is an experimental evaluation of the running time performance of consistency checking and query evaluation with respect to the data size, the percentage of uncertain data, the size of correlated data, and the complexity of the probability distributions. Moreover, we present three models for predicting the average-case running time of probability calculations in consistency checking, query evaluation with and without RDFS reasoning.

1.3 Organization

The remainder of this thesis is organized as follows.

- Chapter 2 presents the background knowledge of this research, which includes RDF, RDFS, SPARQL, a review of probabilistic models for RDF, and Bayesian networks.

- Chapter 3 describes the syntax and semantics of our proposed probabilistic model for RDF. It also introduces the notion of justifications and describes how to compute them. Justifications are used in consistency checking and query evaluation.

- Chapter 4 presents a consistency checking algorithm to check if the uncertain data in our model satisfy the RDFS semantics or not. For data that are frequently updated, it presents an incremental consistency checking algorithm which performs fast rechecking each time the data are updated.
- Chapter 5 extends the SPARQL queries for our proposed model. It defines the answers to the extended queries and describes how to compute them.
- Chapter 6 presents an experimental evaluation of the running time performance of the consistency checking algorithms in Chapter 4 and the query evaluation algorithm in Chapter 5.
- Chapter 7 concludes this thesis and suggests some future work.

Chapter 2

Background

This chapter presents the background knowledge of this research, which includes Resource Description Framework (RDF), RDF Schema (RDFS), and SPARQL, which is the World Wide Web Consortium (W3C) query language for RDF. It also reviews the probabilistic models for RDF. Finally, it describes Bayesian networks and probabilistic logic sampling, which are chosen in our proposed probabilistic model for RDF to represent probability distributions and perform probabilistic inference respectively.

2.1 RDF and RDFS

Resource Description Framework (RDF) [39] is a World Wide Web Consortium (W3C) Recommendation. It is a data model that uses Uniform Resource Identifier (URI) references [6] to identify things and uses RDF triples to make statements. In the latest development of RDF [17], URIs are replaced by Internationalized Resource Identifiers (IRIs) [24], which are a generalization of URIs.

Vocabulary	Uses
rdf:type, rdfs:Class	Define classes
rdf:type, rdf:Property	Define properties
rdfs:subClassOf	Define class hierarchies
rdfs:subPropertyOf	Define property hierarchies
rdfs:domain, rdfs:range	Define which classes and properties are used together

Table 2.1: RDFS vocabulary and its uses.

Let $\mathbb{U}$ be a set of URI references and contain three mutually disjoint sets: classes $\mathbb{C}$, properties $\mathbb{P}$, and individuals $\mathbb{I}$. Let $\mathbb{L}$ be a set of literals. An RDF triple is defined as follows.

Definition 1 (RDF Triple)**.** *An RDF triple is a triple (*s*,* p*,* o*) in* $\mathbb{U} \times \mathbb{P} \times (\mathbb{U} \cup \mathbb{L})$*, where* $\mathbb{U}$*,* $\mathbb{P}$*, and* $\mathbb{L}$ *are sets of URI references, properties, and literals respectively. The elements of the RDF triple* s*,* p*, and* o *are called the subject, property, and object respectively.*

RDF triples are classified according to their subjects. An RDF triple is a **schema triple** if its subject is in $(\mathbb{C} \cup \mathbb{P})$. It is an **instance triple** if its subject is in $\mathbb{I}$. Schema triples describe classes and properties, and they are more permanent and definitional. Instance triples describe individuals, and they are more dynamic.

RDF data are a set of RDF triples. They can be viewed as a directed graph. The subject and object of each RDF triple represent vertices of the graph. The (subject, object) pair of each RDF triple represents a directed edge of the graph, and the label of the directed edge is the property of the RDF triple.

RDF Schema (RDFS) [11] is an extension of RDF. It provides a vocabulary to describe application-specific classes and properties, class and property hierarchies, and which classes and properties are used together. Table 2.1 shows the RDFS vocabulary

1	If ($?p$, rdfs: domain, $?x$), ($?u$, $?p$, $?w$) $\in W$, then ($?u$, rdf: type, $?x$) $\in W$.
2	If ($?p$, rdfs: range, $?x$), ($?u$, $?p$, $?v$) $\in W$, then ($?v$, rdf: type, $?x$) $\in W$.
3	If ($?p$, rdfs: subPropertyOf, $?q$), ($?q$, rdfs: subPropertyOf, $?r$) $\in W$, then ($?p$, rdfs: subPropertyOf, $?r$) $\in W$.
4	If ($?p$, rdf: type, rdf: Property) $\in W$, then ($?p$, rdfs: subPropertyOf, $?p$) $\in W$.
5	If ($?p$, rdfs: subPropertyOf, $?q$), then ($?p$, rdf: type, rdf: Property), ($?q$, rdf: type, rdf: Property) $\in W$ and $\forall$ $?u$ $\forall$ $?w$ (($?u$, $?p$, $?w$) $\in W \Rightarrow$ ($?u$, $?q$, $?w$) $\in W$).
6	If ($?x$, rdfs: subClassOf, $?y$), then ($?x$, rdf: type, rdfs: Class), ($?y$, rdf: type, rdfs: Class) $\in W$ and $\forall$ $?u$ (($?u$, rdf: type, $?x$) $\in W \Rightarrow$ ($?u$, rdf: type, y) $\in W$).
7	If ($?x$, rdf: type, rdfs: Class) $\in W$, then ($?x$, rdfs: subClassOf, $?x$) $\in W$.
8	If ($?x$, rdfs: subClassOf, $?y$), ($?y$, rdfs: subClassOf, $?z$) $\in W$, then ($?x$, rdfs: subClassOf, $?z$) $\in W$.
9	RDFS axiomatic triples $\subseteq W$.

Table 2.2: RDFS semantic conditions.

(rdfs:Resource, rdf:type, rdfs:Class)	(rdf:type, rdf:type, rdf:Property)
(rdfs:Literal, rdf:type, rdfs:Class)	(rdf:type, rdfs:domain, rdfs:Resource)
(rdfs:Class, rdf:type, rdfs:Class)	(rdf:type, rdfs:range, rdfs:Class)
(rdf:Property, rdf:type, rdfs:Class)	(rdfs:subPropertyOf, rdf:type, rdf:Property)
(rdfs:domain, rdf:type, rdf:Property)	(rdfs:subPropertyOf, rdfs:domain, rdf:Property)
(rdfs:domain, rdfs:domain, rdf:Property)	(rdfs:subPropertyOf, rdfs:range, rdf:Property)
(rdfs:domain, rdfs:range, rdfs:Class)	(rdfs:subClassOf, rdf:type, rdf:Property)
(rdfs:range, rdf:type, rdf:Property)	(rdfs:subClassOf, rdfs:domain, rdfs:Class)
(rdfs:range, rdfs:domain, rdf:Property)	(rdfs:subClassOf, rdfs:range, rdfs:Class)
(rdfs:range, rdfs:range, rdfs:Class)	

Table 2.3: RDFS axiomatic triples.

and its uses. RDFS reasoning leverages the vocabulary to derive additional RDF triples from the RDF triples explicitly declared in the data.

2.1.1 Semantics

An RDFS interpretation denoted by W is a set of RDF triples that satisfies the semantic conditions [39] in Table 2.2. For simplicity, we ignore the semantics of blank nodes, containers, and datatypes. Among these conditions, $?x$, $?y$, and $?z$ are variables for classes. $?p$, $?q$, and $?r$ are variables for properties. $?u$ and $?v$ are variables for individuals. Finally, $?w$ is a variable for both individuals and literals. The last semantic

Names	Rules
rdfs2	$\frac{(?p, \text{rdfs: domain}, ?x)\ (?u, ?p, ?w)}{(?u, \text{rdf: type}, ?x)}$
rdfs3	$\frac{(?p, \text{rdfs: range}, ?x)\ (?u, ?p, ?v)}{(?v, \text{rdf: type}, ?x)}$
rdfs5	$\frac{(?p, \text{rdfs: subPropertyOf}, ?q)\ (?q, \text{rdfs: subPropertyOf}, ?r)}{(?p, \text{rdfs: subPropertyOf}, ?r)}$
rdfs6	$\frac{(?p, \text{rdf: type}, \text{rdf: Property})}{(?p, \text{rdfs: subPropertyOf}, ?p)}$
rdfs7	$\frac{(?p, \text{rdfs: subPropertyOf}, ?q)\ (?u, ?p, ?w)}{(?u, ?q, ?w)}$
rdfs9	$\frac{(?x, \text{rdfs: subClassOf}, ?y)\ (?u, \text{rdf: type}, ?x)}{(?u, \text{rdf: type}, ?y)}$
rdfs10	$\frac{(?x, \text{rdf: type}, \text{rdfs: Class})}{(?x, \text{rdfs: subClassOf}, ?x)}$
rdfs11	$\frac{(?x, \text{rdfs: subClassOf}, ?y)\ (?y, \text{rdfs: subClassOf}, ?z)}{(?x, \text{rdfs: subClassOf}, ?z)}$

Table 2.4: RDFS inference rules.

condition says that RDFS axiomatic triples are in all RDFS interpretations. RDFS axiomatic triples define the classes, domains, and ranges of the RDFS vocabulary, and they are shown in Table 2.3.

An RDFS interpretation W satisfies an RDF triple d iff $d \in W$. W satisfies a set of RDF triples D iff W satisfies every $d \in D$. A set of RDF triples D is consistent iff it has a satisfying interpretation. A set of RDF triples D rdfs-entails ($\models_{rdfs}$) an RDF triple d iff every satisfying interpretation of D is a satisfying interpretation of d. A set of RDF triples D rdfs-entails another set of RDF triples D' iff every satisfying interpretation of D is a satisfying interpretation of D'.

2.1.2 Inference Rules

Table 2.4 shows the RDFS inference rules [39] that correspond to the semantic conditions shown in Table 2.2. Among these rules, the things that the variables $?x$, $?y$, $?z$, $?p$, $?q$, $?r$, $?u$, $?v$, and $?w$ stand for are the same as those in the semantic conditions.

RDF triples above the line are called the **premises**, and RDF triples below the line are called the **conclusions**.

These rules can be divided into two groups. Rules rdfs5, rdfs6, rdfs10 and rdfs11 derive schema triples, and their premises contain schema triples only. Rules rdfs2, rdfs3, rdfs7 and rdfs9 derive instance triples.

2.1.3 RDFS Closure

Definition 2 (RDFS closure)**.** *The RDFS closure of data D denoted by rdfs-closure(D) is the union of D, the set of RDFS axiomatic triples, and the set of RDF triples generated by applying the rules in Table 2.4 recursively.*

2.1.4 Other Terms

An RDF triple d is called an **entailed triple** in data D if it is entailed by $(D \setminus \{d\})$. It is called an **entailing triple** in data D if a subset of D containing d entails an RDF triple in $(D \setminus \{d\})$, but the subset cannot entail the RDF triple in the absence of d. An RDF triple can both be an entailed and entailing triple in D.

An RDF triple is called a **declared triple** if it is in data D. It is called a **derived triple** if it is entailed by D and is not in D.

2.2 SPARQL

2.2.1 SPARQL Query

SPARQL [50] is the W3C query language for RDF. We focus on one common form of SPARQL queries, which is defined as follows. Other query languages for RDF include RDF Data Query Language (RDQL) [51] and Sesame RDF Query Language (SeRQL) [12].

Definition 3 (Triple Pattern)**.** *A triple pattern is an RDF triple any member of which can be a variable. It is a member of* $(\mathbb{U} \cup V) \times (\mathbb{P} \cup V) \times (\mathbb{U} \cup \mathbb{L} \cup V)$*, where* $\mathbb{U}$*,* $\mathbb{P}$*,* $\mathbb{L}$*, and* V *are sets of URI references, properties, literals, and variables respectively.*

Definition 4 (Graph Pattern)**.** *A graph pattern denoted by* G *is a set of triple patterns.*

Definition 5 (SPARQL Query)**.** *We denote by* $Q(V_{sel}, G, \textit{isDistinct})$ *a SPARQL query with a syntax "SELECT (DISTINCT)* V_{sel} *WHERE* G*", where* G *is a graph pattern,* V_{sel} *is a subset of query variables in* G*, and isDistinct is a Boolean variable with a domain* $\{\boldsymbol{T}, \boldsymbol{F}\}$*. The DISTINCT keyword is optional, and the variable isDistinct assigned* ***T*** *and* ***F*** *indicates the presence and absence of the DISTINCT keyword respectively.*

2.2.2 Answer to a SPARQL Query

An answer to a SPARQL query $Q(V_{\mathrm{sel}}, G, \mathrm{isDistinct})$ on RDF data D is a sequence of solutions, and a solution is the restriction of a solution to G on D to the set of query variables V_{sel}. The answer does not contain duplicate solutions if the variable isDistinct is true. A solution to G on D denoted by μ is a partial function from V to $(\mathbb{U} \cup \mathbb{L})$ such that $D \models_{rdfs} \mu(G)$ or equivalently rdfs-closure$(D) \supseteq \mu(G)$ (with RDFS reasoning) or $D \supseteq \mu(G)$ (without RDFS reasoning), where V, $\mathbb{U}$, and $\mathbb{L}$ are sets of query variables, URI references and literals respectively. $\mu(G)$ is the set of RDF triples obtained by replacing every variable v in G with $\mu(v)$.

2.2.3 Evaluation of a SPARQL Query

An evaluation of a SPARQL query $Q(V_{\text{sel}}, G, \text{isDistinct})$ on RDF data D consists of four steps.

1. Find the set of all solutions to the graph pattern G on D. In SPARQL algebra, the set of solutions is BGP(G).

2. Convert the set of solutions BGP(G) into a sequence of solutions M^{ToList}. In SPARQL algebra, M^{ToList} = ToList(BGP(G)).

3. Convert the sequence of solutions M^{ToList} into a sequence of solutions M^{Project}, where the ith solution in M^{Project} is the restriction of the ith solution in M^{ToList} to the set of query variables V_{sel}. In SPARQL algebra, M^{Project} = Project(M^{ToList}, V_{sel}).

4. If the variable isDistinct is true, create a sequence of solutions M^{Distinct} by removing duplicate solutions from the sequence of solutions M^{Project}. In SPARQL algebra, M^{Distinct} = Distinct(M^{Project}). Otherwise, leave the sequence M^{Project} unmodified, that is, M^{Distinct} = M^{Project}.

The answer to the SPARQL query is the sequence of solutions M^{Distinct}.

2.3 Review of Probabilistic Models for RDF

This section reviews the probabilistic models for RDF. In relational databases, RDF data can be modeled as a relation with attributes “subject”, “property”, and “object”. However, RDFS reasoning is not supported. Probabilistic models for relational databases encode the uncertainty about the attribute of a tuple [3] or about the existence of a tuple [19, 26, 20, 52]. The model in [3] associates possible attribute values of a tuple with probability values. The models in [19, 26, 20] associate each tuple with a probability value. In [19, 26], tuples are assumed to be independent of each other. In [20], tuples are partitioned into blocks. Tuples from the same block are mutually exclusive, and tuples from different blocks are independent of each other. The model in [52] associates each tuple with a random variable, and the joint probability distribution of all the variables is described compactly using a factored representation [48].

An RDF vocabulary is proposed in [27, 28] to represent the uncertainty about the truth values of correlated n-ary relations [46]. A n-ary relation is represented by a group of RDF triples, and an RDF triple is a special case of a n-ary relation. The probability distributions of correlated n-ary relations are represented by Bayesian networks [48], and the nodes of the networks are n-ary relations. RDFS reasoning is not supported. A query for this representation is asking the probability of some n-ary relation of interest given the truth values of some n-ary relations.

Probabilistic RDF (pRDF) [59] models the uncertainty about the object of an RDF triple. The probability of a conjunction of RDF triples is computed using triangular norms in fuzzy logic [25]. pRDF supports the inference of the property rdfs:subPropertyOf from the RDFS vocabulary (rules rdfs5 and rdfs7 in Table 2.4) and application-specific transitive properties. If a property p_t is transitive, then the RDF triple (x, p_t, z) can be derived from the RDF triples (x, p_t, y) and (y, p_t, z). The query proposed for pRDF has a query pattern of the form (s, p, o, λ), where (s, p, o) is an RDF triple, λ is a probability value, and at most one member of the query pattern can be a variable. A solution to the query is a binding of the query variable such that the query pattern matches the declared data or the data derived by the inference of subproperties or transitive properties and the probability of the matched data is above λ.

Probabilistic RDF database (pRDFDB) [37] models the uncertain RDF triples that are statistically independent of each other. It assigns a probability value to each uncertain RDF triple. pRDFDB supports the inference of transitive properties. However, the independence assumption among the RDF triples may violate the RDFS semantics if the RDFS vocabulary is used. Take the uncertain RDF triples d_1 = (PolyU, rdf: type, University), d_2 = (degreeFrom, rdfs: range, University), and d_3 = (John, degreeFrom, PolyU) as an example. In pRDFDB, probability values p_1, p_2, and p_3 are assigned to these uncertain RDF triples respectively. The probability that d_1 = false, d_2 = true, and d_3 = true is $(1 - p_1)p_2p_3$, which could be non-zero. However, it should be zero since $\{d_2$, $d_3\}$ rdfs-entails d_1 by rule rdfs3. See Table 2.4 for the rule. The SPARQL query with a basic graph pattern is proposed for pRDFDB. The answer to the query includes the solutions to the basic graph pattern and the probabilities of the solutions.

Probabilistic RDF graph database (pRDFGDB) [44] views RDF data as a directed graph. It assumes the directed edges of the graph are deterministic and models the uncertainty of the label of a vertex of the graph given the labels of its parent vertices,

Probabilistic model	Uncertainty modeled	Inference supported	Query
Probabilistic relational database [3, 19, 26, 20, 52]	Element of an RDF triple [3]; Truth values of independent RDF triples [19, 26]; Truth values of mutually exclusive RDF triples [20]; Truth values of correlated RDF triples [52]	None	SQL
Fukushige's vocabulary [27, 28]	Truth values of correlated n-ary relations	None	Probability of the n-ary relation of interest given the truth values of some n-ary relations
Probabilistic RDF (pRDF) [59]	Object of an RDF triple	rdfs:subPropertyOf property, transitive property	Triple pattern
Probabilistic RDF database (pRDFDB) [37]	Truth values of independent RDF triples	Transitive property	Graph pattern
Probabilistic RDF graph database (pRDFGDB) [44]	Label of a vertex given the labels of its parent vertices	None	Graph pattern
Probabilistic RDFS (pRDFS) (Chapter 3)	Truth values of correlated RDF triples	RDFS (Table 2.4)	Graph pattern

Table 2.5: Comparison among the probabilistic models for RDF.

which are vertices pointing to the vertex via directed edges. pRDFGDB does not support any inference. The SPARQL query with a basic graph pattern is proposed for pRDFGDB. The answer to the query is the solutions to the graph pattern, the probabilities of which are greater than a user-specified value.

Table 2.5 compares the probabilistic models for RDF in terms of the uncertainty modeled, inference supported, and query. Our proposed model probabilistic RDFS (pRDFS) is described in Chapter 3.

2.4 Bayesian Network

Bayesian networks [48] are widely used in modeling uncertain knowledge. The application domains include medical diagnosis [33, 32, 34], information retrieval [21], ecosystem services [41], forensic DNA profiling evidence [8], dependability, risk analysis, and maintenance [60].

2.4.1 Representation

A Bayesian network is a directed, acyclic graph. Each node in the network represents a random variable X_i and is annotated with a conditional probability distribution P(X_i | Parents(X_i)), where Parents(X_i) are the parents of X_i. A node X_j is a parent of X_i if there is a directed edge from X_j to X_i. The **in-degree** of a node X_i is the number of edges whose heads are X_i, and the **out-degree** of a node X_i is the number of edges whose tails are X_i.

A Bayesian network over random variables $\{X_1$, X_2, ..., $X_n\}$ represents the joint probability distribution over $\{X_1$, X_2, ..., $X_n\}$ denoted by P(X_1, X_2, ..., X_n), which is shown in (2.1).

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i \mid \text{Parents}(X_i)). \tag{2.1}$$

The number of values specified for a Bayesian network is usually much smaller than the number of values specified for a full joint probability distribution because of the conditional independence relationships encoded by the structure of the network. A node is independent of its non-descendants given the values of its parents.

2.4.2 Inference

The task of inference generally refers to computing the probability of some assignment of values to the random variables of a Bayesian network given the evidence (an assignment of values to some other random variables of the network). Exact inference like the variable elimination [62] and the conditioning [47] algorithms in unconstrained Bayesian networks is NP-hard [15]. Approximate inference is also NP-hard [18]. We consider one family of approximate inference methods called direct sampling to trade accuracy for speed.

Probabilistic logic sampling [35] creates N_{sampl} samples of random values for the nodes of a network. For each sample, the random values are generated in the topological ordering, which is an ordering of nodes such that every node comes after its parents in the ordering. The random value for each node is generated from the conditional probability distribution of the node given the random values of its parents that have already been generated. The probability of the assignment of values is approximated by the number of samples matching both the assignment and the evidence divided by the number of samples matching the evidence.

Samples that do not match the evidence are rejected in probability logic sampling. Likelihood weighting [29, 53] makes an improvement to probabilistic logic sampling. It only generates samples that match the evidence, and each sample is weighted by a likelihood term. In this thesis, only the prior probability is computed, so the computations of both methods are the same.

The minimum value of N_{sampl} depends on the required accuracy of the estimate of the probability. We use the Hoeffding bound [36] to determine the minimum number of samples, and the equation for this is shown in (2.2). The probability that the estimate deviates from the true value more than ϵ is equal to or less than δ. The minimum number of samples grows logarithmically in $1/\delta$ and quadratically in $1/\epsilon$.

$$N_{\text{sampl}} \geq \frac{ln(2/\delta)}{2\epsilon^2}. \tag{2.2}$$

2.4.3 Learning

Bayesian networks are constructed by domain experts, who identify the variables, determine the relationships of direct influence among the variables, and specify the conditional probability table of each variable. They can also be learned from data, and this task is NP-hard [14]. One family of methods is called score-and-search methods [16, 42, 43]. It specifies a scoring function that measures how well a network fits the data and a greedy search procedure that finds the network with the locally best score. Another family of methods is called constraint-based methods [54, 55]. It performs statistical and causal inference tests on the data to discover the conditional independence and dependence relationships among the data and recovers the network structure from these relationships. In this thesis, we assume that the probability distributions of uncertain data encoded by Bayesian networks are given.

Chapter 3

Probabilistic Model pRDFS

This chapter introduces a probabilistic model for RDF called probabilistic RDFS (pRDFS), which allows statistical relationships among correlated RDF triples to be encoded [57]. The syntax and semantics of the model are described in Sections 3.1 and 3.2 respectively. Section 3.3 describes algorithms of finding the data that entail an RDF triple of interest. The algorithms are used in consistency checking and query evaluation described in later chapters.

3.1 Syntax

A probabilistic RDFS (pRDFS) theory has three parts. Firstly, it has a set of RDF triples. Secondly, it has a set of independent probability distributions over subsets of the RDF triples. The RDF triples are treated as random variables with a domain {true, false} abbreviated as {**T**, **F**}. Thirdly, it has a function that maps each RDF triple to its corresponding probability distribution. A pRDFS theory is formally defined as follows.

Definition 6 (pRDFS theory). *A pRDFS theory is a triple (D, $\mathcal{P}$, θ), where*

1. *D is a set of RDF triples.* $\{D_1, D_2, ..., D_n\}$ *is a partition of D, any two different elements of which are statistically independent.*

2. $\mathcal{P}$ = $\{P_{D_1},\ P_{D_2},\ ...,\ P_{D_n}\}$ *is a set of probability distributions, where* P_{D_i} : $\{\boldsymbol{T}, \boldsymbol{F}\}^{|D_i|} \rightarrow [0, 1]$ *is a probability distribution over* D_i.

3. $\theta : D \rightarrow \mathcal{P},\ d \mapsto P_{D_i}$ *is a function mapping an RDF triple to its probability distribution, where* $d \in D_i$.

Let τ: $D \rightarrow \{\mathbf{T}, \mathbf{F}\}$ be a truth value assignment for data D. It maps each RDF triple in D to a truth value. We define the probability of a truth value assignment for D denoted by $Pr(\tau)$ as $P_D(d_1 = \tau(d_1),\ d_2 = \tau(d_2),\ ...,\ d_{|D|} = \tau(d_{|D|}))$, where P_D is the probability distribution over D and D = $\{d_1,\ d_2,\ ...,\ d_{|D|}\}$. Because of the independence assumption of the elements of the partition $\{D_1, D_2, ..., D_n\}$, $Pr(\tau)$ = $Pr(\tau|_{D_1}) \times Pr(\tau|_{D_2}) \times \cdots \times Pr(\tau|_{D_n})$, where $\tau|_{D_i}$: $D_i \rightarrow \{\mathbf{T}, \mathbf{F}\}$ is the restriction of τ to D_i.

For a small value of $|D_i|$, the probability distribution over D_i can be represented by a table listing the probabilities of all possible truth value assignments for D_i. For a large value of $|D_i|$, the probability distribution can be modeled using the Bayesian network (Section 2.4).

Not all RDF triples are allowed to be uncertain or false. RDFS axiomatic triples shown in Table 2.3 are always true. Schema triples ($?p$ rdf: type, rdf: Property) and

($?x$ rdf: type, rdfs: Class) are assumed to be certain, where $?p$ and $?x$ are variables for properties and classes respectively. Otherwise, some RDF triples may be invalid. By rules rdfs6 and rdfs10 shown in Table 2.4, it follows that schema triples ($?p$ rdfs: subPropertyOf $?p$) and ($?x$ rdfs: subClassOf $?x$) are also certain.

Example 1 (pRDFS theory)**.** *A pRDFS theory is shown in Table 3.1. It has 14 RDF triples (Table 3.1a). The first ten RDF triples are certain and true. Their probabilities of being true are one (Table 3.1c). Suppose that professors Tom and May both specialize in the semantic web. Their chances being the teacher of the course semantic web are equal, and this is specified by the probability distribution $P_{\{d_{11},d_{12}\}}$ (Table 3.1d). Students John and Mary have the same interest. Their chances to take the course semantic web are described by the probability distribution $P_{\{d_{13},d_{14}\}}$ (Table 3.1e).*

3.2 Semantics

We define the pRDFS interpretation, pRDFS satisfaction, and consistency of a pRDFS theory as follows.

Definition 7 (pRDFS interpretation)**.** *A pRDFS interpretation is a mapping $I : \mathbb{W} \to [0, 1]$ such that $\sum_{W \in \mathbb{W}} I(W) = 1$, where $\mathbb{W}$ is the set of all possible worlds (RDFS interpretations).*

A world (RDFS interpretation) W satisfies a truth value assignment τ for data D if and only if the RDF triples assigned true by τ are in W, and the RDF triples assigned false by τ are not in W. Mathematically, W satisfies τ iff $\forall d \in D((\tau(d) = \mathbf{T} \Rightarrow d \in W)) \wedge (\tau(d) = \mathbf{F} \Rightarrow d \notin W))$.

Symbol	RDF triple
d_1	(Student, rdf: type, rdfs: Class)
d_2	(Professor, rdf: type, rdfs: Class)
d_3	(Course, rdf: type, rdfs: Class)
d_4	(takesCourse, rdf: type, rdf: Property)
d_5	(teacherOf, rdf: type, rdf: Property)
d_6	(semanticWeb, rdf: type, Course)
d_7	(Tom, rdf: type, Professor)
d_8	(May, rdf: type, Professor)
d_9	(John, rdf: type, Student)
d_{10}	(Mary, rdf: type, Student)
d_{11}	(Tom, teacherOf, semanticWeb)
d_{12}	(May, teacherOf, semanticWeb)
d_{13}	(John, takesCourse, semanticWeb)
d_{14}	(Mary, takesCourse, semanticWeb)

(a) Set of RDF triples, D_1.

Probability distribution
$P_{\{d_1\}}$
$P_{\{d_2\}}$
$P_{\{d_3\}}$
$P_{\{d_4\}}$
$P_{\{d_5\}}$
$P_{\{d_6\}}$
$P_{\{d_7\}}$
$P_{\{d_8\}}$
$P_{\{d_9\}}$
$P_{\{d_{10}\}}$
$P_{\{d_{11},d_{12}\}}$
$P_{\{d_{13},d_{14}\}}$

(b) Set of probability distributions, $\mathcal{P}_1$.

d_i	Probability
T	1
F	0

(c) Probability distribution over d_i, $P_{\{d_i\}}$, where i = 1, 2, ..., 10.

d_{11}	d_{12}	Probability
T	**T**	0
T	**F**	0.5
F	**T**	0.5
F	**F**	0

(d) Probability distribution over $\{d_{11}, d_{12}\}$, $P_{\{d_{11},d_{12}\}}$.

d_{13}	d_{14}	Probability
T	**T**	0.4
T	**F**	0.1
F	**T**	0.1
F	**F**	0.4

(e) Probability distribution over $\{d_{13}, d_{14}\}$, $P_{\{d_{13},d_{14}\}}$.

x	$\theta(x)$
d_1	$P_{\{d_1\}}$
d_2	$P_{\{d_2\}}$
d_3	$P_{\{d_3\}}$
d_4	$P_{\{d_4\}}$
d_5	$P_{\{d_5\}}$
d_6	$P_{\{d_6\}}$
d_7	$P_{\{d_7\}}$
d_8	$P_{\{d_8\}}$
d_9	$P_{\{d_9\}}$
d_{10}	$P_{\{d_{10}\}}$
d_{11}, d_{12}	$P_{\{d_{11},d_{12}\}}$
d_{13}, d_{14}	$P_{\{d_{13},d_{14}\}}$

(f) Function θ_1 mapping an RDF triple to it probability distribution.

Table 3.1: Example of a consistent pRDFS theory (D_1, $\mathcal{P}_1$, θ_1).

Definition 8 (pRDFS satisfaction)**.** *A pRDFS interpretation I satisfies a pRDFS theory $(D, \mathcal{P}, \theta)$ iff the probability of any truth value assignment τ: $D \rightarrow$ \{**T**, **F**\} computed*

W	$I(W)$
$A \cup \{d_{11}, d_{13}, d_{14}\}$	0.2
$A \cup \{d_{11}, d_{13}\}$	0.05
$A \cup \{d_{11}, d_{14}\}$	0.05
$A \cup \{d_{11}\}$	0.2
$A \cup \{d_{12}, d_{13}, d_{14}\}$	0.2
$A \cup \{d_{12}, d_{13}\}$	0.05
$A \cup \{d_{12}, d_{14}\}$	0.05
$A \cup \{d_{12}\}$	0.2
All other worlds	0

Table 3.2: A satisfying interpretation I of the pRDFS theory in Table 3.1, where A = RDFS axiomatic triples $\cup$ $\{d_1, d_2, ..., d_{10}\}$.

from the theory satisfies the following equation.

$$Pr(\tau) = \begin{cases} \sum_{W \in \mathbb{W}_\tau} I(W) & \textit{if } |\mathbb{W}_\tau| > 0 \\ 0 & \textit{if } |\mathbb{W}_\tau| = 0 \end{cases} \tag{3.1}$$

where $\mathbb{W}_\tau = \{W \in \mathbb{W} \mid W \textit{ satisfies } \tau\}$ is the set of worlds that satisfies τ.

Definition 9 (Consistent pRDFS theory)**.** *A pRDFS theory is consistent iff it has a satisfying interpretation.*

Example 2 (Consistent pRDFS theory)**.** *The pRDFS theory in Table 3.1 has a satisfying interpretation shown in Table 3.2, so it is consistent. The interpretation satisfies (3.1) for all truth value assignments. For example, the truth value assignment τ for the situation that Tom is the teacher of the course semantic web and both John and Mary do not take the course semantic web is $\{$(d_{11}, **T**), (d_{12}, **F**), (d_{13}, **F**), (d_{14}, **F**)$\}$. The probability of τ is computed as $P_{\{d_{11},d_{12}\}}(d_{11}$ = **T**, d_{12} = **F**$)$ × $P_{\{d_{13},d_{14}\}}(d_{13}$ = **F**, d_{14} = **F**$)$ = 0.5 × 0.4 = 0.2. The world that corresponds to this situation is (RDFS axiomatic triples $\cup$ $\{d_1, d_2, ..., d_{10}\}$ $\cup$ $\{d_{11}\}$) shown in the forth row of Table 3.2. It is mapped by the interpretation to 0.2, which is the same as the one computed from the theory.*

Symbol	RDF triple
d_1	(Professor, rdf: type, rdfs: Class)
d_2	(Department, rdf: type, rdfs: Class)
d_3	(worksFor, rdf: type, rdf: Property)
d_4	(headOf, rdf: type, rdf: Property)
d_5	(headOf, rdfs: subPropertyOf, worksFor)
d_6	(Tom, rdf: type, Professor)
d_7	(departmentOfComputing, rdf: type, Department)
d_8	(Tom, headOf, departmentOfComputing)
d_9	(Tom, worksFor, departmentOfComputing)

(a) Set of RDF triples, D_2.

Probability distribution
$P_{\{d_1\}}$
$P_{\{d_2\}}$
$P_{\{d_3\}}$
$P_{\{d_4\}}$
$P_{\{d_5\}}$
$P_{\{d_6\}}$
$P_{\{d_7\}}$
$P_{\{d_8\}}$
$P_{\{d_9\}}$

(b) Set of probability distributions, $\mathcal{P}_2$.

d_i	Probability
T	1
F	0

(c) Probability distribution over d_i, $P_{\{d_i\}}$, where i = 1, 2, ..., 7.

d_8	Probability
T	0.7
F	0.3

(d) Probability distribution over d_8, $P_{\{d_8\}}$.

d_9	Probability
T	0.8
F	0.2

(e) Probability distribution over d_9, $P_{\{d_9\}}$.

x	$\theta(x)$
d_1	$P_{\{d_1\}}$
d_2	$P_{\{d_2\}}$
d_3	$P_{\{d_3\}}$
d_4	$P_{\{d_4\}}$
d_5	$P_{\{d_5\}}$
d_6	$P_{\{d_6\}}$
d_7	$P_{\{d_7\}}$
d_8	$P_{\{d_8\}}$
d_9	$P_{\{d_9\}}$

(f) Function θ_2 mapping an RDF triple to it probability distribution.

Table 3.3: Example of an inconsistent pRDFS theory (D_2, $\mathcal{P}_2$, θ_2).

Example 3 (Inconsistent theory)**.** *This example shows that the pRDFS theory* $(D_2$, $\mathcal{P}_2$, $\theta_2)$ *shown in Table 3.3 is not consistent. The pRDFS theory has nine RDF triples (Table 3.3a). The first seven RDF triples are true, and their probability distributions are described in Table 3.3c. The last two RDF triples* d_8 *and* d_9 *are uncertain, and their probability distributions* $P_{\{d_8\}}$ *and* $P_{\{d_9\}}$ *are specified in Tables 3.3d and 3.3e respectively.*

Consider the truth value assignment τ = $\{$*(*d_1*,* ***T****), (*d_2*,* ***T****), (*d_3*,* ***T****), (*d_4*,* ***T****), (*d_5*,* ***T****), (*d_6*,* ***T****), (*d_7*,* ***T****), (*d_8*,* ***T****), (*d_9*,* ***F****)*$\}$*. There is no world that contains* d_5 *and* d_8*, but does not contain* d_9*. It is because* $\{d_5, d_8\} \models_{rdfs} d_9$ *by the RDFS inference rule rdfs7*

(Table 2.4). By the definition of pRDFS satisfaction (Definition 8), the probability of τ should be zero for any satisfying pRDFS interpretation. However, the probability of τ computed from the theory is $P_{\{d_8\}}(d_8 = \boldsymbol{T}) \times P_{\{d_9\}}(d_9 = \boldsymbol{F})$ = 0.7 × 0.2 = 0.14. The theory does not have any satisfying interpretations, so it is inconsistent.

A theory $(D, \mathcal{P}, \theta)$ prdfs-entails ($\models_{prdfs}$) another theory $(D', \mathcal{P}', \theta')$ iff every satisfying interpretation of $(D, \mathcal{P}, \theta)$ is a satisfying interpretation of $(D', \mathcal{P}', \theta')$.

3.3 Finding the Data that Entail an RDF Triple of Interest

This section examines two special subsets of data that entail an RDF triple of interest. One is called a justification. It is used to check the consistency of a pRDFS theory (see Chapter 4). It is also used to compute the probability of a solution to a query, where the solution contains derived RDF triples (see Chapter 5). The other special subset is called a minimal justification, which is used to reduce computations (see Propositions 4 and 5).

A justification for an RDF triple t is a subset of data that entails t, and the subset cannot entail t in the absence of any triple in the subset. This term is formally defined in OWL-DL [38]. It is defined similarly in RDFS as follows.

Definition 10 (Justification)**.** *A justification for an RDF triple t in data D denoted by J is a subset of D such that $J \models_{rdfs} t$ and $\nexists J' \subset J (J' \models_{rdfs} t)$.*

(p, rdf: type, rdf: Property)
(q, rdf: type, rdf: Property)
(r, rdf: type, rdf: Property)
(s, rdf: type, rdf: Property)
(p, rdfs: subPropertyOf, q)
(q, rdfs: subPropertyOf, r)
(q, rdfs: subPropertyOf, s)
(r, rdfs: subPropertyOf, s)

(a) Schema data of D_3, H_3.

(u, p, v)

(b) Instance data of D_3, R_3.

(p, rdf: type, rdf: Property)
(q, rdf: type, rdf: Property)
(r, rdf: type, rdf: Property)
(s, rdf: type, rdf: Property)
(p, rdfs: subPropertyOf, q)
(q, rdfs: subPropertyOf, r)
(q, rdfs: subPropertyOf, s)
(r, rdfs: subPropertyOf, s)
(p, rdfs: subPropertyOf, r)
(p, rdfs: subPropertyOf, s)
(p, rdfs: subPropertyOf, p)
(q, rdfs: subPropertyOf, q)
(r, rdfs: subPropertyOf, r)
(s, rdfs: subPropertyOf, s)

(c) RDFS closure of schema data H_3, rdfs-closure(H_3).

Table 3.4: Data D_3 used to illustrate the concepts of justifications and how to find the justifications, where p, q, r, and s are properties. u and v are individuals.

Example 4 (Justification)**.** *Consider data D_3 shown in Table 3.4, where p, q, r, and s are properties. u and v are individuals. J_1 = $\{$(p, rdfs: subPropertyOf, q), (q, rdfs: subPropertyOf, r), (r, rdfs: subPropertyOf, s)$\}$ and J_2 = $\{$(p, rdfs: subPropertyOf, q), (q, rdfs: subPropertyOf, s)$\}$ are both justifications for the RDF triple (p, rdfs: subPropertyOf, s) in D_3 because each of them is a subset of D_3 and entails (p, rdfs: subPropertyOf, s). $\{$(p, rdfs: subPropertyOf, q), (q, rdfs: subPropertyOf, s), (r, rdfs: subPropertyOf, s)$\}$ is a subset of D_3 and also entails (p, rdfs: subPropertyOf, s), but it is not a justification for (p, rdfs: subPropertyOf, s) in D_3 because its subset $\{$(p, rdfs: subPropertyOf, q), (q, rdfs: subPropertyOf, s)$\}$ is sufficient to entail (p, rdfs: subPropertyOf, s).*

A minimal justification for an RDF triple is a justification for the triple, and it cannot entail other justifications for the triple. It is defined as follows.

Definition 11 (Minimal Justification)**.** *A justification J for an RDF triple t in data D*

Names	Rules
rdfs7	$\frac{(?q, \text{rdfs: subPropertyOf}, ?p)\ (?u, ?q, ?w)}{(?u, ?p, ?w)}$
rdfs9	$\frac{(?x, \text{rdfs: subClassOf}, ?y)\ (?u, \text{rdf: type}, ?x)}{(?u, \text{rdf: type}, ?y)}$
rdfs2+rdfs9	$\frac{(?x, \text{rdfs: subClassOf}, ?y)\ (?p, \text{rdfs: domain}, ?x)\ (?u, ?p, ?w)}{(?u, \text{rdf: type}, ?y)}$
rdfs7+rdfs2+rdfs9	$\frac{\begin{array}{c}(?x, \text{rdfs: subClassOf}, ?y)\ (?p, \text{rdfs: domain}, ?x) \\ (?q, \text{rdfs: subPropertyOf}, ?p)\ (?u, ?q, ?w)\end{array}}{(?u, \text{rdf: type}, ?y)}$
rdfs3+rdfs9	$\frac{(?x, \text{rdfs: subClassOf}, ?y)\ (?p, \text{rdfs: range}, ?x)\ (?v, ?p, ?u)}{(?u, \text{rdf: type}, ?y)}$
rdfs7+rdfs3+rdfs9	$\frac{\begin{array}{c}(?x, \text{rdfs: subClassOf}, ?y)\ (?p, \text{rdfs: range}, ?x) \\ (?q, \text{rdfs: subPropertyOf}, ?p)\ (?v, ?q, ?u)\end{array}}{(?u, \text{rdf: type}, ?y)}$
rdfs2	$\frac{(?p, \text{rdfs: domain}, ?x)\ (?u, ?p, ?w)}{(?u, \text{rdf: type}, ?x)}$
rdfs7+rdfs2	$\frac{(?p, \text{rdfs: domain}, ?x)\ (?q, \text{rdfs: subPropertyOf}, ?p)\ (?u, ?q, ?w)}{(?u, \text{rdf: type}, ?x)}$
rdfs3	$\frac{(?p, \text{rdfs: range}, ?x)\ (?v, ?p, ?u)}{(?u, \text{rdf: type}, ?x)}$
rdfs7+rdfs3	$\frac{(?p, \text{rdfs: range}, ?x)\ (?q, \text{rdfs: subPropertyOf}, ?p)\ (?v, ?q, ?u)}{(?u, \text{rdf: type}, ?x)}$

Table 3.5: All possible chains of inference rules that derive instance triples given that the schema data H are equal to the RDFS closure of H.

is minimal if $\nexists$ *$J' \in$ JUST(t, D) ($J \models_{rdfs} J'$ and $J \neq J'$), where JUST(t, D) is the set of all justifications for* t *in* D.

A justification J for an RDF triple t in data D is called a minimal justification in the sense that for every $J' \in$ JUST(t, D) such that $J' \models_{rdfs} J$, $|J| \leq |J'|$. A justification is non-minimal if it is not minimal. The set of all justifications for an RDF triple is the union of the sets of all minimal and non-minimal justifications.

Example 5 (Minimal Justification). *Both J_1 and J_2 in Example 4 are justifications for the RDF triple (p, rdfs: subPropertyOf, s) in data D_3. J_2 is minimal, but J_1 is not because J_1 entails J_2.*

3.3.1 Computing Minimal Justifications for Instance Triples under H = rdfs-closure(H)

This section describes a method of computing all minimal justifications for an instance triple in data D under the condition that the schema data H of D are equal to the RDFS closure of H. The RDFS inference rules that derive instance triples are rdfs2, rdfs3, rdfs7, and rdfs9. See Table 2.4 for these rules. We use backward chaining to find all possible chains of these rules that derive instance triples, and they are shown in Table 3.5. For example, the rule rdfs2+rdfs9 in row 3 of Table 3.5 is formed by chaining rules rdfs2 and rdfs9. The set of RDF triples in the premises of each rule is a justification for the instance triple in the conclusion of the rule. These possible chains of rules do not include those formed by repeated applications of rules rdfs7 and rdfs9 because any justification found by repeated applications of these rules is not minimal. This is proved as follows.

Proposition 1. *Given that the schema data H in data D are equal to the RDFS closure of H, any justification for an instance triple t in D found by repeated applications of rule rdfs7 or rule rdfs9, or both is not minimal.*

Proof. Let J be a justification for t in D found by repeated applications of rule rdfs7 or rule rdfs9, or both. J is of the form $\{(p_m$, rdfs: subPropertyOf, $p_{m-1})$, $(p_{m-1}$, rdfs: subPropertyOf, $p_{m-2})$, $\ldots$, $(p_1$, rdfs: subPropertyOf, $p_0)\} \cup \{(x_n$, rdfs: subClassOf, $x_{n-1})$, $(x_{n-1}$, rdfs: subClassOf, $x_{n-2})$, $\ldots$, $(x_1$, rdfs: subClassOf, $x_0)\} \cup J_{\text{remain}}$, where p_0, p_1, $\ldots$, p_m are properties, x_0, x_1, $\ldots$, x_n are classes, m and n are the numbers of times rules rdfs7 and rdfs9 are applied respectively, and J_{remain} is a set that contains the

Algorithm 1 FIND-MIN-JUST$^{*}{}_{\text{inst}}(t, D)$ finds all minimal justifications for an instance triple t in data D, where the schema data H of D equal the RDFS closure of H.

```
Input: $t$, instance triple; $D$, data.
Output: $\mathbb{J}$, set of all minimal justifications for $t$ in $D$.
$H \leftarrow$ schema data of $D$; $R \leftarrow$ instance data of $D$.
$s \leftarrow$ subject of $t$; $p \leftarrow$ property of $t$; $o \leftarrow$ object of $t$.
$\mathbb{J} \leftarrow \emptyset$.
if $t \in R$ then
    $\mathbb{J} \leftarrow \mathbb{J} \cup \{\{t\}\}$.
else if $p \neq$ rdf:type then
    $\mathbb{J} \leftarrow \mathbb{J} \cup \{\{(?q$, rdfs: subPropertyOf, $p)$, $(s, ?q, o)\} \mid (?q$, rdfs: subPropertyOf, $p) \in H$, $(s, ?q, o) \in R\}$. // rule rdfs7
else
    $X_{\text{rdfs9}} \leftarrow \{?x \mid (?x$, rdfs: subClassOf, $o) \in H$, $(s$, rdf: type, $?x) \in R\}$.
    $\mathbb{J} \leftarrow \mathbb{J} \cup \{\{(?x$, rdfs: subClassOf, $o)$, $(s$, rdf: type, $?x)\} \mid ?x \in X_{\text{rdfs9}}\}$. // rule rdfs9
    $X_{\text{rdfs2+9}} \leftarrow \{(?x, ?p, ?w) \mid (?x$, rdfs: subClassOf, $o) \in H$, $(?p$, rdfs: domain, $?x) \in H$, $(s, ?p, ?w) \in R$, $?x \notin X_{\text{rdfs9}}\}$.
    $\mathbb{J} \leftarrow \mathbb{J} \cup \{\{(?x$, rdfs: subClassOf, $o)$, $(?p$, rdfs: domain, $?x)$, $(s, ?p, ?w)\} \mid (?x, ?p, ?w) \in X_{\text{rdfs2+9}}\}$. // rule rdfs2+rdfs9
    $\mathbb{J} \leftarrow \mathbb{J} \cup \{\{(?x$, rdfs: subClassOf, $o)$, $(?p$, rdfs: domain, $?x)$, $(?q$, rdfs: subPropertyOf, $?p)$, $(s, ?q, ?w)\} \mid (?x$, rdfs: subClassOf, $o) \in H$, $(?p$, rdfs: domain, $?x) \in H$, $(?q$, rdfs: subPropertyOf, $?p) \in H$, $(s, ?q, ?w) \in R$, $(?x, ?p, ?w) \notin X_{\text{rdfs2+9}}$, $?x \notin X_{\text{rdfs9}}\}$. // rule rdfs7+rdfs2+rdfs9
    $X_{\text{rdfs3+9}} \leftarrow \{(?x, ?p, ?v) \mid (?x$, rdfs: subClassOf, $o) \in H$, $(?p$, rdfs: range, $?x) \in H$, $(?v, ?p, s) \in R$, $?x \notin X_{\text{rdfs9}}\}$.
    $\mathbb{J} \leftarrow \mathbb{J} \cup \{\{(?x$, rdfs: subClassOf, $o)$, $(?p$, rdfs: range, $?x)$, $(?v, ?p, s)\} \mid (?x, ?p, ?v) \in X_{\text{rdfs3+9}}\}$. // rule rdfs3+rdfs9
    $\mathbb{J} \leftarrow \mathbb{J} \cup \{\{(?x$, rdfs: subClassOf, $o)$, $(?p$, rdfs: range, $?x)$, $(?q$, rdfs: subPropertyOf, $?p)$, $(?v, ?q, s)\} \mid (?x$, rdfs: subClassOf, $o) \in H$, $(?p$, rdfs: range, $?x) \in H$, $(?q$, rdfs: subPropertyOf, $?p) \in H$, $(?v, ?q, s) \in R$, $(?x, ?p, ?v) \notin X_{\text{rdfs3+9}}$, $?x \notin X_{\text{rdfs9}}\}$. // rule rdfs7+rdfs3+rdfs9
    $X_{\text{rdfs2}} \leftarrow \{(?p, ?w) \mid (?p$, rdfs: domain, $o) \in H$, $(s, ?p, ?w) \in R\}$.
    $\mathbb{J} \leftarrow \mathbb{J} \cup \{\{(?p$, rdfs: domain, $o)$, $(s, ?p, ?w)\} \mid (?p, ?w) \in X_{\text{rdfs2}}\}$. // rule rdfs2
    $\mathbb{J} \leftarrow \mathbb{J} \cup \{\{(?p$, rdfs: domain, $o)$, $(?q$, rdfs: subPropertyOf, $?p)$, $(s, ?q, ?w)\} \mid (?p$, rdfs: domain, $o) \in H$, $(?q$, rdfs: subPropertyOf, $?p) \in H$, $(s, ?q, ?w) \in R$, $(?p, ?w) \notin X_{\text{rdfs2}}\}$. // rule rdfs7+rdfs2
    $X_{\text{rdfs3}} \leftarrow \{(?p, ?v) \mid (?p$, rdfs: range, $o) \in H$, $(?v, ?p, s) \in R\}$.
    $\mathbb{J} \leftarrow \mathbb{J} \cup \{\{(?p$, rdfs: range, $o)$, $(?v, ?p, s)\} \mid (?p, ?v) \in X_{\text{rdfs3}}\}$. // rule rdfs3
    $\mathbb{J} \leftarrow \mathbb{J} \cup \{\{(?p$, rdfs: range, $o)$, $(?q$, rdfs: subPropertyOf, $?p)$, $(?v, ?q, s)\} \mid (?p$, rdfs: range, $o) \in H$, $(?q$, rdfs: subPropertyOf, $?p) \in H$, $(?v, ?q, s) \in R$, $(?p, ?v) \notin X_{\text{rdfs3}}\}$. // rule rdfs7+rdfs3
end if
return $\mathbb{J}$.
```

remaining RDF triples of J. $m > 1$ or $n > 1$, or both.

Since H is equal to the RDFS closure of H, schema triples (p_m, rdfs: subPropertyOf, p_0) and (x_n, rdfs: subClassOf, x_0) are in H. ($\{(p_m$, rdfs: subPropertyOf, $p_0)$, $(x_n$, rdfs: subClassOf, $x_0)\} \cup J_{\text{remain}}$) is a justification for t in D and is entailed by J. Therefore, J is not minimal. □

FIND-MIN-JUST$^*_{\text{inst}}$ (Algorithm 1) uses the chains of rules in Table 3.5 to find all minimal justifications for an instance triple t in data D. If t is in D, the only minimal justification for t is $\{t\}$ itself. (Line 5). If t is not in D, FIND-MIN-JUST$^*_{\text{inst}}$ uses the first rule in Table 3.5 for t whose property is not rdf: type (Line 7) and the rest of the rules in Table 3.5 for t with property rdf: type (Lines 9-22). FIND-MIN-JUST$^*_{\text{inst}}$ substitutes the conclusions of these rules with t and searches for all substitutions for the variables in the premises of these rules such that the premises are in D. Then, the premises are minimal justifications for t.

For rules rdfs2+rdfs9, rdfs7+rdfs2+rdfs9, rdfs3+rdfs9, rdfs7+rdfs3+rdfs9, rdfs7+rdfs2, and rdfs7+rdfs3, we exclude substitutions obtained by other rules because justifications formed from these substitutions are not minimal. For example, for rule rdfs2+rdfs9 (Lines 11-12), we exclude substitutions X_{rdfs9} obtained by rule rdfs9 (Line 9).

Example 6 (Excluded Substitutions)**.** *This example illustrates why some substitutions found in FIND-MIN-JUST$^*_{inst}$ (Algorithm 1) are excluded. Suppose FIND-MIN-JUST$^*_{inst}$ is used to find all minimal justifications for the instance triple t = (s, rdf: type, o) in data $\{$(x$'$, rdfs: subClassOf, o), (s, rdf: type, x$'$), (p$'$, rdfs: domain, x$'$), (s, p$'$, w$'$)$\}$. x' is a substitution for the variable ?x in the premises of rule rdfs9 (Line 9), and J_1 = $\{$(x$'$, rdfs: subClassOf, o), (s, rdf: type, x$'$)$\}$ is a minimal justification for t (Line 10). (x', p', w') is a substitution for the variables (?x, ?p, ?w) in the premises of rule rdfs2+rdfs9, and J_2 = $\{$(x$'$, rdfs: subClassOf, o), (p$'$, rdfs: domain, x$'$), (s, p$'$, w$'$)$\}$ is a justification for t. However, J_2 is not minimal because $J_2 \models_{rdfs} J_1$. Therefore, the substitution (x', p', w') is excluded in Line 11.*

Algorithm 2 FIND-MIN-JUST$_{\text{schema}}$(t, D) finds all minimal justifications for a schema triple t in data D.

```
Input: $t$, schema triple; $D$, data.
Output: $\mathbb{J}$, set of all minimal justifications for $t$ in $D$.
 1: $H \leftarrow$ schema data of $D$.
 2: $\mathbb{J} \leftarrow \emptyset$.
 3: if $t \in H$ then
 4:     $\mathbb{J} \leftarrow \{\{t\}\}$.
 5: else
 6:     if property of $t \in$ {rdfs: subPropertyOf, rdfs: subClassOf} then
 7:         $\mathbb{J} \leftarrow$ FIND-MIN-JUST$_{\text{subXOf}}$($t$, $H$). (See Algorithm 3)
 8:     end if
 9: end if
10: return $\mathbb{J}$.
```

3.3.2 Computing Minimal Justifications for Schema Triples

This section describes an algorithm called FIND-MIN-JUST$_{\text{schema}}$ (Algorithm 2), which computes all minimal justifications for a schema triple t in data D. If t is in D, the only minimal justification for t is $\{t\}$ itself (Line 4). The RDFS inference rules that derive schema triples are rules rdfs5, rdfs6, rdfs10, and rdfs11. See Table 2.4 for these rules. Only schema triples with properties rdfs: subPropertyOf and rdfs: subClassOf can be derived, and they are entailed by schema triples only. If t is not in D and the property of t is either rdfs: subPropertyOf or rdfs: subClassOf, FIND-MIN-JUST$_{\text{subXOf}}$ (Algorithm 3) is called to find the justifications for t (Line 7). Otherwise, there are no justifications for t.

FIND-MIN-JUST$_{\text{subXOf}}$ (Algorithm 3) uses backward chaining of rules rdfs5, rdfs6, rdfs10, and rdfs11 to find all minimal justifications for a schema triple t whose property is either rdfs: subPropertyOf or rdfs: subClassOf. FIND-MIN-JUST$_{\text{subXOf}}$ first applies rules rdfs5 and rdfs11 for t with properties rdfs: subPropertyOf and rdfs: subClassOf respectively. It searches for all substitutions for the variable $?a$ in the premises of

Algorithm 3 FIND-MIN-JUST$_{\text{subXOf}}(t, D)$ finds all minimal justifications for a schema triple t whose property is either rdfs: subPropertyOf or rdfs: subClassOf in data D.

```
Input: $t$, schema triple whose property is either rdfs: subPropertyOf or rdfs: subClassOf; $D$, data.
Output: $\mathbb{J}$, set of all minimal justifications for $t$ in $D$.
 1: $s \leftarrow$ subject of $t$; $p \leftarrow$ property of $t$; $o \leftarrow$ object of $t$.
 2: $H \leftarrow$ schema data of $D$.
 3: $\mathbb{J} \leftarrow \emptyset$.
 4: if $t \notin H$ then
 5:     for all $?a \in \{?a \mid ((s, p, ?a), (?a, p, o)) \in \text{rdfs-closure}(H)^2, ?a \neq s, ?a \neq o\}$ do
 6:         $\mathbb{J}_1 \leftarrow$ FIND-MIN-JUST$_{\text{subXOf}}((s, p, ?a), H)$.
 7:         $\mathbb{J}_2 \leftarrow$ FIND-MIN-JUST$_{\text{subXOf}}((?a, p, o), H)$.
 8:         for all $(J_1, J_2) \in \mathbb{J}_1 \times \mathbb{J}_2$ do
 9:             if IS-MINIMAL$_{\text{subXOf}}(J_1 \cup J_2, H)$ = true then  (See Algorithm 4)
10:                 $\mathbb{J} \leftarrow \mathbb{J} \cup \{J_1 \cup J_2\}$.
11:             end if
12:         end for
13:     end for
14:     if $p$ = rdfs: subPropertyOf and $s = o$ then
15:         $\mathbb{J} \leftarrow \mathbb{J} \cup \{\{(s$, rdf: type, rdf: Property$)\}\}$.
16:     else if $p$ = rdfs: subClassOf and $s = o$ then
17:         $\mathbb{J} \leftarrow \mathbb{J} \cup \{\{(s$, rdf: type, rdfs: Class$)\}\}$.
18:     end if
19: else
20:     $\mathbb{J} \leftarrow \{\{t\}\}$.
21: end if
22: return $\mathbb{J}$.
```

rule rdfs5 or rdfs11 such that the premises are in the RDFS closure of the schema data (Line 5). It calls itself recursively until the premises are all in the schema data (Lines 6-7). Justifications found in this way may not be minimal. IS-MINIMAL (Algorithm 4) is called in Line 9 to check if a justification is minimal. FIND-MIN-JUST$_{\text{subXOf}}$ then applies rules rdfs6 and rdfs10 in Lines 15 and 17 for t whose subject and object are equal.

IS-MINIMAL$_{\text{subXOf}}$ (Algorithm 4) is used in Line 9 of FIND-MIN-JUST$_{\text{subXOf}}$ to check if a justification J for an RDF triple whose property is either rdfs: subPropertyOf or rdfs: subClassOf in data D is minimal or not. It applies rule rdfs5 or rdfs11 to check if any part of J entails an RDF triple in D. If it does, J is not minimal.

Algorithm 4 IS-MINIMAL$_{\text{subXOf}}$(J, D) checks if a justification J for an RDF triple whose property is either rdfs: subPropertyOf or rdfs: subClassOf in data D is minimal or not.

```
Input: $J$, justification for an RDF triple whose property is either rdfs: subPropertyOf or rdfs: subClassOf in data $D$; $D$, data.
Output: true or false.
Let $J$ be $\{(a_1, p, a_2), (a_2, p, a_3), \ldots, (a_m, p, a_{m+1})\}$.
for $i = 1$ to $(m - 1)$ do
    for $j = (i + 2)$ to $(m + 1)$ do
        if $(a_i, p, a_j) \in D$ then
            return false.
        end if
    end for
end for
return true.
```

Algorithm 5 COMPU-CLOSURE$_{\text{schema}}$(H) computes the RDFS closure of schema data H.

```
Input: $H$, schema data.
Output: $H_c$, RDFS closure of $H$.
 1: $H_c \leftarrow H$.
 2: for all $p \in$ {rdfs: subClassOf, rdfs: subPropertyOf} do
 3:     $D_{\text{ex}} \leftarrow \emptyset$.
 4:     $D_{\text{not\_ex}} \leftarrow$ RDF triples with property $p$ in $H$.
 5:     while $D_{\text{not\_ex}} \neq \emptyset$ do
 6:         $D_{temp} \leftarrow \{\ (?a, p, ?c) \mid ((?a, p, ?b), (?b, p, ?c)) \in (D_{\text{not\_ex}} \times D_{\text{not\_ex}} \cup D_{\text{not\_ex}} \times D_{\text{ex}} \cup D_{\text{ex}} \times D_{\text{not\_ex}})\ \} \setminus (D_{\text{ex}} \cup D_{\text{not\_ex}})$.
 7:         $D_{\text{ex}} \leftarrow D_{\text{ex}} \cup D_{\text{not\_ex}}$.
 8:         $D_{\text{not\_ex}} \leftarrow D_{temp}$.
 9:     end while
10:     $H_c \leftarrow H_c \cup D_{\text{ex}}$.
11: end for
12: $H_c \leftarrow H_c \cup \{(?p, \text{rdfs: subPropertyOf}, ?p) \mid (?p, \text{rdf: type, rdf: Property}) \in H\}$.
13: $H_c \leftarrow H_c \cup \{(?x, \text{rdfs: subClassOf}, ?x) \mid (?x, \text{rdf: type, rdfs: Class}) \in H\}$.
14: return $H_c$.
```

COMPU-CLOSURE$_{\text{schema}}$ (Algorithm 5) is used in Line 5 of FIND-MIN-JUST$_{\text{subXOf}}$ to compute the RDFS closure of schema data. COMPU-CLOSURE$_{\text{schema}}$ first applies rules rdfs5 and rdfs11 repeatedly until no more RDF triples can be derived (Lines 5-9). Variables D_{ex} and $D_{\text{not\_ex}}$ hold the RDF triples that have and have not been examined respectively. In Line 6, COMPU-CLOSURE$_{\text{schema}}$ only examines the pairs of RDF triples in $D_{\text{not\_ex}} \times D_{\text{not\_ex}}$, $D_{\text{not\_ex}} \times D_{\text{ex}}$, and $D_{\text{ex}} \times D_{\text{not\_ex}}$ because the pairs of RDF triples in $D_{\text{ex}} \times D_{\text{ex}}$ have been examined in the previous iterations. COMPU-CLOSURE$_{\text{schema}}$

then applies rules rdfs6 and rdfs10 to the properties and classes in the schema data to complete the closure.

Example 7 (RDFS Closure of Schema Data)**.** *This example illustrates the use of COMPU-CLOSURE$_{schema}$ (Algorithm 5) to find the RDFS closure of schema data of data D_3 (Table 3.4). COMPU-CLOSURE$_{schema}$ initially assigns all schema data with property rdfs: subPropertyOf to the variable D_{not_ex} (Line 4). In the first iteration of the while loop (Lines 5-9), all pairs of RDF triples in $D_{not\_ex} \times D_{not\_ex}$ are examined, and two additional RDF triples (p, rdfs: subPropertyOf, r) and (p, rdfs: subPropertyOf, s) are derived. (p, rdfs: subPropertyOf, r) and (p, rdfs: subPropertyOf, s) are derived from the pairs ((p, rdfs: subPropertyOf, q), (q, rdfs: subPropertyOf, r)) and ((p, rdfs: subPropertyOf, q), (q, rdfs: subPropertyOf, s)) respectively. No additional triple is derived in the second iteration of the while loop, and the loop ends. Then, COMPU-CLOSURE$_{schema}$ derives four additional RDF triples (p, rdfs: subPropertyOf, p), (q, rdfs: subPropertyOf, q), (r, rdfs: subPropertyOf, r), and (s, rdfs: subPropertyOf, s) for properties p, q, r, and s respectively in Line 12 to complete the closure, which is shown in Table 3.4c.*

Example 8 (Minimal Justifications for a Schema Triple)**.** *This example illustrates the operation of FIND-MIN-JUST$_{schema}$ (Algorithm 2) to find all minimal justifications for the schema triple t = (p, rdfs: subPropertyOf, s) in data D_3 (Table 3.4). Since t is not in D and its property is rdfs: subPropertyOf, FIND-MIN-JUST$_{subXOf}$ (Algorithm 3) is called to find the minimal justifications.*

In Line 6 of FIND-MIN-JUST$_{subXOf}$, two pairs of RDF triples, which entail t, are *found in (rdfs-closure(H_3))^2, where rdfs-closure(H_3) is the RDFS closure of the schema data H_3 of D_3 and is computed in Example 7. One pair is ((p, rdfs: subPropertyOf, q), (q, rdfs: subPropertyOf, s)). FIND-MIN-JUST$_{subXOf}$ is called recursively in Lines 6-7 to find the sets of all minimal justifications for the elements of the pair in H_3. The sets are $\{\{$(p, rdfs: subPropertyOf, q)$\}\}$ and $\{\{$(q, rdfs: subPropertyOf, s)$\}\}$, and they are assigned to $\mathbb{J}_1$ and $\mathbb{J}_2$ respectively. The justification for t formed from $\mathbb{J}_1$ and $\mathbb{J}_2$ is $\{$(p, rdfs: subPropertyOf, q), (q, rdfs: subPropertyOf, s)$\}$. It is passed to IS-MINIMAL$_{subXOf}$ (Algorithm 4) in Line 9 to check if it is minimal. IS-MINIMAL$_{subXOf}$ checks if t is in D_3. t is not in D_3, so the justification is minimal.*

The other pair is ((p, rdfs: subPropertyOf, r), (r, rdfs: subPropertyOf, s)). The sets of all minimal justifications for the elements of the pair are $\{\{$(p, rdfs: subPropertyOf, q), (q, rdfs: subPropertyOf, r)$\}\}$ and $\{\{$(r, rdfs: subPropertyOf, s)$\}\}$. The justification for t formed from them is $\{$(p, rdfs: subPropertyOf, q), (q, rdfs: subPropertyOf, r), (r, rdfs: subPropertyOf, s)$\}$. It is passed to IS-MINIMAL$_{subXOf}$ to check if it is minimal. IS-MINIMAL$_{subXOf}$ checks if (p, rdfs: subPropertyOf, r), t, and (q, rdfs: subPropertyOf, s) are in D_3. (q, rdfs: subPropertyOf, s) is in D_3, so the justification is not minimal. Therefore, the set of all minimal justifications for t in D_3 is $\{\{$(p, rdfs: subPropertyOf, q), (q, rdfs: subPropertyOf, s)$\}\}$.

3.3.3 Computing Minimal Justifications for Instance Triples

This section describes an algorithm called FIND-MIN-JUST$_{\text{inst}}$ (Algorithm 6), which makes use of FIND-MIN-JUST$^*{}_{\text{inst}}$ and FIND-MIN-JUST$_{\text{schema}}$ described in Sections 3.3.1 and 3.3.2 to find all minimal justifications for an instance triple t in data D. Un-

Algorithm 6 FIND-MIN-JUST$_{\text{inst}}$(t, D) finds all minimal justifications for an instance triple t in data D.

```
Input: $t$, instance triple; $D$, data.
Output: $\mathbb{J}$, set of all minimal justifications for $t$ in $D$.
 1: $H \leftarrow$ schema data of $D$.
 2: $\mathbb{J}_{\text{final}} \leftarrow \emptyset$.
 3: $\mathbb{J}_{\text{inst}} \leftarrow$ FIND-MIN-JUST$^*_{\text{inst}}$($t$, $D \cup$ rdfs-closure($H$)). (See Algorithm 1)
 4: for all $J \in \mathbb{J}_{\text{inst}}$ do
 5:     $H_{\text{derived}} = \{x \in J \mid x \notin H\}$.
 6:     if $H_{\text{derived}} = \emptyset$ then
 7:         $\mathbb{J}_{\text{final}} \leftarrow \mathbb{J}_{\text{final}} \cup \{J\}$.
 8:     else
 9:         $J_{\text{derived}} = \prod_{h \in H_{\text{derived}}}$ FIND-MIN-JUST$_{\text{schema}}(h, H)$. (See Algorithm 2)
10:         for all $j_{\text{derived}} \in J_{\text{derived}}$ do
11:             $\mathbb{J}_{\text{final}} \leftarrow \mathbb{J}_{\text{final}} \cup \{(J \setminus H_{\text{derived}}) \cup$ (union of all elements in tuple $j_{\text{derived}}$)$\}$.
12:         end for
13:     end if
14: end for
15: return $\mathbb{J}_{\text{final}}$.
```

like FIND-MIN-JUST$^*_{\text{inst}}$ (Algorithm 1), FIND-MIN-JUST$_{\text{inst}}$ does not require that the schema data H of D equal the RDFS closure of H.

FIND-MIN-JUST$_{\text{inst}}$ first calls FIND-MIN-JUST$^*_{\text{inst}}$ in Line 3 to find all minimal justifications for t in the union of D and the RDFS closure of H. Minimal justifications found in this way may contain derived schema triples. FIND-MIN-JUST$_{\text{inst}}$ then calls FIND-MIN-JUST$_{\text{schema}}$ in Line 9 to find all minimal justifications for each of the derived schema triples. Finally, it replaces the derived schema triples with the minimal justifications for them in Line 11 to give the minimal justifications for t in D.

Example 9 (Minimal Justifications for an Instance Triple)**.** *This example illustrates the operation of FIND-MIN-JUST$_{inst}$ (Algorithm 6) to find all minimal justifications for the instance triple t = (u, s, v) in data D_3 (Table 3.4). FIND-MIN-JUST$_{inst}$ first calls FIND-MIN-JUST$^*_{inst}$ (Algorithm 1) in Line 3 to find all minimal justifications for t in ($D_3 \cup$ rdfs-closure(H_3)), where rdfs-closure(H_3) is the RDFS closure of the schema data H_3 of D_3 and is computed in Example 7. There is only one minimal justification* *for t in (D_3 ∪ rdfs-closure(H_3)), and it is* $\{$*(p, rdfs: subPropertyOf, s), (u, p, v)*$\}$. *(p, rdfs: subPropertyOf, s) in the minimal justification is a derived schema triple.*

FIND-MIN-JUST$_{inst}$ then calls FIND-MIN-JUST$_{schema}$ (Algorithm 2) in Line 9 to find all minimal justifications for the derived schema triple (p, rdfs: subPropertyOf, s) in H_3, and the only minimal justification for it is $\{$*(p, rdfs: subPropertyOf, q), (q, rdfs: subPropertyOf, s)*$\}$*, which is computed in Example 8.*

At last, FIND-MIN-JUST$_{inst}$ replaces the derived schema triple with the minimal justification for it in Line 11 to give the minimal justification for t in D_3, which is $\{$*(p, rdfs: subPropertyOf, q), (q, rdfs: subPropertyOf, s), (u, p, v)*$\}$.

3.3.4 Time Complexity

Three algorithms of finding minimal justifications (FIND-MIN-JUST$^{*}{}_{\text{inst}}$, FIND-MIN-JUST$_{\text{schema}}$, and FIND-MIN-JUST$_{\text{inst}}$) have been described. This section examines the time complexities of them with respect to the data size $|D|$. We assume that the size of schema data $|H|$ is fixed and the size of instance data $|R|$ grows with $|D|$. Moreover, we assume that the schema data H are separated from the instance data R. Otherwise, separating data D by checking the subjects of all RDF triples in D runs in linear time.

FIND-MIN-JUST$^{*}{}_{\text{inst}}$ (Algorithm 1) is about finding substitutions for the variables in the premises of the rules in Table 3.5. We build two indexes for each of the schema and instance data for finding substitutions using the binary search algorithm. One index is sorted by the subjects, properties, and objects of RDF triples. It is used to search for RDF triples and RDF triples with unknown objects. The other index is sorted by the properties and objects of RDF triples. It is used to search for RDF triples with unknown subjects.

There are at most three schema triples and one instance triple in the premises of each rule. The order of finding substitutions for the variables is as follows. We find substitutions for the variables in the schema triples first since $|H|$ is usually much smaller than $|R|$. If there is more than one schema triple, schema triples with fewer variables are chosen. For example, in Line 13 of FIND-MIN-JUST$^{*}_{\text{inst}}$, the order of finding substitutions for rule rdfs7+rdfs2+rdfs9 is the variable $?x$ in ($?x$, rdfs: subClassOf, o), the variable $?p$ in ($?p$, rdfs: domain, $?x$), the variable $?q$ in ($?q$, rdfs: subPropertyOf, $?p$), and the variable $?w$ in (s, $?q$, $?w$).

The time complexity of finding substitutions for each rule is O(log$|D|$). There are at most nine rules to apply. Hence, the overall time complexity of FIND-MIN-JUST$^{*}_{\text{inst}}$ is still O(log$|D|$).

FIND-MIN-JUST$_{\text{schema}}$ (Algorithm 2) works with schema data only since schema data are entailed by schema data only. It does not depend on $|D|$ and runs in constant time.

FIND-MIN-JUST$_{\text{inst}}$ (Algorithm 6) makes use of FIND-MIN-JUST$^{*}_{\text{inst}}$ and FIND-MIN-JUST$_{\text{schema}}$ and runs in logarithmic time.

Chapter 4

pRDFS Consistency

A pRDFS theory is inconsistent if it does not have any satisfying interpretation. This chapter describes a method of checking the consistency of a given pRDFS theory, which is based on [57]. Moreover, for pRDFS theories that are frequently updated, it describes an incremental approach to the consistency checking. The approach does not check the consistency of an updated pRDFS theory anew. It only checks the part affected by the update on the theory and saves the time of unnecessary checking.

4.1 Consistency Checking

We first define a truth value assignment for RDF triples that is not consistent with the RDFS semantics. There are no worlds (RDFS interpretations) that satisfy this assignment.

Definition 12 (Inconsistent truth value assignment)**.** *A truth value assignment* $\tau : D \to \{\boldsymbol{T}, \boldsymbol{F}\}$ *is inconsistent iff* $\{d \in D \mid \tau(d) = \boldsymbol{T}\}$ *rdfs-entails any RDF triple in* $\{d \in$

$D \mid \tau(d) = \boldsymbol{F}\}$, where D is a set of RDF triples.

The following proposition follows from the definitions of pRDFS satisfaction (Definition 8) and a consistent pRDFS theory (Definition 9).

Proposition 2. *A pRDFS theory $(D, \mathcal{P}, \theta)$ is inconsistent iff $\exists\ \tau\ (Pr(\tau) > 0)$, where τ is an inconsistent truth value assignment and $Pr(\tau)$ is the probability of τ computed from the theory.*

Proof. Assume $\exists\ \tau\ (Pr(\tau) > 0)$ is true. Let τ' be an inconsistent truth assignment such that $Pr(\tau') > 0$. There is no world that satisfies τ'. By the definition of pRDFS satisfaction (Definition 8), for any pRDFS interpretation that satisfies the theory, $Pr(\tau') = 0$, but $Pr(\tau') > 0$. The theory does not have any satisfying interpretation and is inconsistent.

We prove the contrapositive of the statement that if a pRDFS theory is inconsistent, $\exists\ \tau\ (Pr(\tau) > 0)$ is true. Suppose $\exists\ \tau\ (Pr(\tau) > 0)$ is false, or equivalently $\forall\ \tau$ $(Pr(\tau) = 0)$ is true. We can construct a satisfying interpretation I of the theory. The interpretation I maps each world W in $\mathbb{W}_{\tau''}$ to the probability value $Pr(\tau'')/|\mathbb{W}_{\tau''}|$ for all consistent truth value assignments τ'', where $\mathbb{W}_{\tau''}$ is the set of worlds that satisfies τ''. Therefore, the theory is consistent. □

Proposition 2 suggests that we can perform consistency checking of a pRDFS theory by checking that the probabilities of all inconsistent truth value assignments are zeros. Mathematically, the condition for the consistency of a theory with data D is (4.1).

$$\forall d \in D\, \forall K \in \{X \subseteq (D \setminus \{d\}) \mid X \models_{rdfs} d\} Pr(d = \mathbf{F} \wedge \bigwedge_{k \in K} k = \mathbf{T}) = 0. \quad (4.1)$$

(4.1) can be simplified as (4.2) by means of justifications defined in Definition 10. This is proved in Proposition 3.

$$\forall d \in D\, \forall J \in JUST(d, D \setminus \{d\}) Pr(d = \mathbf{F} \wedge \bigwedge_{j \in J} j = \mathbf{T}) = 0. \quad (4.2)$$

Proposition 3. *If* $\forall d \in D\, \forall J \in$ *JUST*$(d, D \setminus \{d\})$ *Pr*$(d = \boldsymbol{F} \wedge \bigwedge_{j \in J} j = \boldsymbol{T})$ *= 0, then* $\forall d \in D\, \forall K \in \{X \subseteq (D \setminus \{d\}) \mid X \models_{rdfs} d\} Pr(d = \boldsymbol{F} \wedge \bigwedge_{k \in K} k = \boldsymbol{T}) = 0$*, where* D *is the data of a pRDFS theory and JUST(x, X) is the set of all justifications for an RDF triple* x *in data* X*.*

Proof. For any $d \in D$ and any $K \in \{x \subseteq (D \setminus \{d\}) \mid x \models_{rdfs} d\}$, if $K \in$ JUST$(d, D \setminus \{d\})$, it is given that $Pr(d = \mathbf{F} \wedge \bigwedge_{k \in K} k = \mathbf{T}) = 0$. If $K \notin$ JUST$(d, D \setminus \{d\})$, $\exists K' \in$ JUST$(d, D \setminus \{d\})$ such that $K' \subset K$. $Pr(d = \mathbf{F} \wedge \bigwedge_{k \in K} k = \mathbf{T}) \leq Pr(d = \mathbf{F} \wedge \bigwedge_{k \in K'} k = \mathbf{T}) = 0$. □

(4.2) can be further simplified as (4.3) by means of minimal justifications defined in Definition 11. This is proved in Proposition 4.

$$\forall d \in D\, \forall J \in JUST_{\text{min}}(d, D \setminus \{d\}) Pr(d = \mathbf{F} \wedge \bigwedge_{j \in J} j = \mathbf{T}) = 0. \quad (4.3)$$

Proposition 4. *If* $\forall d \in D\, \forall J \in$ *JUST*$_{min}(d, D \setminus \{d\})$ *Pr*$(d = \boldsymbol{F} \wedge \bigwedge_{j \in J} j = \boldsymbol{T})$ *= 0,*

$\bigwedge_{j \in J}$	$\bigwedge_{j \in J'} j$	$\bigwedge_{j \in J} j \vee \bigwedge_{j \in J'} j$
T	**T**	**T**
F	**T**	**T**
F	**F**	**F**

Table 4.1: Truth table for $\bigwedge_{j \in J}$, $\bigwedge_{j \in J'} j$, and $\bigwedge_{j \in J} j \vee \bigwedge_{j \in J'} j$, where J and J' are non-minimal and minimal justifications for an RDF triple respectively and $J \models_{rdfs} J'$.

*then $\forall d \in D\ \forall J \in$ JUST($d, D \setminus \{d\}$) Pr($d =$ **F** $\wedge \bigwedge_{j \in J} j =$ **T**) = 0, where D is the data of a pRDFS theory, JUST(x, X) is the set of all justifications for an RDF triple x in data X, and JUST$_{min}$(x, X) is the set of all minimal justifications for an RDF triple x in data X.*

Proof. For any d in D and any J in JUST($d, D \setminus \{d\}$), if $J \in$ JUST$_{\text{min}}$($d, D \setminus \{d\}$), it is given that Pr($d =$ **F** $\wedge \bigwedge_{j \in J} j =$ **T**) = 0. If $J \notin$ JUST$_{\text{min}}$($d, D \setminus \{d\}$), by the definition of a minimal justification (Definition 11), there exists a J' in JUST$_{\text{min}}$($d, D \setminus \{d\}$) such that $J \models_{rdfs} J'$.

$$Pr(d = \mathbf{F} \wedge \bigwedge_{j \in J} j = \mathbf{T}) \tag{4.4}$$

$$\leq Pr(d = \mathbf{F} \wedge (\bigwedge_{j \in J} j \vee \bigwedge_{j \in J'} j) = \mathbf{T}) \tag{4.5}$$

$$= Pr(d = \mathbf{F} \wedge \bigwedge_{j \in J'} j = \mathbf{T}) = 0 \tag{4.6}$$

The expression ($\bigwedge_{j \in J} j \vee \bigwedge_{j \in J'} j$) in (4.5) is equivalent to $\bigwedge_{j \in J'} j$ in (4.6). See the truth table for them in Table 4.1. When $\bigwedge_{j \in J} j$ is true, $\bigwedge_{j \in J'} j$ is also true because $J \models_{rdfs} J'$. $\bigwedge_{j \in J} j \vee \bigwedge_{j \in J'} j$ = **T** $\vee$ **T** = **T** is also true. When $\bigwedge_{j \in J} j$ is false, $\bigwedge_{j \in J} j \vee \bigwedge_{j \in J'} j$ = **F** $\vee \bigwedge_{j \in J'} j$ = $\bigwedge_{j \in J'} j$. □

IS-CONSISTENT (Algorithm 7) uses (4.3) to check the consistency of a pRDFS

Algorithm 7 IS-CONSISTENT($(D, \mathcal{P}, \theta)$) determines if a pRDFS theory $(D, \mathcal{P}, \theta)$ is consistent or not.

```
Input: $(D, \mathcal{P}, \theta)$, pRDFS theory.
Output: true or false.
 1: for all $d \in D$ such that $Pr(d = \mathbf{F}) > 0$ do
 2:     if $d$ is an instance triple then
 3:         $\mathbb{J} \leftarrow$ FIND-MIN-JUST$_{\text{inst}}$($d$, $D \setminus \{d\}$). (See Algorithm 6)
 4:     else
 5:         $\mathbb{J} \leftarrow$ FIND-MIN-JUST$_{\text{schema}}$($d$, $D \setminus \{d\}$). (See Algorithm 2)
 6:     end if
 7:     for all $J \in \mathbb{J}$ do
 8:         if $Pr(d = \mathbf{F} \wedge \bigwedge_{j \in J} j = \mathbf{T}) > 0$ return false.
 9:     end for
10: end for
11: return true.
```

Algorithm 8 IS-CONSISTENT$_{\text{trueSchema}}$($(D, \mathcal{P}, \theta)$) determines if a pRDFS theory $(D, \mathcal{P}, \theta)$ is consistent or not, where the schema data of D are true.

```
Input: $(D, \mathcal{P}, \theta)$, pRDFS theory, where the schema data of $D$ are certain.
Output: true or false.
 1: $H \leftarrow$ schema data of $D$; $R \leftarrow$ instance data of $D$.
 2: for all $d \in R$ such that $Pr(d = \mathbf{F}) > 0$ do
 3:     $\mathbb{J} \leftarrow$ FIND-MIN-JUST$^{*}_{\text{inst}}$($d$, ($D \cup$ rdfs-closure($H$)) $\setminus \{d\}$). (See Algorithm 1)
 4:     for all $J \in \mathbb{J}$ do
 5:         if $Pr(d = \mathbf{F} \wedge \bigwedge_{j \in (J \cap R)} j = \mathbf{T}) > 0$ return false.
 6:     end for
 7: end for
 8: return true.
```

theory. The for loop in Lines 1-10 iterates over the RDF triples that are not certainly true. If an RDF triple d' is certainly true, the probability $Pr(d' = \mathbf{F} \wedge \bigwedge_{j \in J} j = \mathbf{T})$ in Line 8 must be zero. Therefore, we can skip the checking of d'. In the for loop, IS-CONSISTENT first finds all minimal justifications in Line 3 or 5. It then checks the probabilities of the inconsistent truth value assignments created from the minimal justifications in Line 8.

IS-CONSISTENT$_{\text{trueSchema}}$ (Algorithm 8) is a simplified version of IS-CONSISTENT (Algorithm 7) for pRDFS theories whose schema data are true. The for loop in Lines 2-7 iterates over instance data only. IS-CONSISTENT$_{\text{trueSchema}}$ calls a simpler algorithm FIND-MIN-JUST$^{*}_{\text{inst}}$ (Algorithm 1) in Line 3 to find minimal justifications. Although the justifications may contain derived schema data, this does not affect the probability of the inconsistent truth value assignment in Line 5 since both the declared and derived schema data are true.

4.2 Incremental Consistency Checking

The running time of consistency checking could be long if both the data size and the percentage of RDF triples that are not certainly true are large. However, we perform the checking only once on static pRDFS theories. For pRDFS theories that are frequently updated, we provide an incremental approach to the consistency checking. The approach does not check the consistency of an updated pRDFS theory anew. It only checks the part affected by the update on the theory and saves the time of unnecessary checking.

There are two basic operations to update a pRDFS theory. One is removing an RDF triple and its probability information from the theory. This operation does not void the consistency of the theory.

The other operation is adding an RDF triple t and its probability information to the theory. Given that the theory before the addition of t is consistent, the probabilities of the inconsistent truth value assignments that do not contain t are zero. We only need to check the probabilities of the inconsistent truth value assignments that contain t. Mathematically, the conditions for the consistency of a consistent pRDFS theory with data D after adding an RDF triple t and its probability information to the theory are

Algorithm 9 IS-CONSISTENT$_{\text{inc}}((D, \mathcal{P}, \theta), t)$ determines if a consistent pRDFS theory $(D, \mathcal{P}, \theta)$ is still consistent after adding an RDF triple $t \notin D$ and its probability information to the theory.

```
Input: $(D, \mathcal{P}, \theta)$, consistent pRDFS theory; $t$, RDF triple not in $D$.
Output: true or false.
if $t$ is an instance triple then
    $\mathbb{J} \leftarrow$ FIND-MIN-JUST$_{\text{inst}}(t, D)$. (See Algorithm 6)
else
    $\mathbb{J} \leftarrow$ FIND-MIN-JUST$_{\text{schema}}(t, D)$. (See Algorithm 2)
end if
for all $J \in \mathbb{J}$ do
    if $Pr(t = \mathbf{F} \wedge \bigwedge_{j \in J} j = \mathbf{T}) > 0$ return false.
end for
if $t$ is an instance triple then
    PAIRS $\leftarrow$ FIND-TRIPLE-JUST-PAIR$_{\text{inst}}(t, D)$. (See Algorithm 10)
else
    PAIRS $\leftarrow$ FIND-ITRIPLE-JUST-PAIR$_{\text{schema}}(t, D)$. (See Algorithms 12 and 13)
    PAIRS $\leftarrow$ PAIRS $\cup$ FIND-STRIPLE-JUST-PAIR$_{\text{schema}}(t, D)$. (See Algorithm 14)
end if
for all $(t', J') \in$ PAIRS do
    if $Pr(t' = \mathbf{F} \wedge \bigwedge_{j \in J'} j = \mathbf{T}) > 0$ return false.
end for
return true.
```

(4.7) and (4.8).

$$\forall J \in JUST_{\text{min}}(t, D)\ (Pr(t = \mathbf{F} \wedge \bigwedge_{j \in J} j = \mathbf{T}) = 0). \tag{4.7}$$

$$\begin{aligned} &\forall d \in D\ \forall J \in \{X \in JUST_{\text{min}}(d, (D \cup \{t\}) \setminus \{d\}) \mid t \in X\} \\ &(Pr(d = \mathbf{F} \wedge \bigwedge_{j \in J} j = \mathbf{T}) = 0). \end{aligned} \tag{4.8}$$

In (4.7), an inconsistent truth value assignment is constructed from t and a minimal justification for t. In (4.8), it is constructed from an RDF triple and a minimal justification for the RDF triple such that the minimal justification contains t.

IS-CONSISTENT$_{\text{inc}}$ (Algorithm 9) uses (4.7) and (4.8) to check if a consistent pRDFS theory is still consistent after the addition of an RDF triple t and its probability information. It consists of two parts. The first part of IS-CONSISTENT$_{\text{inc}}$ (Lines 1-8) corresponds to checking (4.7). It finds all minimal justifications for t using FIND-MIN-JUST$_{\text{inst}}$ (Algorithm 6) and FIND-MIN-JUST$_{\text{schema}}$ (Algorithm 2), which are described in Section 3.3.

The second part of IS-CONSISTENT$_{\text{inc}}$ (Lines 9-17) corresponds to checking (4.8). It finds all ordered pairs of an RDF triple d in data D and a minimal justification J for d in $((D \cup \{t\}) \setminus \{d\})$ such that $t \in J$.

If t is an instance triple, d must be an instance triple because schema triples are entailed by schema data only, and justifications for schema triples do not contain any instance triple. FIND-TRIPLE-JUST-PAIR$_{\text{inst}}$ (Algorithm 10) is called to find the pairs. It first finds minimal justifications for d in $((D \cup \text{rdfs-closure}(H) \cup \{t\}) \setminus \{d\})$ such that the minimal justifications contain t, so the chains of rules in Table 3.5 can be applied, where rdfs-closure(H) is the RDFS closure of the schema data H of D. FIND-TRIPLE-JUST-PAIR$_{\text{inst}}$ substitutes t for an instance triple in the premises of each rule and searches for all substitutions for the variables in the rules such that the premises of the rules are in $((D \cup \text{rdfs-closure}(H) \cup \{t\}) \setminus \{d\})$ and the conclusions of the rules are in D. The premises are the minimal justifications and may contain derived schema data. REPLACE-DERIVED-TRIPLE (Algorithm 11) is called in Line 17. It calls FIND-MIN-JUST$_{\text{schema}}$ (Algorithm 2) to find all minimal justifications for the derived schema data in Line 8 and replaces the derived schema data with the minimal justifications for them in Line 10 to give the final result.

If t is a schema triple and d is an instance triple, FIND-ITRIPLE-JUST-PAIR$_{\text{schema}}$

Algorithm 10 FIND-TRIPLE-JUST-PAIR$_{\text{inst}}(t, D)$ finds all pairs of an RDF triple d in data D and a minimal justification J for d in $((D \cup \{t\}) \setminus \{d\})$ such that $t \in J$, where t is an instance triple not in D.

```
Input: $D$, data; $t$, instance triple not in $D$.
Output: PAIRS, set of pairs of an RDF triple $d$ in $D$ and a minimal justification $J$ for $d$ in $((D \cup \{t\}) \setminus \{d\})$ such that $t \in J$.
PAIRS $\leftarrow \emptyset$.
$s \leftarrow$ subject of $t$; $p \leftarrow$ property of $t$; $o \leftarrow$ object of $t$.
$H \leftarrow$ schema data of $D$; $R \leftarrow$ instance data of $D$.
if $p$ = rdf: type then
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((s,$ rdf: type, $?y), \{(o,$ rdfs: subClassOf, $?y), t\}\big) \mid (s,$ rdf: type, $?y) \in R, (o,$ rdfs: subClassOf, $?y) \in$ rdfs-closure$(H)\}$. // rule rdfs9
else
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((s, ?q, o), \{(p,$ rdfs: subPropertyOf, $?q), t\}\big) \mid (s, ?q, o) \in R, (p,$ rdfs: subPropertyOf, $?q) \in$ rdfs-closure$(H)\}$. // rule rdfs7
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((s,$ rdf: type, $?x), \{(p,$ rdfs: domain, $?x), t\}\big) \mid (s,$ rdf: type, $?x) \in R, (p,$ rdfs: domain, $?x) \in$ rdfs-closure$(H)\}$. // rule rdfs2
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((s,$ rdf: type, $?x), \{(?q,$ rdfs: domain, $?x), (p,$ rdfs: subPropertyOf, $?q), t\}\big) \mid (s,$ rdf: type, $?x) \in R, (?q,$ rdfs: domain, $?x) \in$ rdfs-closure$(H), (p,$ rdfs: subPropertyOf, $?q) \in$ rdfs-closure$(H), ?q \neq p\}$. // rule rdfs7+rdfs2
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((s,$ rdf: type, $?y), \{(?x,$ rdfs: subClassOf, $?y), (p,$ rdfs: domain, $?x), t\}\big) \mid (s,$ rdf: type, $?y) \in R, (?x,$ rdfs: subClassOf, $?y) \in$ rdfs-closure$(H), (p,$ rdfs: domain, $?x) \in$ rdfs-closure$(H), ?x \neq ?y\}$. // rule rdfs2+rdfs9
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((s,$ rdf: type, $?y), \{(?x,$ rdfs: subClassOf, $?y), (?q,$ rdfs: domain, $?x), (p,$ rdfs: subPropertyOf, $?q), t\}\big) \mid (s,$ rdf: type, $?y) \in R, (?x,$ rdfs: subClassOf, $?y) \in$ rdfs-closure$(H), (?q,$ rdfs: domain, $?x) \in$ rdfs-closure$(H), (p,$ rdfs: subPropertyOf, $?q), \in$ rdfs-closure$(H), ?x \neq ?y, ?q \neq p\}$. // rule rdfs7+rdfs2+rdfs9
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((o,$ rdf: type, $?x), \{(p,$ rdfs: range, $?x), t\}\big) \mid (o,$ rdf: type, $?x) \in R, (p,$ rdfs: range, $?x) \in$ rdfs-closure$(H)\}$. // rule rdfs3
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((o,$ rdf: type, $?x), \{(?q,$ rdfs: range, $?x), (p,$ rdfs: subPropertyOf, $?q), t\}\big) \mid (o,$ rdf: type, $?x) \in R, (?q,$ rdfs: range, $?x) \in$ rdfs-closure$(H), (p,$ rdfs: subPropertyOf, $?q) \in$ rdfs-closure$(H), ?q \neq p\}$. // rule rdfs7+rdfs3
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((o,$ rdf: type, $?y), \{(?x,$ rdfs: subClassOf, $?y), (p,$ rdfs: range, $?x), t\}\big) \mid (o,$ rdf: type, $?y) \in R, (?x,$ rdfs: subClassOf, $?y) \in$ rdfs-closure$(H), (p,$ rdfs: range, $?x) \in$ rdfs-closure$(H), ?x \neq ?y\}$. // rule rdfs3+rdfs9
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((o,$ rdf: type, $?y), \{(?x,$ rdfs: subClassOf, $?y), (?q,$ rdfs: range, $?x), (p,$ rdfs: subPropertyOf, $?q), t\}\big) \mid (o,$ rdf: type, $?y) \in R, (?x,$ rdfs: subClassOf, $?y) \in$ rdfs-closure$(H), (?q,$ rdfs: range, $?x) \in$ rdfs-closure$(H), (p,$ rdfs: subPropertyOf, $?q) \in$ rdfs-closure$(H), ?x \neq ?y, ?q \neq p\}$. // rule rdfs7+rdfs3+rdfs9
end if
return REPLACE-DERIVED-TRIPLE(PAIRS, $D \cup \{t\}$). (See Algorithm 11)
```

(Algorithms 12 and 13) is called to find the pairs. It first finds minimal justifications for d in $((D \cup \text{rdfs-closure}(H \cup \{t\})) \setminus \{d\})$ such that the minimal justifications contain t or any derived schema triple in $(\text{rdfs-closure}(H \cup \{t\}) \setminus \text{rdfs-closure}(H))$, so the chains of rules in Table 3.5 can be applied. FIND-ITRIPLE-JUST-PAIR$_{\text{schema}}$

Algorithm 11 REPLACE-DERIVED-TRIPLE(PAIRS, D) replaces each derived schema triple t in PAIRS with a minimal justification for t in data D.

```
Input: $D$, data; PAIRS, set of pairs of an RDF triple $d$ in $D$ and a minimal justification for $d$ in $\big((D \cup \text{rdfs-closure}(H)) \setminus \{d\}\big)$, where $H$ is schema data of $D$.
Output: PAIRS$'$, set of pairs of an RDF triple $d$ in $D$ and a minimal justification for $d$ in $(D \setminus \{d\})$.
PAIRS$' \leftarrow \emptyset$.
$H \leftarrow$ schema data of $D$.
for all $(d, J) \in$ PAIRS do
    $H_{\text{derived}} = \{x \in J \mid x \notin H\}$.
    if $H_{\text{derived}} = \emptyset$ then
        PAIRS$' \leftarrow$ PAIRS$' \cup \{(d, J)\}$.
    else
        $J_{\text{derived}} = \prod_{h \in H_{\text{derived}}}$ FIND-MIN-JUST$_{\text{schema}}(h, H)$. (See Algorithm 2)
        for all $j_{\text{derived}} \in J_{\text{derived}}$ do
            PAIRS$' \leftarrow$ PAIRS$' \cup \big\{\big(d, (J \setminus H_{\text{derived}}) \cup (\text{union of all elements in tuple } j_{\text{derived}})\big)\big\}$.
        end for
    end if
end for
return PAIRS$'$.
```

substitutes t or a derived schema triple in $(\text{rdfs-closure}(H \cup \{t\}) \setminus \text{rdfs-closure}(H))$ for a schema triple in the premises of each rule and searches for all substitutions for the variables in the rules such that the premises of the rules are in $((D \cup \text{rdfs-closure}(H \cup \{t\})\}) \setminus \{d\})$ and the conclusions of the rules are in D. The premises are the minimal justifications and may contain derived schema data. FIND-ITRIPLE-JUST-PAIR$_{\text{schema}}$ calls REPLACE-DERIVED-TRIPLE (Algorithm 11) in Line 35 to replace the derived schema data with the minimal justifications for them to give the final result.

If both t and d are schema triples, FIND-STRIPLE-JUST-PAIR$_{\text{schema}}$ (Algorithm 14) is called to find the pairs. It first uses rules rdfs5, rdfs6, rdfs10, and rdfs11 in Table 2.4 to find minimal justifications for d in $((D \cup \text{rdfs-closure}(H \cup \{t\})\}) \setminus \{d\})$ such that the minimal justifications contain t. The minimal justifications found may contain derived schema data. FIND-STRIPLE-JUST-PAIR$_{\text{schema}}$ then calls REPLACE-DERIVED-TRIPLE (Algorithm 11) in Line 17 to replace the derived schema data with the minimal justifications for them to give the final result.

Algorithm 12 FIND-ITRIPLE-JUST-PAIR$_{\text{schema}}(t,\ D)$ finds all pairs of an instance triple d in data D and a minimal justification J for d in $((D \cup \{t\}) \setminus \{d\})$ such that $t \in J$, where t is a schema triple not in D (Part 1).

```
Input: $D$, data; $t$, schema triple not in $D$.
Output: PAIRS, set of pairs of an instance triple $d$ in $D$ and a minimal justification $J$ for $d$ in $((D \cup \{t\}) \setminus \{d\})$ such that $t \in J$.
PAIRS $\leftarrow \emptyset$.
$H \leftarrow$ schema data of $D$; $R \leftarrow$ instance data of $D$.
for all $(s_h, p_h, o_h) \in \big(\{t\} \cup (\text{rdfs-closure}(H \cup \{t\}) \setminus \text{rdfs-closure}(H))\big)$ do
    if $p_h$ = rdfs: domain then
        $X_{\text{rdfs2}} \leftarrow \{(?u, ?w) \mid (?u, \text{rdf: type}, o_h) \in R, (?u, s_h, ?w) \in R\}$.
        PAIRS $\leftarrow$ PAIRS $\cup \{\big((?u, \text{rdf: type}, o_h), \{(s_h, \text{rdfs: domain}, o_h), (?u, s_h, ?w)\}\big) \mid (?u, ?w) \in X_{\text{rdfs2}}\}$. // rule rdfs2
        PAIRS $\leftarrow$ PAIRS $\cup \{\big((?u, \text{rdf: type}, o_h), \{(s_h, \text{rdfs: domain}, o_h), (?q, \text{rdfs: subPropertyOf}, s_h), (?u, ?q, ?w)\}\big) \mid (?u, \text{rdf: type}, o_h) \in R, (?q, \text{rdfs: subPropertyOf}, s_h) \in \text{rdfs-closure}(H), (?u, ?q, ?w) \in R, ?q \neq s_h, (?u, ?w) \notin X_{\text{rdfs2}}\}$. // rule rdfs7+rdfs2
        $X_{\text{rdfs2+9}} \leftarrow \{(?u, ?y, ?w) \mid (?u, \text{rdf: type}, ?y) \in R, (o_h, \text{rdfs: subClassOf}, ?y) \in \text{rdfs-closure}(H), (?u, s_h, ?w) \in R, ?y \neq o_h\}$.
        PAIRS $\leftarrow$ PAIRS $\cup \{\big((?u, \text{rdf: type}, ?y), \{(o_h, \text{rdfs: subClassOf}, ?y), (s_h, \text{rdfs: domain}, o_h), (?u, s_h, ?w)\}\big) \mid (?u, ?y, ?w) \in X_{\text{rdfs2+9}}\}$. // rule rdfs2+rdfs9
        PAIRS $\leftarrow$ PAIRS $\cup \{\big((?u, \text{rdf: type}, ?y), \{(o_h, \text{rdfs: subClassOf}, ?y), (s_h, \text{rdfs: domain}, o_h), (?q, \text{rdfs: subPropertyOf}, s_h), (?u, ?q, ?w)\}\big) \mid (?u, \text{rdf: type}, ?y) \in R, (o_h, \text{rdfs: subClassOf}, ?y) \in \text{rdfs-closure}(H), (?q, \text{rdfs: subPropertyOf}, s_h) \in \text{rdfs-closure}(H), (?u, ?q, ?w) \in R, ?y \neq o_h, ?q \neq s_h, (?u, ?y, ?w) \notin X_{\text{rdfs2+9}}\}$. // rule rdfs7+rdfs2+rdfs9
    else if $p_h$ = rdfs: range then
        $X_{\text{rdfs3}} \leftarrow \{(?u, ?v) \mid (?u, \text{rdf: type}, o_h) \in R, (?v, s_h, ?u) \in R\}$.
        PAIRS $\leftarrow$ PAIRS $\cup \{\big((?u, \text{rdf: type}, o_h), \{(s_h, \text{rdfs: range}, o_h), (?v, s_h, ?u)\}\big) \mid (?u, ?v) \in X_{\text{rdfs3}}\}$. // rule rdfs3
        PAIRS $\leftarrow$ PAIRS $\cup \{\big((?u, \text{rdf: type}, o_h), \{(s_h, \text{rdfs: range}, o_h), (?q, \text{rdfs: subPropertyOf}, s_h), (?v, ?q, ?u)\}\big) \mid (?u, \text{rdf: type}, o_h) \in R, (?q, \text{rdfs: subPropertyOf}, s_h) \in \text{rdfs-closure}(H), (?v, ?q, ?u) \in R, ?q \neq s_h, (?u, ?v) \notin X_{\text{rdfs3}}\}$. // rule rdfs7+rdfs3
        $X_{\text{rdfs3+9}} \leftarrow \{(?u, ?y, ?v) \mid (?u, \text{rdf: type}, ?y) \in R, (o_h, \text{rdfs: subClassOf}, ?y) \in \text{rdfs-closure}(H), (?v, s_h, ?u) \in R, ?y \neq o_h\}$.
        PAIRS $\leftarrow$ PAIRS $\cup \{\big((?u, \text{rdf: type}, ?y), \{(o_h, \text{rdfs: subClassOf}, ?y), (s_h, \text{rdfs: range}, o_h), (?v, s_h, ?u)\}\big) \mid (?u, ?y, ?v) \in X_{\text{rdfs3+9}}\}$. // rule rdfs3+rdfs9
        PAIRS $\leftarrow$ PAIRS $\cup \{\big((?u, \text{rdf: type}, ?y), \{(o_h, \text{rdfs: subClassOf}, ?y), (s_h, \text{rdfs: range}, o_h), (?q, \text{rdfs: subPropertyOf}, s_h), (?v, ?q, ?u)\}\big) \mid (?u, \text{rdf: type}, ?y) \in R, (o_h, \text{rdfs: subClassOf}, ?y) \in \text{rdfs-closure}(H), (?q, \text{rdfs: subPropertyOf}, s_h) \in \text{rdfs-closure}(H), (?v, ?q, ?u) \in R, ?y \neq o_h, ?q \neq s_h, (?u, ?y, ?v) \notin X_{\text{rdfs3+9}}\}$. // rule rdfs7+rdfs3+rdfs9
```

Example 10 (Consistency Checking After Adding an Instance Triple)**.** *This example illustrates the operation of IS-CONSISTENT$_{inc}$ (Algorithm 9) using a consistent theory with data D_4 (Table 4.2). IS-CONSISTENT$_{inc}$ is used to check the consistency of the theory after the instance triple $t = (u, p, v)$ and its probability information are added to the theory, where u and v are individuals and p is a property.*

Algorithm 13 FIND-ITRIPLE-JUST-PAIR$_{\text{schema}}(t, D)$ finds all pairs of an instance triple d in data D and a minimal justification J for d in $((D \cup \{t\}) \setminus \{d\})$ such that $t \in J$, where t is a schema triple not in D (Part 2).

```
    else if $p_h$ = rdfs: subPropertyOf then
        PAIRS ← PAIRS ∪ $\{((?u, o_h, ?w), \{(s_h,$ rdfs: subPropertyOf, $o_h), (?u, s_h, ?w)\})$ | $(?u, o_h, ?w) \in R, (?u, s_h, ?w) \in R\}$. // rule rdfs7
        PAIRS ← PAIRS ∪ $\{((?u,$ rdf: type, $?x), \{(o_h,$ rdfs: domain, $?x), (s_h,$ rdfs: subPropertyOf, $o_h), (?u, s_h, ?w)\})$ | $(?u,$ rdf: type, $?x) \in R, (o_h,$ rdfs: domain, $?x) \in$ rdfs-closure($H$), $(?u, s_h, ?w) \in R\}$. // rule rdfs7+rdfs2
        PAIRS ← PAIRS ∪ $\{((?u,$ rdf: type, $?y), \{(?x,$ rdfs: subClassOf, $?y), (o_h,$ rdfs: domain, $?x), (s_h,$ rdfs: subPropertyOf, $o_h), (?u, s_h, ?w)\})$ | $(?u,$ rdf: type, $?y) \in R, (?x,$ rdfs: subClassOf, $?y) \in$ rdfs-closure($H$), $(o_h,$ rdfs: domain, $?x) \in$ rdfs-closure($H$), $(?u, s_h, ?w) \in R, ?x \neq ?y\}$. // rule rdfs7+rdfs2+rdfs9
        PAIRS ← PAIRS ∪ $\{((?u,$ rdf: type, $?x), \{(o_h,$ rdfs: range, $?x), (s_h,$ rdfs: subPropertyOf, $o_h), (?v, s_h, ?u)\})$ | $(?u,$ rdf: type, $?x) \in R, (o_h,$ rdfs: range, $?x) \in$ rdfs-closure($H$), $(?v, s_h, ?u) \in R\}$. // rule rdfs7+rdfs3
        PAIRS ← PAIRS ∪ $\{((?u,$ rdf: type, $?y), \{(?x,$ rdfs: subClassOf, $?y), (o_h,$ rdfs: range, $?x), (s_h,$ rdfs: subPropertyOf, $o_h), (?v, s_h, ?u)\})$ | $(?u,$ rdf: type, $?y) \in R, (?x,$ rdfs: subClassOf, $?y) \in$ rdfs-closure($H$), $(o_h,$ rdfs: range, $?x) \in$ rdfs-closure($H$), $(?v, s_h, ?u) \in R, ?x \neq ?y\}$. // rule rdfs7+rdfs3+rdfs9
    else if $p_h$ = rdfs: subClassOf then
        $X_{\text{rdfs9}} \leftarrow \{?u$ | $(?u,$ rdf: type, $o_h) \in R, (?u,$ rdf: type, $s_h) \in R\}$.
        PAIRS ← PAIRS ∪ $\{((?u,$ rdf: type, $o_h), \{(s_h,$ rdfs: subClassOf, $o_h), (?u,$ rdf: type, $s_h)\})$ | $?u \in X_{\text{rdfs9}}\}$. // rule rdfs9
        $X_{\text{rdfs2+9}} \leftarrow \{(?u, ?p, ?w)$ | $(?u,$ rdf: type, $o_h) \in R, (?p,$ rdfs: domain, $s_h) \in$ rdfs-closure($H$), $(?u, ?p, ?w) \in R, ?u \notin X_{\text{rdfs9}}\}$.
        PAIRS ← PAIRS ∪ $\{((?u,$ rdf: type, $o_h), \{(s_h,$ rdfs: subClassOf, $o_h), (?p,$ rdfs: domain, $s_h), (?u, ?p, ?w)\})$ | $(?u, ?p, ?w) \in X_{\text{rdfs2+9}}\}$. // rule rdfs2+rdfs9
        PAIRS ← PAIRS ∪ $\{((?u,$ rdf: type, $o_h), \{(s_h,$ rdfs: subClassOf, $o_h), (?p,$ rdfs: domain, $s_h), (?q,$ rdfs: subPropertyOf, $?p), (?u, ?q, ?w)\})$ | $(?u,$ rdf: type, $o_h) \in R, (?p,$ rdfs: domain, $s_h) \in$ rdfs-closure($H$), $(?q,$ rdfs: subPropertyOf, $?p) \in$ rdfs-closure($H$), $(?u, ?q, ?w) \in R, (?u, ?p, ?w) \notin X_{\text{rdfs2+9}}\}$. // rule rdfs7+rdfs2+rdfs9
        $X_{\text{rdfs3+9}} \leftarrow \{(?u, ?p, ?v)$ | $(?u,$ rdf: type, $o_h) \in R, (?p,$ rdfs: range, $s_h) \in$ rdfs-closure($H$), $(?v, ?p, ?u) \in R, ?u \notin X_{\text{rdfs9}}\}$.
        PAIRS ← PAIRS ∪ $\{((?u,$ rdf: type, $o_h), \{(s_h,$ rdfs: subClassOf, $o_h), (?p,$ rdfs: range, $s_h), (?v, ?p, ?u)\})$ | $(?u, ?p, ?v) \in X_{\text{rdfs3+9}}\}$. // rule rdfs3+rdfs9
        PAIRS ← PAIRS ∪ $\{((?u,$ rdf: type, $o_h), \{(s_h,$ rdfs: subClassOf, $o_h), (?p,$ rdfs: range, $s_h), (?q,$ rdfs: subPropertyOf, $?p), (?v, ?q, ?u)\})$ | $(?u,$ rdf: type, $o_h) \in R, (?p,$ rdfs: range, $s_h) \in$ rdfs-closure($H$), $(?q,$ rdfs: subPropertyOf, $?p) \in$ rdfs-closure($H$), $(?v, ?q, ?u) \in R, (?u, ?p, ?v) \notin X_{\text{rdfs3+9}}\}$. // rule rdfs7+rdfs3+rdfs9
    end if
end for
return REPLACE-DERIVED-TRIPLE(PAIRS, $D \cup \{t\}$). (See Algorithm 11).
```

IS-CONSISTENT$_{inc}$ first calls FIND-MIN-JUST$_{inst}$ (Algorithm 6) in Line 2 to find all minimal justifications for t in D_4, but there are no minimal justifications for t in D_4.

Algorithm 14 FIND-STRIPLE-JUST-PAIR$_{\text{schema}}$(t, D) finds all pairs of a schema triple d in data D and a minimal justification J for d in $((D \cup \{t\}) \setminus \{d\})$ such that $t \in J$, where t is a schema triple not in D.

```
Input: $D$, data; $t$, schema triple not in $D$.
Output: PAIRS, set of pairs of a schema triple $d$ in $D$ and a minimal justification $J$ for $d$ in $((D \cup \{t\}) \setminus \{d\})$ such that $t \in J$.
PAIRS $\leftarrow \emptyset$.
$s_h \leftarrow$ subject of $t$; $p_h \leftarrow$ property of $t$; $o_h \leftarrow$ object of $t$.
$H \leftarrow$ schema data of $D$; $R \leftarrow$ instance data of $D$.
if $p_h$ = rdf: type and $o_h$ = rdf: Property then
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((s_h$, rdfs: subPropertyOf, $s_h)$, $\{t\}\big) \mid (s_h$, rdfs: subPropertyOf, $s_h) \in H\}$. // rule rdfs6
else if $p_h$ = rdf: type and $o_h$ = rdfs: Class then
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((s_h$, rdfs: subClassOf, $s_h)$, $\{t\}\big) \mid (s_h$, rdfs: subClassOf, $s_h) \in H\}$. // rule rdfs10
else if $p_h$ = rdfs: subPropertyOf and $s_h \neq o_h$ then
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((s_h$, rdfs: subPropertyOf, $?p)$, $\{t, (o_h$, rdfs: subPropertyOf, $?p)\}\big) \mid (s_h$, rdfs: subPropertyOf, $?p) \in H$, $(o_h$, rdfs: subPropertyOf, $?p) \in$ rdfs-closure($H$), $?p \neq o_h\}$. // rule rdfs5
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((?p$, rdfs: subPropertyOf, $o_h)$, $\{(?p$, rdfs: subPropertyOf, $s_h)$, $t\}\big) \mid (?p$, rdfs: subPropertyOf, $o_h) \in H$, $(?p$, rdfs: subPropertyOf, $s_h) \in$ rdfs-closure($H$), $?p \neq s_h\}$. // rule rdfs5
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((?p$, rdfs: subPropertyOf, $?q)$, $\{(?p$, rdfs: subPropertyOf, $s_h)$, $t$, $(o_h$, rdfs: subPropertyOf, $?q)\}\big) \mid (?p$, rdfs: subPropertyOf, $?q) \in H$, $(?p$, rdfs: subPropertyOf, $s_h) \in$ rdfs-closure($H$), $(o_h$, rdfs: subPropertyOf, $?q) \in$ rdfs-closure($H$), $?p \neq s_h$, $?q \neq o_h\}$. // rule rdfs5+rdfs5
else if $p_h$ = rdfs: subClassOf and $s_h \neq o_h$ then
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((s_h$, rdfs: subClassOf, $?x)$, $\{t, (o_h$, rdfs: subClassOf, $?x)\}\big) \mid (s_h$, rdfs: subClassOf, $?x) \in H$, $(o_h$, rdfs: subClassOf, $?x) \in$ rdfs-closure($H$), $?x \neq o_h\}$. // rule rdfs11
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((?x$, rdfs: subClassOf, $o_h)$, $\{(?x$, rdfs: subClassOf, $s_h)$, $t\}\big) \mid (?x$, rdfs: subClassOf, $o_h) \in H$, $(?x$, rdfs: subClassOf, $s_h) \in$ rdfs-closure($H$), $?x \neq s_h\}$. // rule rdfs11
    PAIRS $\leftarrow$ PAIRS $\cup \{\big((?x$, rdfs: subClassOf, $?y)$, $\{(?x$, rdfs: subClassOf, $s_h)$, $t$, $(o_h$, rdfs: subClassOf, $?y)\}\big) \mid (?x$, rdfs: subClassOf, $?y) \in H$, $(?x$, rdfs: subClassOf, $s_h) \in$ rdfs-closure($H$), $(o_h$, rdfs: subClassOf, $?y) \in$ rdfs-closure($H$), $?x \neq s_h$, $?y \neq o_h\}$. // rule rdfs11+rdfs11
end if
return REPLACE-DERIVED-TRIPLE(PAIRS, $D \cup \{t\}$). (See Algorithm 11).
```

IS-CONSISTENT$_{inc}$ then calls FIND-TRIPLE-JUST-PAIR$_{inst}$ (Algorithm 10) in Line 10 to find all pairs of an RDF triple d in D_4 and a minimal justification J for d in $((D_4 \cup \{t\}) \setminus \{d\})$ such that $t \in J$. FIND-TRIPLE-JUST-PAIR$_{inst}$ first finds all pairs of an RDF triple d in D_4 and a minimal justification J for d in $((D_4 \cup$ rdfs-closure(H_4) $\cup \{t\}) \setminus \{d\})$ such that $t \in J$, where rdfs-closure(H_4) is the RDFS closure of the schema data H_4 of D_4 and is shown in Table 4.2c. The pair $\big($(u, r, v), $\{$(p, rdfs: subPropertyOf, r), $t\}\big)$ is found in Line 7 because (u, r, v) is in D_4, and (q, rdfs: subPropertyOf, r)

(p, rdf: type, rdf: Property)
(q, rdf: type, rdf: Property)
(r, rdf: type, rdf: Property)
(p, rdfs: subPropertyOf, q)
(q, rdfs: subPropertyOf, r)

(a) Schema data of D_4, H_4.

(u, r, v)

(b) Instance data of D_4, R_4.

(p, rdf: type, rdf: Property)
(q, rdf: type, rdf: Property)
(r, rdf: type, rdf: Property)
(p, rdfs: subPropertyOf, q)
(q, rdfs: subPropertyOf, r)
(p, rdfs: subPropertyOf, r)
(p, rdfs: subPropertyOf, p)
(q, rdfs: subPropertyOf, q)
(r, rdfs: subPropertyOf, r)

(c) RDFS closure of schema data H_4, rdfs-closure(H_4).

Table 4.2: Data D_4 used to illustrate the operation of IS-CONSISTENT$_{\text{inc}}$ (Algorithm 9), where p, q, and r are properties. u and v are individuals.

is in rdfs-closure(H_4). FIND-TRIPLE-JUST-PAIR$_{inst}$ then calls REPLACE-DERIVED-TRIPLE (Algorithm 11) in Line 17 to replace the derived schema triple (p, rdfs: subPropertyOf, r) in the pair with a minimal justification for it. The pair becomes $\big($(u, r, v), $\{$(p, rdfs: subPropertyOf, q), (q, rdfs: subPropertyOf, r), $t\}\big)$.

Finally, IS-CONSISTENT$_{inc}$ checks if the probability of the inconsistent truth value assignment created from the pair is zero in Line 16. If so, the theory with the addition of t and its probability information is still consistent.

Example 11 (Consistency Checking After Adding a Schema Triple)**.** *This example illustrates the operation of IS-CONSISTENT$_{inc}$ (Algorithm 9) using a consistent pRDFS theory with data D_5 (Table 4.3). IS-CONSISTENT$_{inc}$ is used to check the consistency of the theory after the schema triple t = (p, rdfs: subPropertyOf, q) and its probability information are added to the theory, where p and q are properties.*

IS-CONSISTENT$_{inc}$ first calls FIND-MIN-JUST$_{schema}$ (Algorithm 2) in Line 4 to find all minimal justifications for t in D_5, but there are no minimal justifications for t in D_5.

IS-CONSISTENT$_{inc}$ then calls FIND-ITRIPLE-JUST-PAIR$_{schema}$ (Algorithms 12 and

(p, rdf: type, rdf: Property)
(q, rdf: type, rdf: Property)
(r, rdf: type, rdf: Property)
(s, rdf: type, rdf: Property)
(p, rdfs: subPropertyOf, s)
(q, rdfs: subPropertyOf, r)
(q, rdfs: subPropertyOf, s)

(a) Schema data of D_5, H_5.

(u, p, v)
(u, q, v)
(u, r, v)

(b) Instance data of D_5, R_5.

(p, rdf: type, rdf: Property)
(q, rdf: type, rdf: Property)
(r, rdf: type, rdf: Property)
(s, rdf: type, rdf: Property)
(p, rdfs: subPropertyOf, s)
(q, rdfs: subPropertyOf, r)
(q, rdfs: subPropertyOf, s)
(p, rdfs: subPropertyOf, p)
(q, rdfs: subPropertyOf, q)
(r, rdfs: subPropertyOf, r)
(s, rdfs: subPropertyOf, s)

(c) RDFS closure of schema data H_5, rdfs-closure(H_5).

(p, rdf: type, rdf: Property)	(p, rdfs: subPropertyOf, p)
(q, rdf: type, rdf: Property)	(q, rdfs: subPropertyOf, q)
(r, rdf: type, rdf: Property)	(r, rdfs: subPropertyOf, r)
(s, rdf: type, rdf: Property)	(s, rdfs: subPropertyOf, s)
(p, rdfs: subPropertyOf, s)	(q, rdfs: subPropertyOf, r)
(p, rdfs: subPropertyOf, q)	(q, rdfs: subPropertyOf, s)
(p, rdfs: subPropertyOf, r)	

(d) RDFS closure of schema data H_5 and the schema triple (p, rdfs: subPropertyOf, q), rdfs-closure($H_5 \cup \{$(p, rdfs: subPropertyOf, q)$\}$).

Table 4.3: Data D_5 used to illustrate the operation of IS-CONSISTENT$_{\text{inc}}$ (Algorithm 9), where p, q, r, and s are properties. u and v are individuals.

13) in Line 12 to find all pairs of an instance triple d in D_5 and a minimal justification J for d in $((D_5 \cup \{t\}) \setminus \{d\})$ such that $t \in J$. FIND-ITRIPLE-JUST-PAIR$_{schema}$ first finds all pairs of an instance triple d in D_5 and a minimal justification J for d in $((D_5 \cup$ rdfs-closure$(H_5 \cup \{t\})\}) \setminus \{d\})$ such that J contains t or any derived schema triple in (rdfs-closure($H_5 \cup \{t\}$) $\setminus$ rdfs-closure(H_5)), where H_5 is the schema data of D_5. The RDFS closures of the schema data, rdfs-closure(H_5) and rdfs-closure($H_5 \cup \{t\}$) are shown in Tables 4.3c and 4.3d respectively. There is a derived schema triple (p, rdfs: subPropertyOf, r) in (rdfs-closure($H_5 \cup \{t\}$) $\setminus$ rdfs-closure(H_5)). Therefore, two schema triples are considered in the for loop (Lines 3-34). They are t and (p, rdfs: subPropertyOf, r). For t, FIND-ITRIPLE-JUST-PAIR$_{schema}$ finds a pair $\big($(u, q, v), $\{$(u,

p, v), $t$$\}\big)$ in Line 19 because both RDF triples (u, q, v) and (u, p, v) are in D_5. For (p, rdfs: subPropertyOf, r), FIND-ITRIPLE-JUST-PAIR$_{schema}$ finds a pair $\big($(u, r, v), $\{$(u, p, v), (p, rdfs: subPropertyOf, r)$\}\big)$ in Line 19 because both RDF triples (u, r, v) and (u, p, v) are in D_5. FIND-ITRIPLE-JUST-PAIR$_{schema}$ finally calls REPLACE-DERIVED-TRIPLE (Algorithm 11) in Line 35 to replace the derived schema triple (p, rdfs: subPropertyOf, r) in the pair with a minimal justification for it. The pairs returned by FIND-ITRIPLE-JUST-PAIR$_{schema}$ are $\big($(u, q, v), $\{$(u, p, v), $t$$\}\big)$ and $\big($(u, r, v), $\{$(u, p, v), t, (q, rdfs: subPropertyOf, r)$\}\big)$.

IS-CONSISTENT$_{inc}$ then calls FIND-STRIPLE-JUST-PAIR$_{schema}$ (Algorithm 14) in Line 13 to find all pairs of a schema triple d in D_5 and a minimal justification J for d in $((D_5 \cup \{t\}) \setminus \{d\})$ such that $t \in J$. FIND-STRIPLE-JUST-PAIR$_{schema}$ finds a pair $\big($(p, rdfs: subPropertyOf, s), $\{$$t$, ($q$, rdfs: subPropertyOf, s)$\}\big)$ in Line 9 because both schema triples (p, rdfs: subPropertyOf, s) and (q, rdfs: subPropertyOf, s) are in H_5.

To sum up, the pairs found are $\big($(u, q, v), $\{$(u, p, v), $t$$\}\big)$, $\big($(u, r, v), $\{$(u, p, v), t, (q, rdfs: subPropertyOf, r)$\}\big)$, and $\big($(p, rdfs: subPropertyOf, s), $\{$$t$, ($q$, rdfs: subPropertyOf, s)$\}\big)$. IS-CONSISTENT$_{inc}$ checks if the probabilities of the inconsistent truth value assignments created from these pairs are zero in Line 16. If they are all zero, the theory with the addition of t and its probability information is still consistent.

4.3 Time Complexity

This section examines the time complexities of the consistency checking algorithms IS-CONSISTENT (Algorithm 7) and IS-CONSISTENT$_{\text{inc}}$ (Algorithm 9) with respect to the data size $|D|$.

IS-CONSISTENT (Algorithm 7) consists of two steps. One is finding all minimal justifications for each of the RDF triples that are not certainly true. The time complexities of finding all minimal justifications for an instance triple and a schema triple are O(log$|D|$) and O(1) respectively. See Section 3.3.4 for the derivation of these complexities. Assume that the number of RDF triples that are not certainly true increases linearly with $|D|$. The time complexity of this step is O($|D|$log$|D|$).

The other step is computing the probabilities of the inconsistent truth value assignments created from the minimal justifications. We assume that the probability distributions of the pRDFS theory are represented by Bayesian networks (Section 2.4) with N_{node} nodes and N_{node} does not depend on $|D|$. The nodes of the networks are the random variables for the uncertain RDF triples. We use probabilistic logic sampling (Section 2.4.2) and the same set of random samples to compute the probabilities of all inconsistent truth value assignments. Assume that the running time of generating a random value for a node is constant. The worst-case running time of probability calculation is proportional to $N_{\text{sampl}} \times N_{\text{bn}} \times N_{\text{node}}$, where N_{sampl}, N_{bn}, and N_{node} are the numbers of samples, Bayesian networks, and nodes per Bayesian network respectively. The minimum value of N_{sampl} is a constant given the parameters of the approximation error ϵ and δ. See (2.2) for the computation of N_{sampl}. $N_{\text{bn}} = \frac{p|D|}{N_{\text{node}}}$, where p is the percentage of uncertain data. Hence, the time complexity of this step is O($|D|$). The overall time complexity of IS-CONSISTENT is O($|D|$log$|D|$ + $|D|$) = O($|D|$log$|D|$).

To estimate the average-case running time of probability calculation, assume that the pRDFS theory is consistent or near consistent. The running time of probability calculation is proportional to $N_{\text{sampl}} \times N_{\text{bn(e)}} \times N_{\text{node}}$, where $N_{\text{bn(e)}}$ is the number of Bayesian networks containing entailed triples. Assume that the uncertain entailed triples are randomly distributed to the Bayesian networks. The expected value of $N_{\text{bn(e)}}$ denoted by $E(N_{\text{bn(e)}})$ is calculated as (4.9) [61].

$$E(N_{\text{bn(e)}}) = \frac{p|D|}{N_{\text{node}}}(1 - \prod_{i=1}^{q|D|} \frac{p|D| - N_{\text{node}} - i + 1}{p|D| - i + 1}). \tag{4.9}$$

where p and q are the percentages of uncertain RDF triples and uncertain entailed triples respectively.

IS-CONSISTENT$_{\text{inc}}$ (Algorithm 9) performs consistency checking each time an RDF triple t and its probability information are added to a pRDFS theory. It consists of two parts. In the first part (Lines 1-8), an inconsistent truth value assignment is constructed from t and a minimal justifications for t. The time complexities of finding all minimal justifications for t are O(log$|D|$) and O(1) for t being an instance triple and a schema triple respectively. See Section 3.3.4 for the derivation of these complexities. The running time of calculating the probabilities of all inconsistent truth value assignments is proportional to $N_{\text{sampl}} \times N_{\text{node}}$ since we perform inference on one Bayesian network which contains t. If some RDF triples in the minimal justifications for t are not in the same Bayesian network as t, then the theory is inconsistent. It is because RDF triples in different Bayesian networks are assumed to be independent, but t and the minimal justifications for t are related. The time complexity of probability calculation is O(1). The overall time complexities of this part are O(log$|D|$) and O(1) for t being an instance triple and a schema triple respectively.

In the second part of IS-CONSISTENT$_{\text{inc}}$ (Lines 9-17), an inconsistent truth value assignment is constructed from an RDF triple d and a minimal justification J for d such that J contains t. In the same way as finding minimal justifications in Section 3.3.4, We assume that the size of schema data $|H|$ is fixed and the size of instance data $|R|$ grows with $|D|$. We build two indexes for finding substitutions for the variables in the inference rules using the binary search algorithm. One index is sorted by the subjects, properties, and objects of RDF triples. The other is sorted by the properties and objects of RDF triples. If t is an instance triple, we search for all substitutions for the variables in at most three schema triples in the premises and one instance triple in the conclusion of each rule. There are at most nine rules to apply. See FIND-TRIPLE-JUST-PAIR$_{\text{inst}}$ (Algorithm 10). Hence, the time complexity of finding all pairs (d, J) is O(log$|D|$).

If t is a schema triple and d is an instance triple, we search for all substitutions for the variables in at most two schema triples and one instance triple in the premises and one instance triple in the conclusion of each rule. There are at most five rules to apply. See FIND-ITRIPLE-JUST-PAIR$_{\text{schema}}$ (Algorithms 12 and 13). Assume that the number of substitutions resulting from one of the instance triples increases linearly with $|D|$. The time complexity of finding all pairs (d, J) is O($|D|$log$|D|$).

If both t and d are schema triples, we work with schema data only to find all pairs (d, J). See FIND-STRIPLE-JUST-PAIR$_{\text{schema}}$ (Algorithm 14). Therefore, the time complexity is O(1).

Assume that the number of pairs (d, J) increases linearly with $|D|$. The worst-case running time of calculating the probabilities of all inconsistent truth value assignments created from the pairs is proportional to $N_{\text{sampl}} \times N_{\text{bn}} \times N_{\text{node}}$. $N_{\text{bn}} = \frac{p|D|}{N_{\text{node}}}$, where p

Finding all minimal justifications	O($\|D\|\log\|D\|$)
Calculating probabilities	O($\|D\|$)
Overall	O($\|D\|\log\|D\|$)

Table 4.4: Time complexity of the consistency checking algorithm IS-CONSISTENT (Algorithm 7).

		t is an instance triple	t is a schema triple
First part	Finding all minimal justifications	O($\log\|D\|$)	O(1)
	Calculating probabilities	O(1)	O(1)
Second part	Finding all (triple, minimal justification) pairs	O($\log\|D\|$)	O($\|D\|\log\|D\|$)
	Calculating probabilities	O($\|D\|$)	O($\|D\|$)
Overall		O($\|D\|$)	O($\|D\|\log\|D\|$)

Table 4.5: Time complexity of the incremental consistency checking algorithm IS-CONSISTENT$_{\text{inc}}$ (Algorithm 9).

is the percentage of uncertain data. The time complexity of probability calculation is O($|D|$). The overall time complexities of the second part of IS-CONSISTENT$_{\text{inc}}$ are O($|D|$) and O($|D|\log|D|$) for t being an instance triple and a schema triple respectively. Combining the time complexities of the both parts, the overall time complexities of IS-CONSISTENT$_{\text{inc}}$ are the same as the complexities of the second part.

Tables 4.4 and 4.5 summarize the time complexities of the consistency checking algorithms IS-CONSISTENT (Algorithm 7) and IS-CONSISTENT$_{\text{inc}}$ (Algorithm 9) respectively.

Chapter 5

pRDFS Query Evaluation

This chapter extends SPARQL queries to pRDFS theories. It defines the answer to an extended SPARQL query on a pRDFS theory and describes how to compute it. This chapter is based on [57] and [58].

5.1 Extended SPARQL Query

RDF triples (Definition 1) and triple patterns (Definition 3) are extended by including their truth values.

Definition 13 (Extended RDF Triple)**.** *An extended RDF triple is an RDF triple with the addition of the truth value of the RDF triple. It is a member of* $\mathbb{U} \times \mathbb{P} \times (\mathbb{U} \cup \mathbb{L}) \times \{\boldsymbol{T}, \boldsymbol{F}\}$*, where* $\mathbb{U}$*,* $\mathbb{P}$*, and* $\mathbb{L}$ *are sets of URI references, properties, and literals respectively.*

Example 12 (Extended RDF Triple)**.** *(Tom, worksFor, departmentOfComputing,* ***T****) is an extended RDF triple saying that the RDF triple (Tom, worksFor, departmentOfCom-*

```
SELECT DISTINCT ?department
WHERE {
  ?person rdf:type Professor true .
  ?person worksFor ?department true}
```

(a) Syntax.

$Q_{\text{ext}}(V_{\text{sel}}, G_{\text{ext}}, \text{isDistinct})$, where
V_{sel} = {?department},
G_{ext} = {(?person, rdf:type, Professor, true),
(?person, worksFor, ?department, true)},
isDistinct = true.

(b) Symbol.

Table 5.1: Example of an extended SPARQL query.

puting) is true.

Definition 14 (Extended Triple Pattern)**.** *An extended triple pattern is a triple pattern with the addition of the truth value of the triple pattern. It is a member of* $(\mathbb{U} \cup V) \times (\mathbb{P} \cup V) \times (\mathbb{U} \cup \mathbb{L} \cup V) \times \{\boldsymbol{T}, \boldsymbol{F}\}$*, where* $\mathbb{U}$*,* $\mathbb{P}$*,* $\mathbb{L}$*, and* V *are sets of URI references, properties, literals, and variables respectively.*

Definition 15 (Extended Graph Pattern)**.** *An extended graph pattern denoted by* G_{ext} *is a set of extended triple patterns.*

SPARQL queries are extended such that the patterns of the queries are specified using extended graph patterns.

Definition 16 (Extended SPARQL Query)**.** *We denote by* $Q_{ext}(V_{sel}, G_{ext}, isDistinct)$ *an extended SPARQL query with a syntax "SELECT (DISTINCT)* V_{sel} *WHERE* G_{ext}*", where* G_{ext} *is an extended graph pattern,* V_{sel} *is a subset of query variables in* G_{ext}*, and isDistinct is a Boolean variable with a domain* $\{\boldsymbol{T}, \boldsymbol{F}\}$*. The DISTINCT keyword is optional, and the variable isDistinct assigned* ***T*** *and* ***F*** *indicates the presence and absence of the DISTINCT keyword respectively.*

Example 13 (Extended SPARQL Query)**.** *An extended SPARQL query is shown in Table 5.1, where* G_{ext} *is an extended graph pattern having two extended triple patterns.*

It asks for the departments that have at least one professor and their probabilities.

5.2 Answer to an Extended SPARQL Query

A pRDFS theory can be viewed as a probability distribution over instances of the theory, where an instance of a pRDFS theory is defined as follows.

Definition 17 (Instance of a pRDFS Theory)**.** *Given a pRDFS theory (D, $\mathcal{P}$, θ), an instance of the theory denoted by D_{ext} is a set of extended RDF triples,* $\{(s,\ p,\ o,\ \tau((s,p,o)))\ \mid\ (s,p,o) \in D\}$*, where τ is a truth value assignment for data D. The instance is consistent iff τ is consistent.*

The RDFS closure of an instance of a pRDFS theory is the union of the RDFS closure of RDF triples that take true values and the set of RDF triples that take false values since the RDFS semantics are defined for RDF triples that take true values.

Definition 18 (RDFS Closure of an Instance of a pRDFS Theory)**.** *The RDFS closure of an instance of a pRDFS theory D_{ext} denoted by rdfs-closure(D_{ext}) is a set of extended RDF triples,* $\{(s,p,o,\boldsymbol{T}) \mid (s,p,o) \in \textit{rdfs-closure}(\{(s',p',o') \mid (s',p',o',\boldsymbol{T}) \in D_{ext}\})\} \cup \{(s,p,o,\boldsymbol{F}) \in D_{ext}\}$.

An answer to an extended SPARQL query $Q_{\text{ext}}(V_{\text{sel}}, G_{\text{ext}}, \text{isDistinct})$ on an instance of a pRDFS theory D_{ext} is a sequence of solutions, and a solution is the restriction of a solution to the extended graph pattern G_{ext} on D_{ext} to the set of query variables V_{sel}. The answer does not contain duplicate solutions if the variable isDistinct is true. A solution to G_{ext} on D_{ext} denoted by μ is a partial function from V to $(\mathbb{U} \cup \mathbb{L})$ such that rdfs-closure$(D_{\text{ext}}) \supseteq \mu(G_{\text{ext}})$ (with RDFS reasoning) or $D_{\text{ext}} \supseteq \mu(G_{\text{ext}})$ (without RDFS reasoning), where V, $\mathbb{U}$, and $\mathbb{L}$ are sets of query variables, URI references and literals respectively. $\mu(G_{\text{ext}})$ is the set of extended RDF triples obtained by replacing every variable v in G_{ext} with $\mu(v)$.

An answer to an extended SPARQL query $Q_{\text{ext}}(V_{\text{sel}}, G_{\text{ext}}, \text{isDistinct})$ on a pRDFS theory (D, $\mathcal{P}$, θ) is a sequence of pairs. The first element of a pair is a solution to the extended SPARQL query on an instance of the theory, and the second element is the probability of the solution. If the variable isDistinct is true, the answer does not contain pairs with the same first element.

5.3 Evaluation of an Extended SPARQL Query

An evaluation of an extended SPARQL query $Q_{\text{ext}}(V_{\text{sel}}, G_{\text{ext}}, \text{isDistinct})$ on a pRDFS theory (D, $\mathcal{P}$, θ) consists of five steps.

1. Find the set of all solutions to the graph pattern G on data D, where G is obtained by ignoring the truth values of the extended graph pattern G_{ext}. G = $\{(s, p, o) \mid (s, p, o, t) \in G_{\text{ext}}\}$. In SPARQL algebra, the set of solutions is BGP(G). It covers all solutions to all instances of the theory.

2. Convert the set of solutions BGP(G) into a sequence of solutions M^{ToList}, and compute the matched data of each solution in M^{ToList}. In SPARQL algebra, M^{ToList} = ToList(BGP(G)). μ_i^{ToList} denotes the ith solution in M^{ToList}. The set of the matched data of μ_i^{ToList} denoted by $\Gamma_{\mu_i^{\text{ToList}}}$ is $\{\mu_i^{\text{ToList}}(G_{\text{ext}})\}$, where the matched data $\mu_i^{\text{ToList}}(G_{\text{ext}})$ is a set of extended RDF triples obtained by replacing

every variable v in G_{ext} with $\mu_i^{\text{ToList}}(v)$.

3. Convert the sequence of solutions M^{ToList} into a sequence of solutions M^{Project}, where the ith solution in M^{Project} is the restriction of the ith solution in M^{ToList} to the set of query variables V_{sel}. In SPARQL algebra, M^{Project} = Project(M^{ToList}, V_{sel}). Compute the matched data of each solution in M^{Project}. μ_i^{Project} denotes the ith solution in M^{Project}. The set of the matched data of μ_i^{Project} denoted by $\Gamma_{\mu_i^{\text{Project}}}$ is equal to $\Gamma_{\mu_i^{\text{ToList}}}$.

4. If the variable isDistinct is true, create a sequence of solutions M^{Distinct} by removing duplicate solutions from the sequence of solutions M^{Project}. In SPARQL algebra, M^{Distinct} = Distinct(M^{Project}). Otherwise, leave the sequence M^{Project} unmodified, that is, M^{Distinct} = M^{Project}. Compute the matched data of each solution in M^{Distinct}. μ_i^{Distinct} denotes the ith solution in M^{Distinct}. The set of the matched data of μ_i^{Distinct} denoted by $\Gamma_{\mu_i^{Distinct}}$ is $\{\gamma \in \Gamma_{\mu_j^{Project}} \mid \mu_i^{Distinct} = \mu_j^{Project}\}$, the cardinality of which depends on the number of duplicate solutions.

5. Compute the probabilities of the solutions in M^{Distinct}. The probability of the ith solution μ_i^{Distinct} is equal to the probability of its set of alternative matched data $\Gamma_{\mu_i^{\text{Distinct}}}$. $Pr(\mu_i^{\text{Distinct}})$ = $Pr(\bigvee_{\gamma \in \Gamma_{\mu_i^{\text{Distinct}}}} \gamma)$.

The answer to the extended SPARQL query is a sequence of pairs, where the ith pair is $(\mu_i^{\text{Distinct}}$, $Pr(\mu_i^{\text{Distinct}}))$.

The probability of the set of alternative matched data can be written as the probability of the propositional formula $\mathcal{F}$ in (5.1), where RDF triples are propositional variables with domains {true, false}.

$$\mathcal{F} = \bigvee_{\gamma \in \Gamma_{\mu_i^{\text{Distinct}}}} \Big(\bigwedge_{(s,p,o,\mathbf{T}) \in \gamma} (s,p,o) \wedge \bigwedge_{(s,p,o,\mathbf{F}) \in \gamma} \neg (s,p,o) \Big). \tag{5.1}$$

If the RDF triples occurring in $\mathcal{F}$ are all declared triples, we can specify the probability of $\mathcal{F}$ using a single probability value since a pRDFS theory fully specifies the probability distribution of all declared triples. However, it is not the case if derived triples occur in $\mathcal{F}$.

Consider a simple case of computing the probability of a formula $\mathcal{F}_1 = f$ given a pRDFS theory (D, $\mathcal{P}$, θ), where f is a derived triple. Let JUST(f, D) be the set of all justifications for f in data D and DJ(f, D) be the disjunction of all justifications for f in D, $\bigvee_{J \in \text{JUST}(f,D)} \bigwedge_{j \in J} j$. If DJ$(f, D)$ is true, f is also true by RDFS reasoning. However, if DJ(f, D) is false, f could be true or false. Therefore, the lower probability of $\mathcal{F}_1$ equals the probability of DJ(f, D), and the upper probability of $\mathcal{F}_1$ is 1.

Consider another simple case of computing the probability of a formula $\mathcal{F}_2 = \neg g$ given a pRDFS theory (D, $\mathcal{P}$, θ), where g is a derived triple. If the negation of the disjunction of all justifications for g in data D, $\neg$DJ(g, D) is false, $\neg g$ is also false by RDFS reasoning. However, if $\neg$DJ(g, D) is true, $\neg g$ could be true or false. Therefore, the lower probability of $\mathcal{F}_2$ is 0, and the upper probability of $\mathcal{F}_2$ equals the probability of $\neg$DJ(g, D).

In general, suppose that derived triples f_1, f_2, $\dots$ and negations of derived triples $\neg g_1$, $\neg g_2$, $\dots$ occur in a formula $\mathcal{F}$. DJ(x, D) denotes the disjunction of all justifications for the derived triple x in data D. The probability bounds of $\mathcal{F}$ are computed as (5.2). The lower probability of $\mathcal{F}$ denoted by $Pr_{\text{lower}}(\mathcal{F})$ is computed by substituting

DJ(f_i, D) for f_i and **T** for g_j for all i and j. The upper probability of $\mathcal{F}$ denoted by $Pr_{\text{upper}}(\mathcal{F})$ is computed by substituting DJ(g_j, D) for g_j and **T** for f_i for all i and j.

$$\begin{aligned} Pr_{\text{lower}}(\mathcal{F}) &= Pr(\mathcal{F}(f_i = \text{DJ}(f_i, D), g_j = \mathbf{T} \text{ for all } i, j)) \\ Pr_{\text{upper}}(\mathcal{F}) &= Pr(\mathcal{F}(f_i = \mathbf{T}, g_j = \text{DJ}(g_j, D) \text{ for all } i, j)) \end{aligned} \tag{5.2}$$

We now consider the case that both a derived triple and its negation occur in $\mathcal{F}$. Take a formula $\mathcal{F}_3$ = ($\neg r \wedge e$) $\vee$ ($r \wedge \neg e$) as an example, where r is a declared triple and e is a derived triple. We divide the calculation of the probability of $\mathcal{F}_3$ into two cases DJ(e, D) = true and DJ(e, D) = false by writing $Pr(\mathcal{F}_3)$ = $Pr(\mathcal{F}_3 \wedge$ DJ(e, D)) + $Pr(\mathcal{F}_3 \wedge \neg$DJ($e, D$)), where DJ($e, D$) is the disjunction of all justifications for e in data D. If DJ(e, D) is true, e is true. Hence, $Pr(\mathcal{F}_3 \wedge$ DJ(e, D)) = $Pr(\mathcal{F}_3(e = \mathbf{T}) \wedge$ DJ$(e, D))$. If DJ(e, D) is false, $Pr(\mathcal{F}_3 \wedge \neg$DJ$(e, D))$ is minimum under the following condition. For every value of r, whenever only one of the expressions $\mathcal{F}_3(e = \mathbf{T})$ and $\mathcal{F}_3(e = \mathbf{F})$ is true, the probability of the expression that is true is 0. If DJ(e, D) is false, $Pr(\mathcal{F}_3 \wedge \neg$DJ$(e, D))$ is maximum under the following condition. For every value of r, whenever only one of the expressions $\mathcal{F}_3(e = \mathbf{T})$ and $\mathcal{F}_3(e = \mathbf{F})$ is true, the probability of the expression that is false is 0. Hence, the lower and upper bounds of $Pr(\mathcal{F}_3 \wedge \neg$DJ$(e, D))$ are $Pr(\mathcal{F}_3(e = \mathbf{T}) \wedge \mathcal{F}_3(e = \mathbf{F}) \wedge \neg$DJ$(e, D))$ and $Pr((\mathcal{F}_3(e = \mathbf{T}) \vee \mathcal{F}_3(e = \mathbf{F})) \wedge \neg$DJ$(e, D))$ respectively. To sum up, the probability bounds of $\mathcal{F}_3$ are (5.3).

$$
\begin{aligned}
Pr_{\text{lower}}(\mathcal{F}_3) &= Pr(\mathcal{F}_3(e=\mathbf{T}) \wedge \text{DJ}(e,D)) + Pr(\mathcal{F}_3(e=\mathbf{T}) \wedge \mathcal{F}_3(e=\mathbf{F}) \wedge \neg\text{DJ}(e,D)) \\
Pr_{\text{upper}}(\mathcal{F}_3) &= Pr(\mathcal{F}_3(e=\mathbf{T}) \wedge \text{DJ}(e,D)) + Pr((\mathcal{F}_3(e=\mathbf{T}) \vee \mathcal{F}_3(e=\mathbf{F})) \wedge \neg\text{DJ}(e,D))
\end{aligned}
\tag{5.3}
$$

In general, suppose that derived triples $e_1, e_2, \ldots, e_{N_e}, f_1, f_2, \ldots$ and negations of derived triples $\neg e_1, \neg e_2, \ldots, \neg e_{N_e}, \neg g_1, \neg g_2, \ldots$ occur in a formula $\mathcal{F}$. We divide the calculation of $Pr(\mathcal{F})$ into 2^{N_e} cases by writing $Pr(\mathcal{F}) = \sum_{(x_1,x_2,\ldots,x_{N_e})\in A} Pr(\mathcal{F} \wedge \bigwedge_i x_i)$, where $A = \{\text{DJ}(e_1,D),\ \neg\text{DJ}(e_1,D)\} \times \{\text{DJ}(e_2,D),\ \neg\text{DJ}(e_2,D)\} \times \ldots \times \{\text{DJ}(e_{N_e},D),\ \neg\text{DJ}(e_{N_e},D)\}$ and $\text{DJ}(x,D)$ denotes the disjunction of all justifications for the derived triple x in data D. The lower and upper bounds of $Pr(\mathcal{F})$ are computed as (5.4).

$$
\begin{aligned}
Pr_{\text{lower}}(\mathcal{F}) &= \sum_{(x_1,x_2,\ldots,x_{N_e})\in A} Pr(\bigwedge_{\substack{(e_1,e_2,\ldots,e_{N_e})\in B_1(x_1)\times B_2(x_2)\times\cdots\times B_{N_e}(x_{N_e}),\\ f_i=\text{DJ}(f_i,D),g_j=\mathbf{T}\text{ for all } i,j}} \mathcal{F} \wedge \bigwedge_i x_i) \\
Pr_{\text{upper}}(\mathcal{F}) &= \sum_{(x_1,x_2,\ldots,x_{N_e})\in A} Pr(\bigvee_{\substack{(e_1,e_2,\ldots,e_{N_e})\in B_1(x_1)\times B_2(x_2)\times\cdots\times B_{N_e}(x_{N_e}),\\ f_i=\mathbf{T},g_j=\text{DJ}(g_j,D)\text{ for all } i,j}} \mathcal{F} \wedge \bigwedge_i x_i) \\
A =& \{\text{DJ}(e_1,D),\neg\text{DJ}(e_1,D)\} \times \{\text{DJ}(e_2,D),\neg\text{DJ}(e_2,D)\} \times \cdots \times \\
& \{\text{DJ}(e_{N_e},D),\neg\text{DJ}(e_{N_e},D)\} \\
B_i(x_i) &= \begin{cases} \{\mathbf{T}\} & \text{if } x_i = \text{DJ}(e_i,D) \\ \{\mathbf{T},\mathbf{F}\} & \text{if } x_i = \neg\text{DJ}(e_i,D) \end{cases}
\end{aligned}
\tag{5.4}
$$

From (5.2) and (5.4), the computation of the probability of a formula $\mathcal{F}$ in which derived triples occur involves the computation of the disjunction of all justifications for the derived triples. We prove that the disjunction of all justifications for an RDF triple is equivalent to the disjunction of all minimal justifications for the RDF triple, so it can be replaced by the latter, simpler expression.

Proposition 5. *The disjunction of all justifications for an RDF triple* d *in data* D*,* $\bigvee_{J \in JUST(d,D)} \bigwedge_{j \in J} j$*, is equivalent to the disjunction of all minimal justifications for* d *in* D*,* $\bigvee_{J \in JUST_{min}(d,D)} \bigwedge_{j \in J} j$*, where JUST(d, D) is the set of all justifications for* d *in* D *and JUST*$_{min}$*(d, D) is the set of all minimal justifications for* d *in* D*.*

Proof.

$$\bigvee_{J \in JUST(d,D)} \bigwedge_{j \in J} j \tag{5.5}$$

$$\equiv \bigvee_{J \in JUST_{\text{min}}(d,D)} \bigwedge_{j \in J} j \vee \bigvee_{J \in (JUST(d,D) \setminus JUST_{\text{min}}(d,D))} \bigwedge_{j \in J} j \tag{5.6}$$

$$\equiv \bigvee_{J \in JUST_{\text{min}}(d,D)} \bigwedge_{j \in J} j \tag{5.7}$$

The term $\bigvee_{J \in (JUST(d,D) \setminus JUST_{\text{min}}(d,D))} \bigwedge_{j \in J} j$ in (5.6) disappears because for each $J \in \big(\text{JUST}(d, D) \setminus \text{JUST}_{\text{min}}(d, D)\big)$, $\exists J' \in \text{JUST}_{\text{min}}(d, D)$ such that $J \models_{rdfs} J'$. If $\bigwedge_{j \in J} j$ is true, then $\bigwedge_{j \in J'} j$ is true. Hence, $\bigwedge_{j \in J} j \vee \bigwedge_{j \in J'} j \equiv \bigwedge_{j \in J'} j$. □

Example 14 (Evaluation of an Extended SPARQL Query)**.** *This example illustrates the evaluation of an extended SPARQL query by executing the extended SPARQL query* $Q_{ext}(V_{sel}, G_{ext}, \mathit{isDistinct})$ *in Table 5.1 against the pRDFS theory* (D_6, $\mathcal{P}_6$, θ_6) *in Ta-*

Symbol	RDF triple
d_1	(Professor, rdf: type, rdfs: Class)
d_2	(Department, rdf: type, rdfs: Class)
d_3	(worksFor, rdf: type, rdf: Property)
d_4	(headOf, rdf: type, rdf: Property)
d_5	(headOf, rdfs: subPropertyOf, worksFor)
d_6	(Tom, rdf: type, Professor)
d_7	(May, rdf: type, Professor)
d_8	(departmentOfComputing, rdf: type, Department)
d_9	(Tom, headOf, departmentOfComputing)
d_{10}	(May, worksFor, departmentOfComputing)

(a) Set of RDF triples, D_6.

Probability distribution
$P_{\{d_1\}}$
$P_{\{d_2\}}$
$P_{\{d_3\}}$
$P_{\{d_4\}}$
$P_{\{d_5\}}$
$P_{\{d_6\}}$
$P_{\{d_7\}}$
$P_{\{d_8\}}$
$P_{\{d_9\}}$
$P_{\{d_{10}\}}$

(b) Set of probability distributions, $\mathcal{P}_6$.

d_i	Probability
T	1
F	0

(c) Probability distribution over d_i, $P_{\{d_i\}}$, where i = 1, 2, ..., 8.

d_9	Probability
T	0.8
F	0.2

(d) Probability distribution over d_9, $P_{\{d_9\}}$.

d_{10}	Probability
T	0.7
F	0.3

(e) Probability distribution over d_{10}, $P_{\{d_{10}\}}$.

x	$\theta(x)$
d_1	$P_{\{d_1\}}$
d_2	$P_{\{d_2\}}$
d_3	$P_{\{d_3\}}$
d_4	$P_{\{d_4\}}$
d_5	$P_{\{d_5\}}$
d_6	$P_{\{d_6\}}$
d_7	$P_{\{d_7\}}$
d_8	$P_{\{d_8\}}$
d_9	$P_{\{d_9\}}$
d_{10}	$P_{\{d_{10}\}}$

(f) Function θ_6 mapping an RDF triple to it probability distribution.

Table 5.2: pRDFS theory (D_6, $\mathcal{P}_6$, θ_6) used to illustrate the evaluation of an extended SPARQL query.

ble 5.2.

We first find the set of all solutions to the graph pattern G on D_6, where G is obtained by ignoring the truth values of the extended graph pattern G_{ext}. The set contains two solutions, which are shown in Table 5.3a.

We then convert the set of solutions into a sequence of solutions and compute the matched data of each solution by substituting the values mapped by the solution for the variables in G_{ext}. See Table 5.3b for the results.

G_{ext}	{(?person, rdf:type, Professor, **T**), (?person, worksFor, ?department, **T**)}
G	{(?person, rdf:type, Professor), (?person, worksFor, ?department)}
BGP(G)	{{(?person, Tom), (?department, departmentOfComputing)}, {(?person, May), (?department, departmentOfComputing)}}

(a) Step 1: compute BGP(G).

M^{ToList}	$(\mu_1^{\text{ToList}}, \mu_2^{\text{ToList}})$
μ_1^{ToList}	{(?person, Tom), (?department, departmentOfComputing)}
μ_2^{ToList}	{(?person, May), (?department, departmentOfComputing)}
$\Gamma_{\mu_1^{\text{ToList}}}$	{{(Tom, rdf:type, Professor, **T**), (Tom, worksFor, departmentOfComputing, **T**)}}
$\Gamma_{\mu_2^{\text{ToList}}}$	{{(May, rdf:type, Professor, **T**), (May, worksFor, departmentOfComputing, **T**)}}

(b) Step 2: compute M^{ToList} and $\Gamma_{\mu_i^{\text{ToList}}}$ for all i.

M^{Project}	$(\mu_1^{\text{Project}}, \mu_2^{\text{Project}})$
μ_1^{Project}	{(?department, departmentOfComputing)}
μ_2^{Project}	{(?department, departmentOfComputing)}
$\Gamma_{\mu_1^{\text{Project}}}$	{{(Tom, rdf:type, Professor, **T**), (Tom, worksFor, departmentOfComputing, **T**)}}
$\Gamma_{\mu_2^{\text{Project}}}$	{{(May, rdf:type, Professor, **T**), (May, worksFor, departmentOfComputing, **T**)}}

(c) Step 3: compute M^{Project} and $\Gamma_{\mu_i^{\text{Project}}}$ for all i.

M^{Distinct}	$(\mu_1^{\text{Distinct}})$
μ_1^{Distinct}	{(?department, departmentOfComputing)}
$\Gamma_{\mu_1^{\text{Distinct}}}$	{{(Tom, rdf:type, Professor, **T**), (Tom, worksFor, departmentOfComputing, **T**)}, {(May, rdf:type, Professor, **T**), (May, worksFor, departmentOfComputing, **T**)}}

(d) Step 4: compute M^{Distinct} and $\Gamma_{\mu_i^{\text{Distinct}}}$ for all i.

$\mathcal{F}$	$\big($(Tom, rdf:type, Professor) $\wedge$ (Tom, worksFor, departmentOfComputing)$\big)$ $\vee$ $\big($(May, rdf:type, Professor) $\wedge$ (May, worksFor, departmentOfComputing)$\big)$
$Pr_{\text{lower}}(\mathcal{F})$	$Pr\Big(\big($(Tom, rdf:type, Professor) $\wedge$ (headOf, rdfs: subPropertyOf, worksFor) $\wedge$ (Tom, headOf, departmentOfComputing)$\big)$ $\vee$ $\big($(May, rdf:type, Professor) $\wedge$ (May, worksFor, departmentOfComputing)$\big)\Big)$ = 0.94
$Pr_{\text{upper}}(\mathcal{F})$	$Pr\Big(\big($(Tom, rdf:type, Professor) $\wedge$ **T**$\big)$ $\vee$ $\big($(May, rdf:type, Professor) $\wedge$ (May, worksFor, departmentOfComputing)$\big)\Big)$ = 1

(e) Step 5: compute the probability of μ_1^{Distinct}, $Pr(\mathcal{F})$.

Table 5.3: Intermediate results of the evaluation of the extended SPARQL query in Table 5.1 on the pRDFS theory (D_6, $\mathcal{P}_6$, θ_6) in Table 5.2.

We then compute the restriction of each solution to the set of query variables V_{sel}, which is {?department}, and the two solutions become the same. The matched data of each solution remain the same. The results are shown in Table 5.3c.

We then remove duplicate solutions from the sequence of solutions since the variable isDistinct is true. One duplicate solution is removed from the sequence, and there *is one solution in the sequence. The solution now has two alternative matched data, one of which comes from the duplicate solution. See Table 5.3d for the results.*

The probability of the solution is equal to the probability of the two alternative matched data, which is written as the probability of the propositional formula $\mathcal{F}$ shown in Table 5.3e. A derived triple (Tom, worksFor, departmentOfComputing) occurs in $\mathcal{F}$, so the probability of $\mathcal{F}$ cannot be specified using a single value. There is only one justification for (Tom, worksFor, departmentOfComputing) in D_6, which is $\{$(headOf, rdfs: subPropertyOf, worksFor), (Tom, headOf, departmentOfComputing)$\}$. We use (5.2) to compute the the lower and upper probabilities of $\mathcal{F}$. See Table 5.3e for the results.

Finally, the answer to the extended SPARQL query is a sequence of one pair, and the pair is ($\{$(?department, departmentOfComputing)$\}$, [0.94, 1]).

5.4 Running Time Analysis

This section analyses the running time of the query evaluation. The time complexity of finding all solutions to the graph pattern of a query is the same as in the non-probabilistic case. It is O($|G_{\text{ext}}||D|$) [49] without RDFS reasoning, where $|G_{\text{ext}}|$ is the size of the graph pattern and $|D|$ is the data size.

For the computation of the probabilities of solutions, we assume that the probability distributions of a pRDFS theory are represented by Bayesian networks (Section 2.4) with N_{node} nodes and N_{node} does not depend on $|D|$. The nodes of the networks are the random variables for the uncertain RDF triples. We use probabilistic logic sampling

(Section 2.4.2) and the same set of random samples to compute the probabilities of all solutions.

Without RDFS reasoning, the running time of probability calculation is proportional to $N_{\text{sampl}} \times N_{\text{bn(m)}} \times N_{\text{node}}$, where N_{sampl}, $N_{\text{bn(m)}}$, and N_{node} are the number of samples, the number of Bayesian networks that contain any uncertain RDF triple in the matched data of a solution, and the number of nodes per Bayesian network respectively. The minimum value of N_{sampl} is a constant given the parameters of the approximation error ϵ and δ. See (2.2) for the computation of N_{sampl}. Assume that the percentage of uncertain data in the matched data is the same as the percentage of uncertain data in the data and the uncertain RDF triples in the matched data are randomly drawn from the data. The number of uncertain RDF triples in the matched data of all solutions is $p|G_{\text{ext}}|N_{\text{sol}}$, where p is the percentage of uncertain data and N_{sol} is the number of solutions. The expected value of $N_{\text{bn(m)}}$ denoted by $E(N_{\text{bn(m)}})$ is computed as (5.8) [61].

$$E(N_{\text{bn(m)}}) = \frac{p|D|}{N_{\text{node}}}(1 - \prod_{i=1}^{p|G_{\text{ext}}|N_{\text{sol}}} \frac{p|D| - N_{\text{node}} - i + 1}{p|D| - i + 1}). \tag{5.8}$$

If the number of Bayesian networks $\frac{p|D|}{N_{\text{node}}}$ is much larger than the number of uncertain RDF triples in the matched data of all solutions $p|G_{\text{ext}}|N_{\text{sol}}$, $E(N_{\text{bn(m)}})$ is close to $p|G_{\text{ext}}|N_{\text{sol}}$. With this approximation, the time complexity of probability calculation is O($|G_{\text{ext}}|$).

For the query evaluation with RDFS reasoning, the matched data of a solution to a query may contain both the declared and derived data. Let X be the set of uncertain declared triples in the matched data of all solutions and the uncertain RDF triples in the minimal justifications for the uncertain derived triples in the matched data of all solutions to the query. The running time of probability calculation is proportional to $N_{\text{sampl}} \times N'_{\text{bn(m)}} \times N_{\text{node}}$, where N_{sampl}, $N'_{\text{bn(m)}}$, and N_{node} are the number of samples, the number of Bayesian networks that contain any one of the RDF triples in X, and the number of nodes per Bayesian network respectively.

To develop an expression for the expected value of $N'_{\text{bn(m)}}$, assume that the percentages of uncertain RDF triples and derived triples in the matched data are the same as those in the data. The number of uncertain declared triples in the matched data of a solution is $(1-q)p|G_{\text{ext}}|$, where p and q are the percentages of uncertain data and derived data respectively. The number of derived triples in the matched data of a solution is $q|G_{\text{ext}}|$. In any justification for an derived instance triple, there is only one instance triple and all other RDF triples are schema triples. Assume all schema triples are certain since they are usually definitional in nature. A derived instance triple is uncertain only if the instance triple in every justification for the derived triple is uncertain. If the instance triples in the justifications are randomly drawn from the data, the number of uncertain derived triples in the matched data of a solution is $qp^{N_j}|G_{\text{ext}}|$, where N_j is the average number of minimal justifications for a derived triple. Hence, $|X|$ is $N_{sol}((1-q)p|G_{\text{ext}}| + qp^{N_j}|G_{\text{ext}}|N_j)$. The expected value of $N'_{\text{bn(m)}}$ denoted by $E(N'_{\text{bn(m)}})$ is computed as (5.9).

$$E(N'_{\text{bn(m)}}) = \mathcal{A}(\frac{p|D|}{N_{\text{node}}}, N_{\text{sol}}|X|). \tag{5.9}$$

where $\frac{p|D|}{N_{\text{node}}}$ is the number of Bayesian networks and $\mathcal{A}(x, y) = x(1 - (\frac{x-1}{x})^y)$ is the expected number of distinct items by picking y items randomly from x distinct items with replacement.

Chapter 6

Experimental Study

This chapter presents an experimental evaluation of the running time performance of the consistency checking algorithms in Chapter 4 and the query evaluation algorithm in Chapter 5 with respect to the data size, the percentage of uncertain data, the size of correlated data (by varying the number of nodes in a Bayesian network), and the complexity of the probability distributions (by varying the degree of nodes in a network).

6.1 Environment

The experiments were run on a computer with an Intel Core i7-2670QM processor and 4 GB memory. We wrote the software code in Java and used the RDF toolkit Jena [13], which includes an RDF/XML parser, reasoners, and a SPARQL query engine.

Parameter	Setting
data size, $\|D\|$	250k, **500k**, 750k, 1M
percentage of uncertain data, p	25%, **50%**, 75%, 100%
number of nodes, N_{node}	5, **10**, 15, 20
in- and out-degree of nodes, N_{ideg} and N_{odeg}	1, **2**, 3, 4

Table 6.1: Data settings. The default values are in bold.

6.2 Data

Data from the Berlin SPARQL Benchmark (BSBM) [10] and the Lehigh University Benchmark (LUBM) [30] were used. BSBM contains data about products, offers by vendors, and reviews by consumers. LUBM contains data about universities as well as the people and activities within them. Data generators of both benchmarks can output data of different sizes. The minimum units of data generation for BSBM and LUBM are products and universities respectively. The data sizes chosen in the experiments are shown in the first row of Table 6.1, and the default value is in bold.

6.2.1 Uncertain Data Generation

The data in both BSBM and LUBM are certain. Uncertain data are generated as follows. We assume all schema triples are certain since they are usually definitional in nature. We randomly select a p percentage of the instance triples to be uncertain. There is an extra rule to this random selection process. If an instance triple t is picked to be uncertain, one RDF triple from each justification for t is also randomly selected to be uncertain. The reason is that if all RDF triples in a justification for t are certain, t has to be certain. After all uncertain triples have been selected, we randomly divide them into groups of size N_{node}. However, RDF triples related through the RDFS se-

Symbol	RDF triple
d_1	(Professor, rdf: type, rdfs: Class)
d_2	(Department, rdf: type, rdfs: Class)
d_3	(worksFor, rdf: type, rdf: Property)
d_4	(headOf, rdf: type, rdf: Property)
d_5	(headOf, rdfs: subPropertyOf, worksFor)
d_6	(Tom, rdf: type, Professor)
d_7	(departmentOfComputing, rdf: type, Department)
d_8	(Tom, headOf, departmentOfComputing)
d_9	(Tom, worksFor, departmentOfComputing)

Table 6.2: Data D_7 used to illustrate the generation of uncertain data.

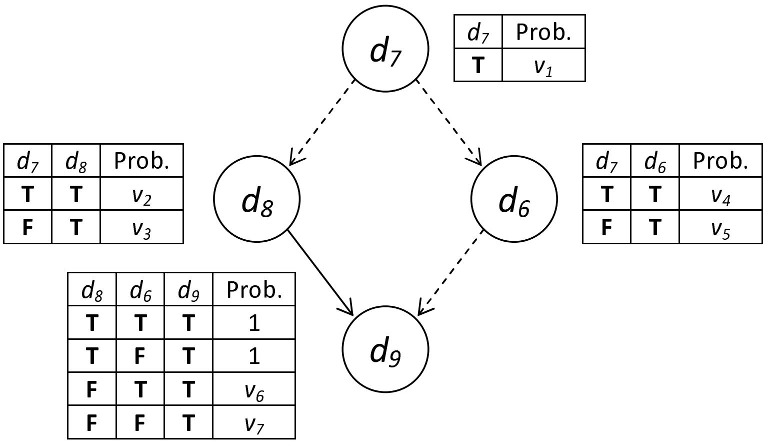
mantics are forced to be in the same group. Therefore, the sizes of some groups may be greater than N_{node}. RDF triples in the same group are statistically correlated and are independent of the RDF triples in other groups.

The probability distribution of the uncertain RDF triples in each group is modeled using the Bayesian Network (Section 2.4). Each node of the network represents an uncertain RDF triple in the group with a domain {true, false}. Each uncertain RDF triple in the justifications for an RDF triple t has a direct influence on t, and there is a directed edge from the uncertain RDF triple to t. The conditional independence relationships among other uncertain RDF triples are generated randomly. Edges are randomly added to the network until both the average in-degree N_{ideg} and out-degree N_{odeg} of nodes equal a specified value.

The conditional probability table of each node t is generated as follows. If all RDF triples in any one of the justifications for t are true, the probability of t being true is one. Otherwise, the probability of t is generated randomly.

Table 6.1 shows the values of p, N_{node}, N_{ideg}, and N_{odeg} used to generate the uncertain data, and the default values are in bold.

Example 15 (Uncertain Data Generation)**.** *This example generates uncertain data for*



d_7	Prob.
T	v_1

d_7	d_8	Prob.
T	**T**	v_2
F	**T**	v_3

d_7	d_6	Prob.
T	**T**	v_4
F	**T**	v_5

d_8	d_6	d_9	Prob.
T	**T**	**T**	1
T	**F**	**T**	1
F	**T**	**T**	v_6
F	**F**	**T**	v_7

Figure 6.1: A randomly generated Bayesian network for the instance triples d_6 to d_9 in Table 6.2. The dashed edges and probability values v_1 to v_7 are randomly generated.

the data D_7 in Table 6.2 with the following values for the percentage of uncertain data p, the number of nodes N_{node}, and the average in-degree N_{ideg} and out-degree N_{odeg} of nodes. p = 100%, N_{node} = 4, and N_{ideg} = N_{odeg} = 1.

RDF triples d_1 to d_5 are schema triples and are assumed to be certain. Instance triples d_6 to d_9 are all uncertain since p is 100%. They are in the same group because N_{node} is four. Their probability distribution is represented by a Bayesian network. The Bayesian network has four nodes, each of which represents an instance triple. There is a directed edge from d_8 to d_9 because $\{d_5, d_8\} \models_{rdfs} d_9$. Other edges are randomly added to the network until both N_{ideg} and N_{odeg} equal one. Fig. 6.1 shows a randomly generated Bayesian network, where the dashed edges are randomly generated. Probability values v_1 to v_7 in the conditional probability tables are randomly generated. To satisfy the RDFS semantics, the probability of d_9 being true is one whenever d_8 is true.

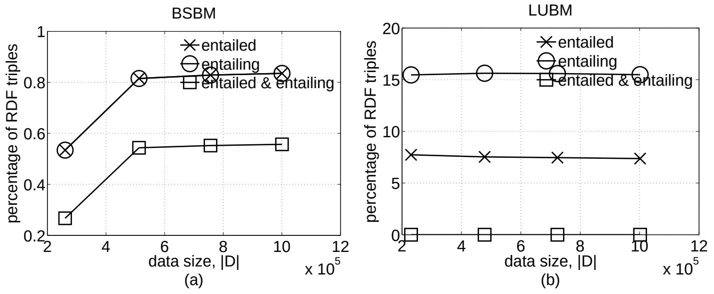


Figure 6.2: Percentages of RDF triples that are entailed, entailing, and both entailed and entailing.

6.2.2 Properties

This section examines several properties of the data, which are used later for the analysis of experimental results. These properties include the percentage of entailed triples, the percentage of entailing triples, the percentage of uncertain entailed triples, the number of minimal justifications per derived instance triple, and the percentage of derived data.

Fig. 6.2 shows the percentages of entailed, entailing, and both entailed and entailing triples under different data sizes. See Section 2.1.4 for the definitions of entailed and entailing triples. The percentage values of each category are roughly constant except the value at the smallest data size of BSBM, which is lower than the others. Note that BSBM contains RDF triples that are both entailed and entailing whereas LUBM does not have any.

Fig. 6.3 shows the percentage of uncertain entailed triples as functions of the data size and the percentage of uncertain data. This quantity is an important factor in the running time of consistency checking because uncertain entailed triples create inconsistent truth value assignments. Figs. 6.3a and 6.3b indicate that the percentage of uncertain entailed triples is roughly constant under different data sizes except the per-

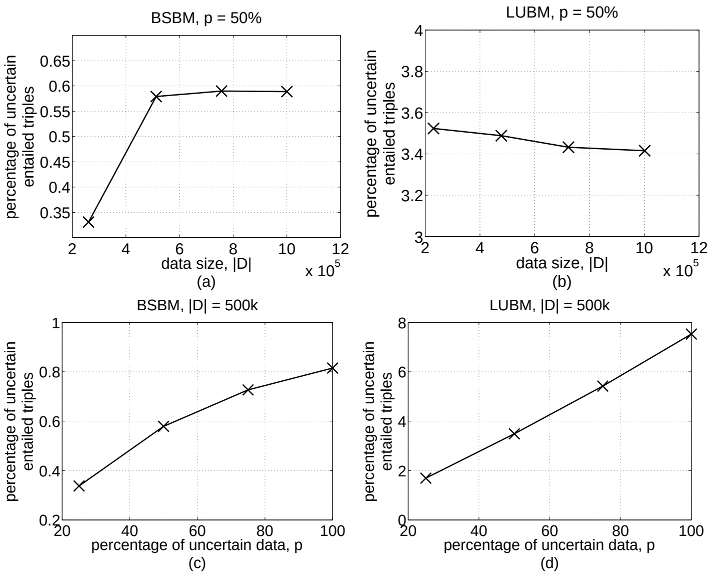


Figure 6.3: Percentage of uncertain entailed triples.

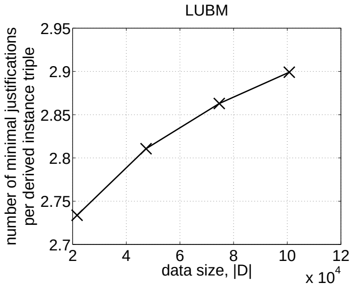


Figure 6.4: Average number of minimal justifications per derived instance triple.

centage at the smallest data size of BSBM, which is lower than the others. Fig. 6.3c shows that the percentage of uncertain entailed triples in BSBM increases faster at small percentages of uncertain data and slower at large percentages of uncertain data. Fig. 6.3d shows that the percentage of uncertain entailed triples in LUBM increases linearly with the percentage of uncertain data.

The number of minimal justifications per derived instance triple affects the amount of data involved in the probability calculation of a solution to a query if the matched data of the solution contains derived data. This quantity as a function of the data size $|D|$ is shown in Fig. 6.4. It increases with $|D|$, but the rate of increase decreases with

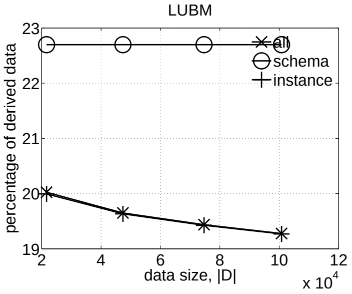


Figure 6.5: Percentage of derived data.

$|D|$.

The percentage of derived data defined as $|\text{rdfs-closure}(D) \setminus D|/|\text{rdfs-closure}(D)|$ determines the expected proportion of derived data in the matched data of a solution to a query if the matched data are randomly drawn from the RDFS closure of data D. This quantity as a function of $|D|$ is plotted in Fig. 6.5. The percentage of derived schema data does not change with $|D|$ since the schema data are fixed. The percentage of derived instance data decreases as $|D|$ increases.

6.3 Queries

Benchmark queries from BSBM and LUBM were used. BSBM has 12 queries. 10 of the queries are of the "select" form. Among these 10 queries, four of them have simple graph patterns, and the others have complex graph patterns. Complex graph patterns are formed by combining smaller patterns using SPARQL keywords OPTIONAL and UNION. The four queries with simple graph patterns were chosen in the experiments. The triple patterns in the graph patterns are changed to the extended triple patterns, and the truth values of the extended triple patterns are set to the true values. The chosen queries are shown in Table 6.3, and the namespace URIs for the prefixes used

```
SELECT DISTINCT ?product ?label
WHERE {
  ?product rdfs:label ?label true .
  ?product rdf:type %ProductType% true .
  ?product bsbm:productFeature %ProductFeature1% true .
  ?product bsbm:productFeature %ProductFeature2% true .
  ?product bsbm:productPropertyNumeric1 ?value1 true .
  FILTER (?value1 > %x%)}
ORDER BY ?label
LIMIT 10
```

(a) BSBM query 1.

```
SELECT DISTINCT ?product ?productLabel
WHERE {
  ?product rdfs:label ?productLabel true .
  FILTER (%ProductXYZ% != ?product)
  %ProductXYZ% bsbm:productFeature ?prodFeature true .
  ?product bsbm:productFeature ?prodFeature true .
  %ProductXYZ% bsbm:productPropertyNumeric1 ?origProperty1 true .
  ?product bsbm:productPropertyNumeric1 ?simProperty1 true .
  FILTER (?simProperty1 < (?origProperty1 + 120) && ?simProperty1 > (?origProperty1 − 120))
  %ProductXYZ% bsbm:productPropertyNumeric2 ?origProperty2 true .
  ?product bsbm:productPropertyNumeric2 ?simProperty2 true .
  FILTER (?simProperty2 < (?origProperty2 + 170) && ?simProperty2 > (?origProperty2 − 170))}
ORDER BY ?productLabel
LIMIT 5
```

(b) BSBM query 5.

```
SELECT ?product ?label
WHERE {
  ?product rdfs:label ?label true .
  ?product rdf:type bsbm:Product true .
  FILTER regex(?label, ”%word1%”)}
```

(c) BSBM query 6.

```
SELECT DISTINCT ?offer ?price
WHERE {
  ?offer bsbm:product %ProductXYZ% true .
  ?offer bsbm:vendor ?vendor true .
  ?offer dc:publisher ?vendor true .
  ?vendor bsbm:country <http://downlode.org/rdf/iso-3166/countries#US> true .
  ?offer bsbm:deliveryDays ?deliveryDays true .
  FILTER (?deliveryDays <= 3)
  ?offer bsbm:price ?price true .
  ?offer bsbm:validTo ?date true .
  FILTER (?date > %currentDate%)}
ORDER BY xsd:double(str(?price))
LIMIT 10
```

(d) BSBM query 10.

Table 6.3: Extended BSBM queries 1, 5, 6, and 10.

Prefixes	Namespace URIs
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
bsbm:	http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/
dc:	http://purl.org/dc/elements/1.1/
xsd:	http://www.w3.org/2001/XMLSchema#
ub:	http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#
d0u0:	http://www.Department0.University0.edu/
d1u0:	http://www.Department1.University0.edu/
d0u0ap0:	http://www.Department0.University0.edu/AssistantProfessor0/

Table 6.4: Namespace prefixes.

Reasoning	LUBM Query													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
no	4	0	6	0	0	0	0	0	0	0	0	0	0	5916
RDFS	4	0	6	34	719	6463	61	6463	134	0	0	0	0	5916
OWL-DL [45]	4	0	6	34	719	7790	67	7790	208	4	224	15	1	5916

Table 6.5: Number of solutions obtained by executing the LUBM queries with different reasoning against the LUBM data at the data size of 100k.

in the queries are shown in Table 6.4. Each query contains some variables enclosed in the % symbols. An instance of the query is generated by replacing these variables with random values from their respective domains. 10 instances are generated for each query, and the running time of answering the query is measured by taking the average of the running time of answering 10 instances of the query.

The LUBM has 14 queries, all of which are of the "select" form and have basic graph patterns. Table 6.5 shows the number of solutions obtained by executing the LUBM queries against the LUBM data at the data size of 100k. The numbers of solutions to queries 4 to 9 obtained with RDFS reasoning are more than those obtained without reasoning. This illustrates the usefulness of having reasoning during query answering.

We selected those queries the answers to which are non-empty. For query evalua-

```
SELECT ?X
WHERE {
  ?X rdf:type ub:GraduateStudent true .
  ?X ub:takesCourse d0u0:GraduateCourse0 true}
```

(a) LUBM query 1.

```
SELECT ?X
WHERE {
  ?X rdf:type ub:Publication true .
  ?X ub:publicationAuthor d0u0:AssistantProfessor0 true}
```

(b) LUBM query 3.

```
SELECT ?X ?Y1 ?Y2 ?Y3
WHERE {
  ?X rdf:type ub:Professor true .
  ?X ub:worksFor <http://www.Department0.University0.edu> true .
  ?X ub:name ?Y1 true .
  ?X ub:emailAddress ?Y2 true .
  ?X ub:telephone ?Y3 true}
```

(c) LUBM query 4.

```
SELECT ?X
WHERE {
  ?X rdf:type ub:Person true .
  ?X ub:memberOf <http://www.Department0.University0.edu> true}
```

(d) LUBM query 5.

```
SELECT ?X
WHERE {
  ?X rdf:type ub:Student true}
```

(e) LUBM query 6.

```
SELECT ?X ?Y
WHERE {
  ?X rdf:type ub:Student true .
  ?Y rdf:type ub:Course true .
  d0u0:AssociateProfessor0 ub:teacherOf ?Y true .
  ?X ub:takesCourse ?Y true}
```

(f) LUBM query 7.

```
SELECT ?X ?Y ?Z
WHERE {
  ?X rdf:type ub:Student true .
  ?Y rdf:type ub:Department true .
  ?X ub:memberOf ?Y true .
  ?Y ub:subOrganizationOf <http://www.University0.edu> true .
  ?X ub:emailAddress ?Z true}
```

(g) LUBM query 8.

Table 6.6: Extended LUBM queries 1, 3, 4, 5, 6, 7, and 8.

tion without reasoning, the LUBM queries 1, 3, 14 were selected. For query evaluation with RDFS reasoning, the LUBM queries 1, 3-9, and 14 were selected. Like the BSBM queries, the triple patterns in the queries are changed to extended triple patterns, and

```
SELECT ?X ?Y ?Z
WHERE {
   ?X rdf:type ub:Student true .
   ?Y rdf:type ub:Faculty true .
   ?Z rdf:type ub:Course true .
   ?X ub:advisor ?Y true .
   ?X ub:takesCourse ?Z true .
   ?Y ub:teacherOf ?Z true}
```

(a) LUBM query 9.

```
SELECT ?X
WHERE {
   ?X rdf:type ub:UndergraduateStudent true}
```

(b) LUBM query 14.

Table 6.7: Extended LUBM queries 9 and 14.

LUBM Query	1	3	4	5	7
Number of solutions, N_{sol}	4	6	34	719	61

Table 6.8: Numbers of solutions to the LUBM queries 1, 3, 4, 5, and 7. They do not change with the data size $|D|$.

the truth values of the extended triple patterns are set to the true values. The selected queries are shown in Tables 6.6 and 6.7.

The running time of answering a query would increase with the number of solutions to the query. The numbers of solutions to the LUBM queries 1, 3, 4, 5, and 7 do not change with the data size $|D|$, and they are shown in Table 6.8. The numbers of solutions to the BSBM queries 1, 5, 6, and 10 and the LUBM queries 6, 8, 9, and 14 vary with $|D|$, and they are plotted in Fig. 6.6.

Table 6.9 shows the properties of the queries and their solutions that would affect the running time of query evaluation. These properties include the matched data size $|G_{\text{ext}}|$ of a solution, the percentage of derived data q in the matched data, and the average number of minimal justifications for a derived triple N_j in the matched data.

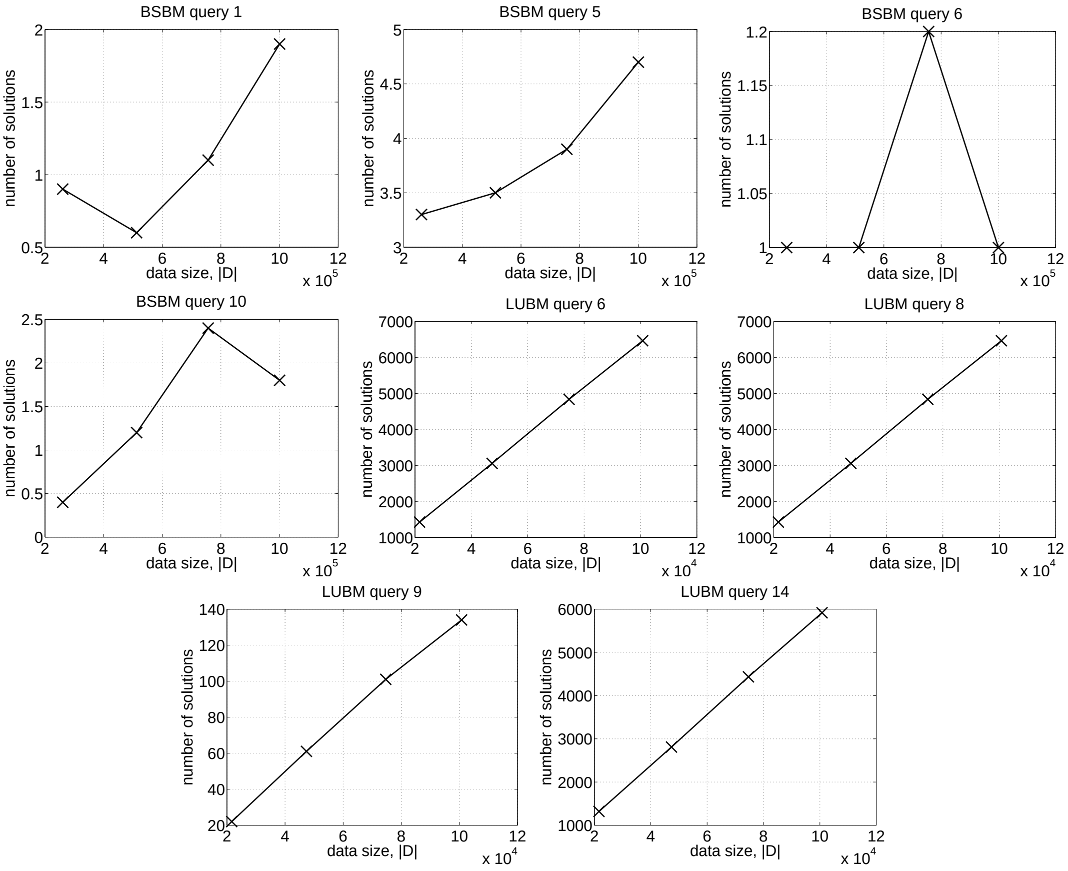


Figure 6.6: Number of solutions to the BSBM queries 1, 5, 6, and 10 and the LUBM queries 6, 8, 9, and 14.

6.4 Consistency Checking

This section examines how the running time of consistency checking scales with the data size, the percentage of uncertain data, the average number of nodes per Bayesian network, and the average in- and out-degrees of Bayesian network nodes. Consistency checking consists of two major components, finding minimal justifications and calculating the probabilities of inconsistent truth value assignments.

BSBM query	1	5	6	10
Size of matched data, $\lvert G_{\text{ext}} \rvert$	5	7	2	7
Percentage of derived data, q	0	0	0	0
Average number of minimal justifications for a derived triple, N_j	n/a	n/a	n/a	n/a

(a) BSBM queries.

LUBM query	1	3	4	5	6	7	8	9	14
Size of matched data, $\lvert G_{\text{ext}} \rvert$	2	2	5	2	1	4	5	6	1
Percentage of derived data, q	0	0	20	53	100	25.8	20	37.2	0
Average number of minimal justifications for a derived triple, N_j	n/a	n/a	8.5	3.93	1	1.02	1	5.99	n/a

(b) LUBM queries.

Table 6.9: Properties of queries and their solutions (n/a: not applicable).

6.4.1 Finding Minimal Justifications

As discussed in Section 4.3, the time complexity of finding minimal justifications is O($|D|$log$|D|$), where $|D|$ is the data size. This agrees with the experimental results shown in Figs. 6.7a and 6.7b that the running time of finding minimal justifications increases linearly with $|D|$log$|D|$.

The running time of finding minimal justifications is proportional to the percentage of RDF triples that are not certainly true, which is equal to the percentage of uncertain data in our experiments. This agrees with the experimental results shown in Figs. 6.7c and 6.7d that the running time increases linearly with the percentage of uncertain data p.

The running time of finding minimal justifications does not depend on the number of nodes per Bayesian network and the degree of nodes. The experiment results shown in Figs. 6.7e to 6.7h also show no trend in the running time with respect to the number and degree of nodes.

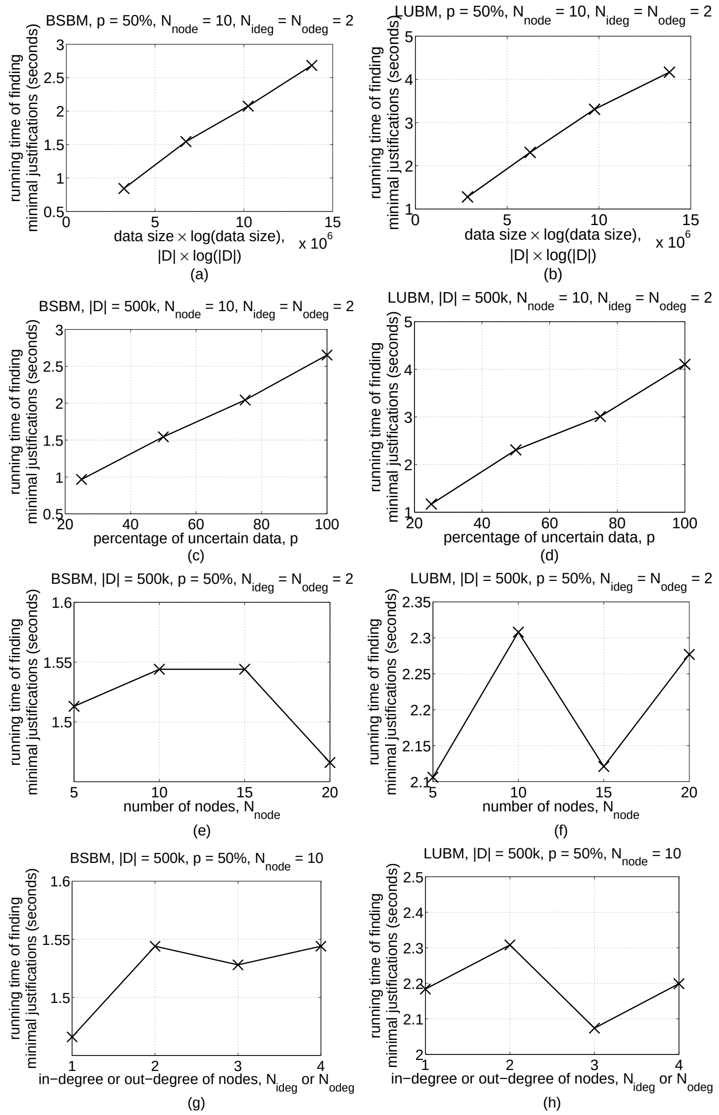


Figure 6.7: Running time of finding minimal justifications in consistency checking.

6.4.2 Calculating Probabilities

Probabilistic logic sampling (Section 2.4.2) is used to calculate the probability. The number of samples used in probabilistic logic sampling is determined by setting both ϵ and δ in (2.2) to 0.01.

As discussed in Section 4.3, the time complexity of probability calculation is O($|D|$) and the quantity $E(N_{\text{bn(e)}}) \times N_{\text{node}}$ is proportional to the expected running time of the

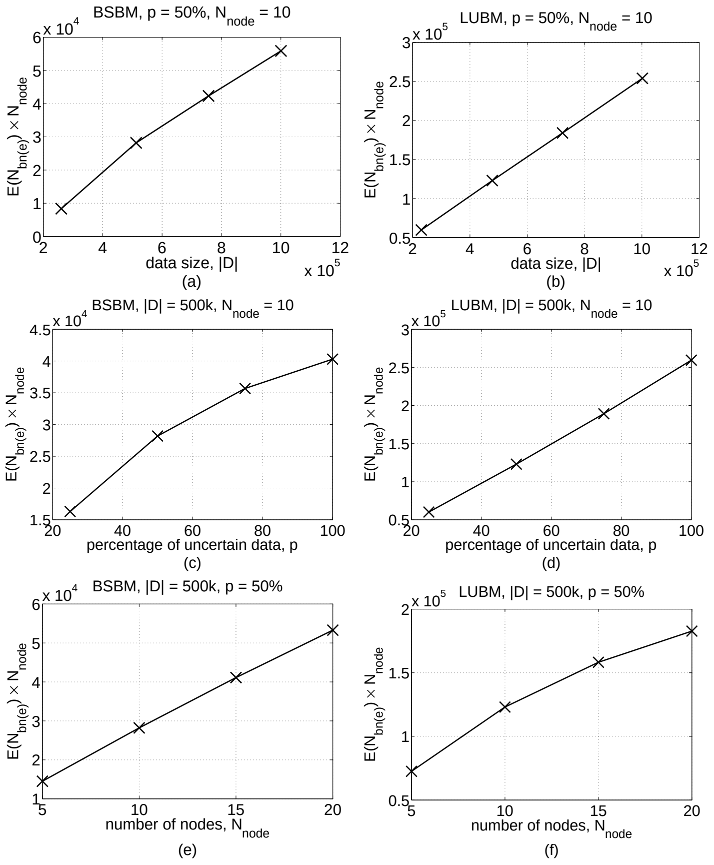


Figure 6.8: The quantity $E(N_{\text{bn(e)}}) \times N_{\text{node}}$, which is proportional to the expected running time of calculating probabilities in consistency checking.

probability calculation, where $|D|$ is the data size, $E(N_{\text{bn(e)}})$ is defined in (4.9), and N_{node} is the number of nodes. We plot $E(N_{\text{bn(e)}}) \times N_{\text{node}}$ in Fig. 6.8 and compare it with the experimental results in Fig. 6.9. The value of the percentage of uncertain entailed triples q in $E(N_{\text{bn(e)}})$ is obtained from Fig. 6.3.

Figs. 6.8a and 6.8b show that $E(N_{\text{bn(e)}}) \times N_{\text{node}}$ increases linearly with the data size. The value at the smallest data size of BSBM deviates from the linear relationship because the value of q at this data size shown in Fig. 6.3a is lower than the values of q at the other data sizes. The experimental results shown in Figs. 6.9a and 6.9b also show that the running time of probability calculation increases linearly with the data

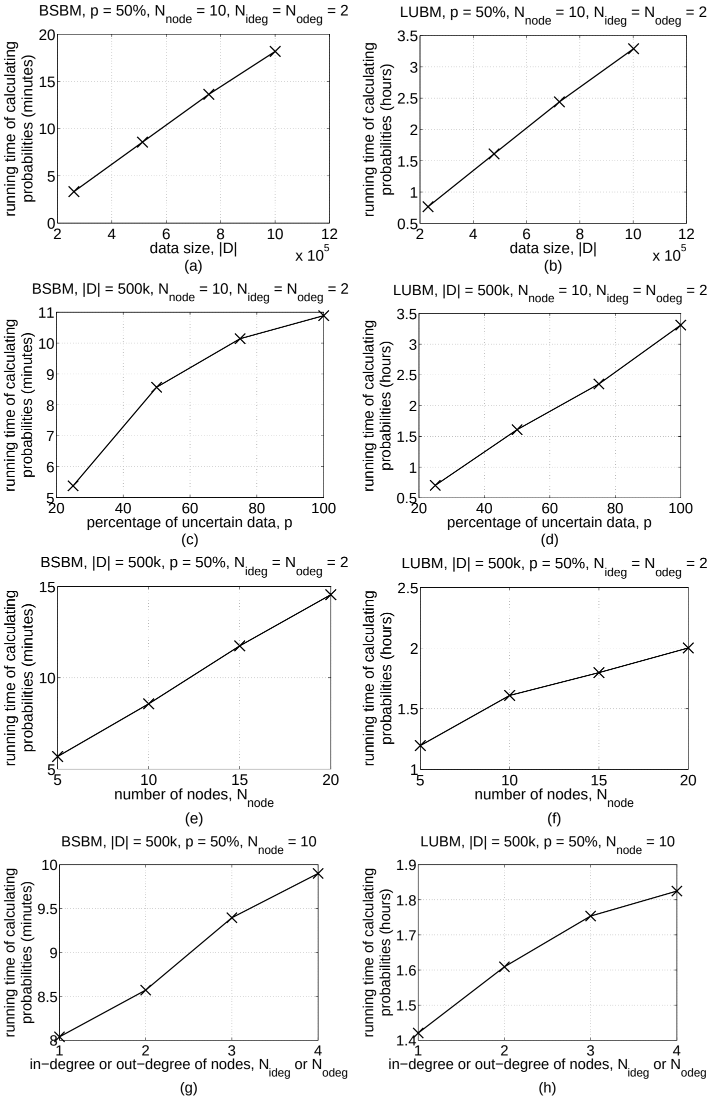


Figure 6.9: Running time of calculating probabilities in consistency checking.

size.

Figs. 6.8c and 6.8d show that $E(N_{\text{bn(e)}}) \times N_{\text{node}}$ grows sublinearly with the percentage of uncertain data p in BSBM and linearly in LUBM. These orders of growth correspond to the orders of growth of the percentage of uncertain entailed triples shown in Figs. 6.3c and 6.3d. This matches the experimental results shown in Figs. 6.9c and 6.9d that the running time of probability calculation grows sublinearly with p in BSBM and linearly in LUBM.

Figs. 6.8e and 6.8f show that $E(N_{\text{bn(e)}}) \times N_{\text{node}}$ increases linearly with the number of nodes N_{node} in BSBM and sublinearly in LUBM. This agrees with the experimental results shown in Figs. 6.9e and 6.9f that the running time of probability calculation increases linearly with N_{node} in BSBM and sublinearly in LUBM.

The running time of probability calculation is expected to increase with the in-degree N_{ideg} or out-degree N_{odeg} of nodes to handle the increased parents of nodes. Figs. 6.9e and 6.9f show that the running time of probability calculation increases linearly with N_{ideg} or N_{odeg} in BSBM and sublinearly in LUBM.

6.5 Incremental Consistency Checking

This section studies the running time performance of incremental consistency checking with respect to the data size, the percentage of uncertain data, the number of nodes, and the degree of nodes.

We measure the running time of checking the consistency of a consistent theory after an RDF triple is added to it. Recall that the operation of removing an RDF triple from a consistent theory does not void the consistency of the theory. We start with a consistent theory and remove an RDF triple t from the consistent theory. Then we add the RDF triple t back and check the consistency of the theory because of the introduction of t. We repeat this process $|D|$ times, where $|D|$ is the number of RDF triples in the theory. Each time a different RDF triple t is chosen from the theory.

We divide the running time of consistency checking into two parts, the time of finding minimal justifications and the time of calculating probabilities, which are shown

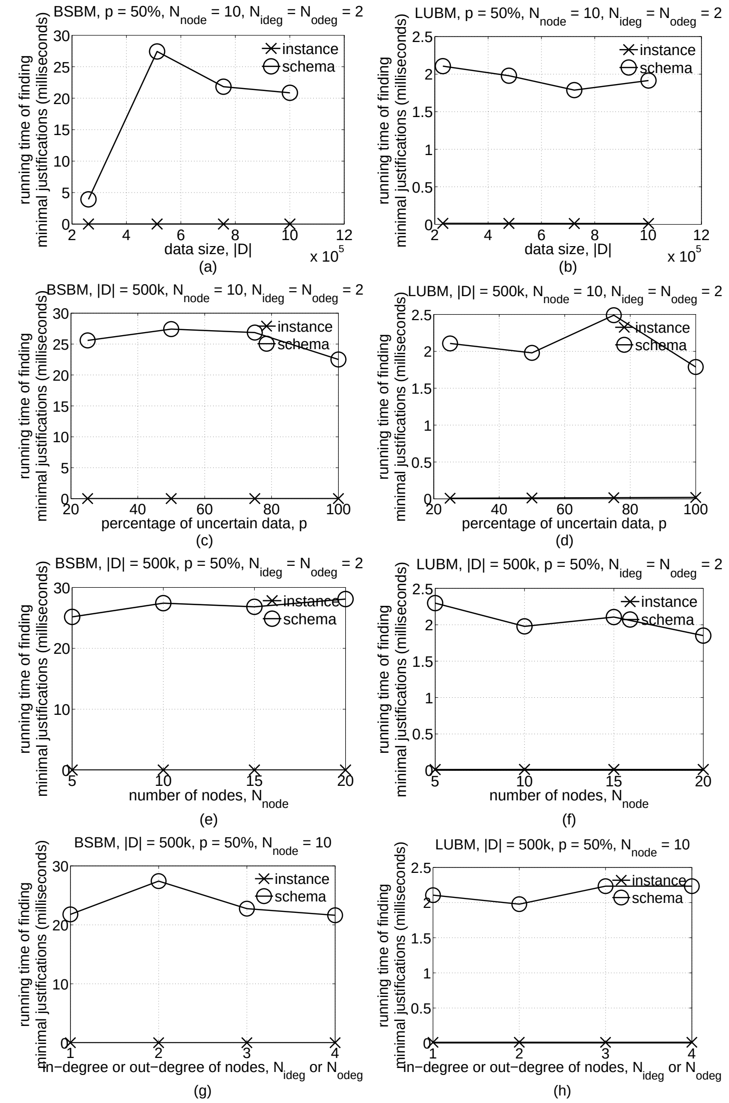


Figure 6.10: Running time of finding minimal justifications in incremental consistency checking.

in Figs. 6.10 and 6.11 respectively. Moreover, we divide the RDF triple t added to the theory into instance and schema triples.

From Figs. 6.10 and 6.11, the average consistency checking time is under 160ms in every case. From Fig. 6.10, the time of finding minimal justifications for schema triples is longer than that for instance triples. It does not change with the data size, the percentage of uncertain data, the number of nodes, and the degree of nodes.

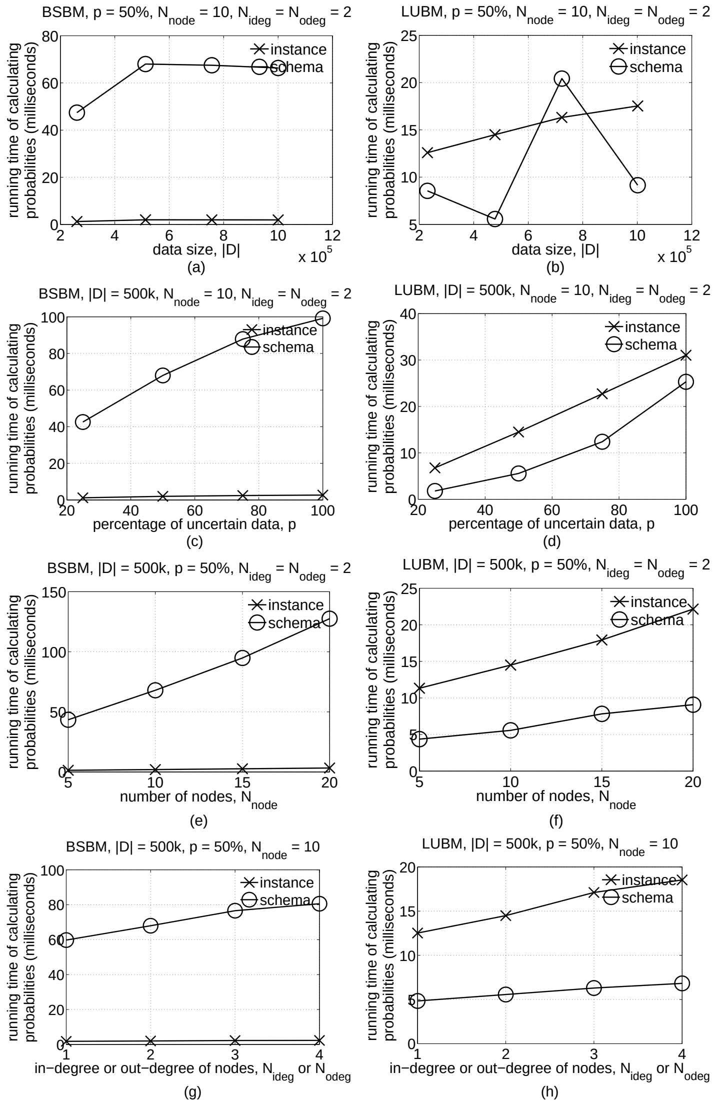


Figure 6.11: Running time of calculating probabilities in incremental consistency checking.

As discussed in Section 4.3, the time complexity of probability calculation is O($|D|$), where $|D|$ is the data size. From Figs. 6.11a and 6.11b, the running time of calculating probabilities either does not change or increases linearly with $|D|$. From Figs. 6.11c and 6.11d, the running time of calculating probabilities increases sublinearly with the percentage of uncertain data p in BSBM. In LUBM, it increases linearly with p for the instance triples. It increases exponentially with p for the schema triples because

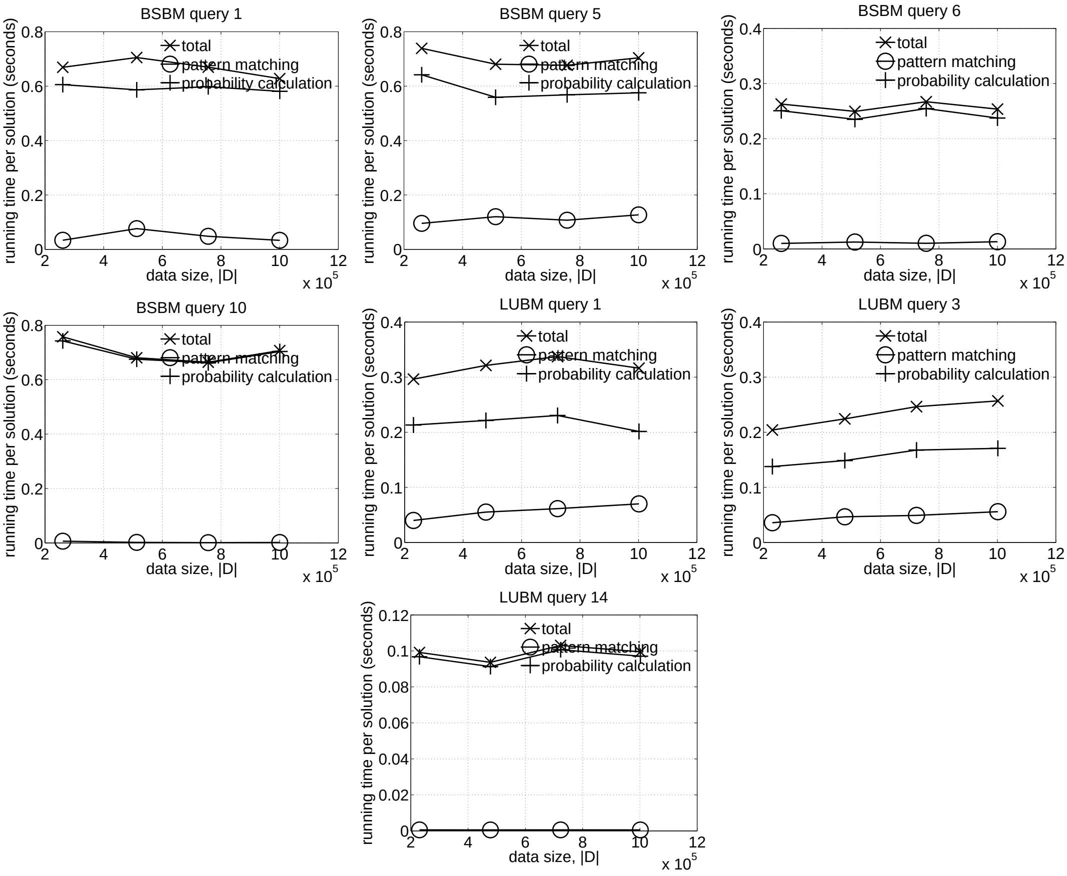


Figure 6.12: Running time of answering queries without RDFS reasoning as a function of the data size $|D|$ (p = 50%, N_{node} = 10, N_{ideg} = N_{odeg} = 2).

the number of inconsistent truth value assignments that contain the schema triples also increases exponentially with p. From Figs. 6.11e to 6.11h, the running time of calculating probabilities increases linearly with the number and the degree of nodes.

6.6 Query Evaluation without RDFS Reasoning

In this section, we examine the running time performance of query evaluation without RDFS reasoning. BSBM queries 1, 5, 6, and 10 and LUBM queries 1, 3, and 14 are selected in this set of experiments because the answers to these queries are non-empty. The number of solutions to LUBM query 14 increases rapidly with the data size. The

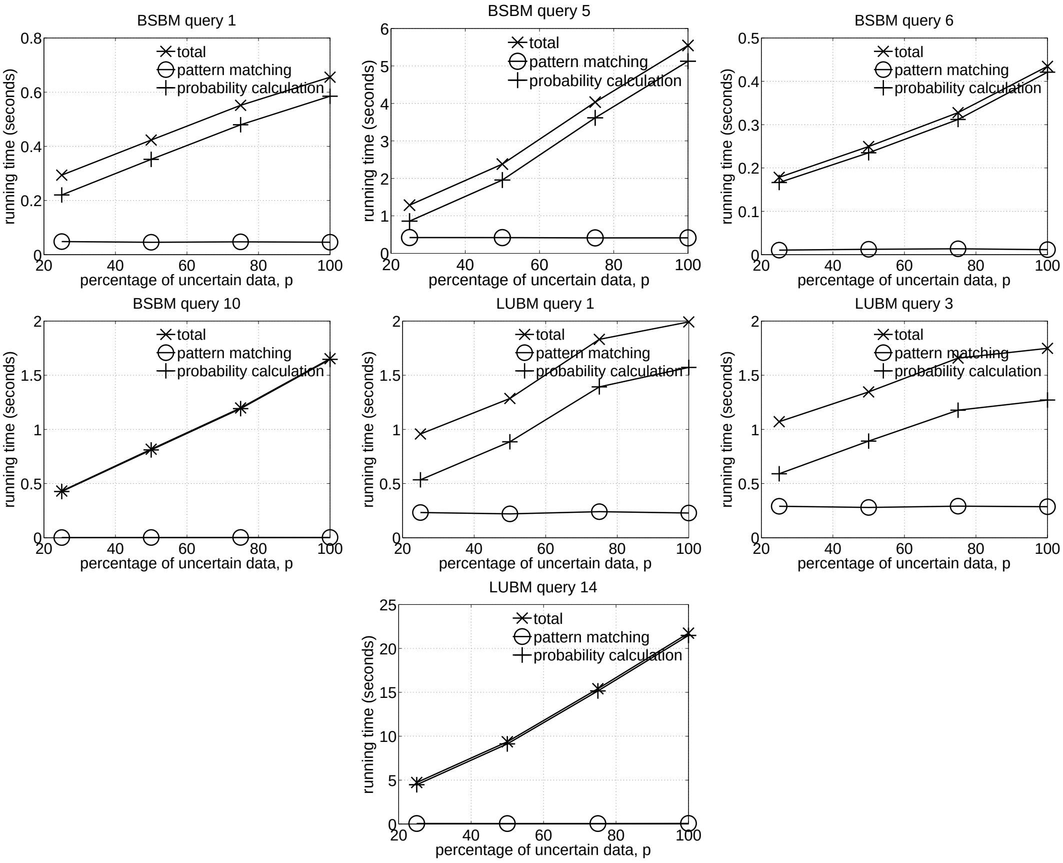


Figure 6.13: Running time of answering queries without RDFS reasoning as a function of the percentage of uncertain data p ($|D|$ = 500k, N_{node} = 10, N_{ideg} = N_{odeg} = 2).

query time at large data size is very long, so we only compute the first 100 solutions for this query.

Query evaluation consists of two major components, pattern matching and probability calculation.

6.6.1 Pattern Matching

Fig. 6.12 shows the running time of pattern matching as a function of the data size. Since the number of solutions to the BSBM queries varies with the data size, the running time per solution is plotted against the data size. The running time of pat-

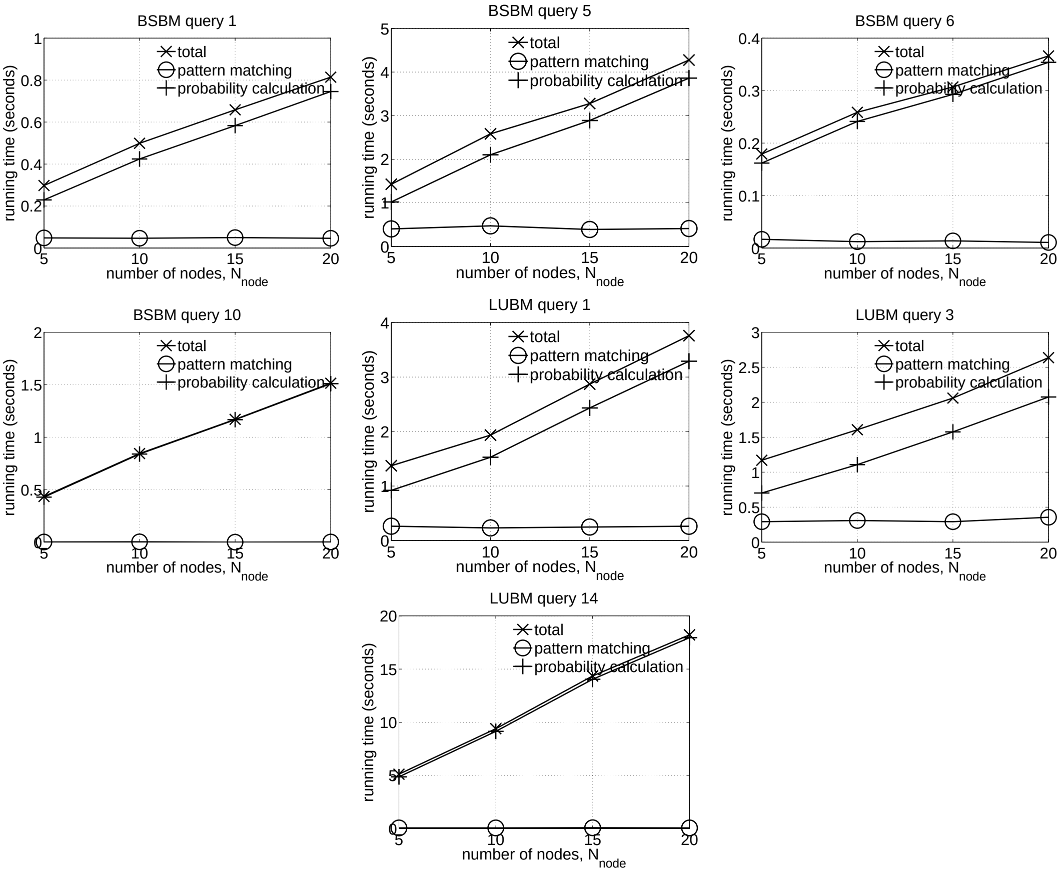


Figure 6.14: Running time of answering queries without RDFS reasoning as a function of the number of nodes N_{node} ($|D|$ = 500k, p = 50%, $N_{\text{ideg}} = N_{\text{odeg}} = 2$).

tern matching increases linearly with the data size for BSBM queries 5 and 6 and for LUBM queries 1 and 3. For the rest of the queries, there is no trend in the running time.

The running time of pattern matching does not depend on the percentage of uncertain data, the number of nodes, and the degree of nodes. This agrees with the experimental results shown in Figs. 6.13, 6.14, and 6.15 respectively.

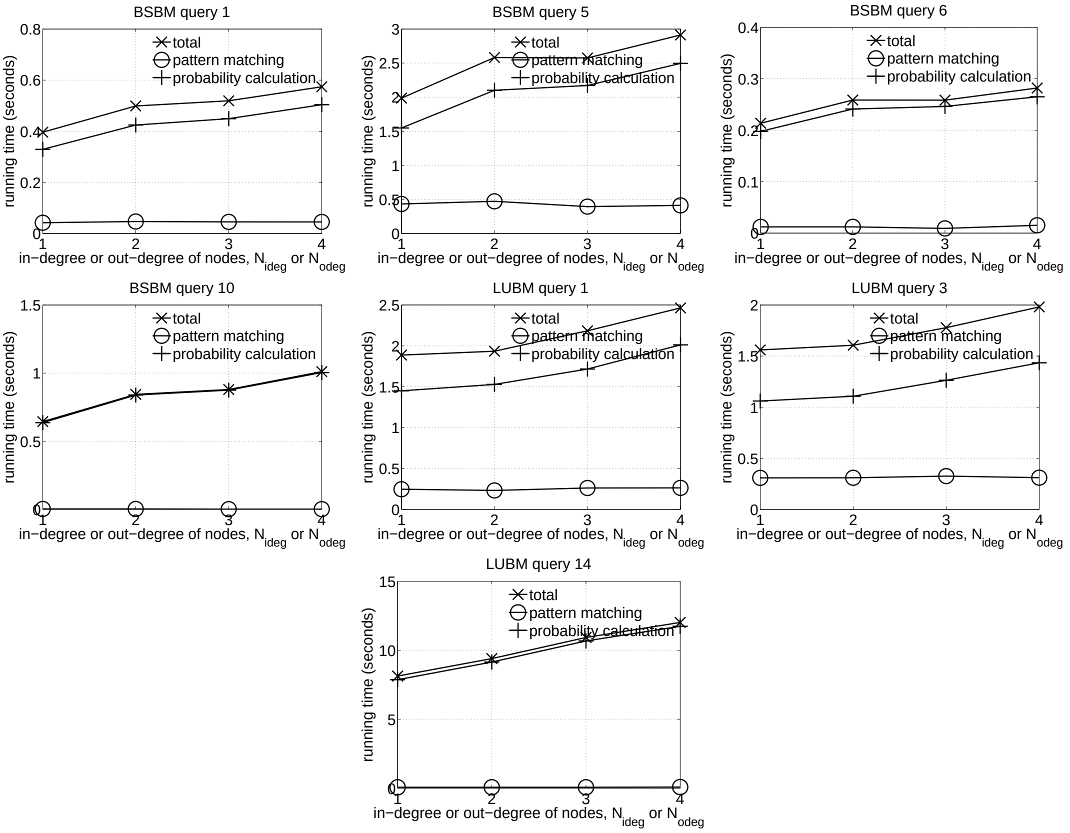


Figure 6.15: Running time of answering queries without RDFS reasoning as a function of the in-degree N_{ideg} or out-degree N_{odeg} of nodes ($|D|$ = 500k, p = 50%, N_{node} = 10).

6.6.2 Probability Calculation

Probabilistic logic sampling (Section 2.4.2) is used to calculate the probability. The number of samples used in probabilistic logic sampling is determined by setting both ϵ and δ in (2.2) to 0.01.

As discussed in Section 5.4, the time complexity of probability calculation is approximately O($|G_{\text{ext}}|$) and the quantity $E(N_{\text{bn(m)}}) \times N_{\text{node}}$ is proportional to the expected running time of the probability calculation, where $|G_{\text{ext}}|$ is the size of matched data, $E(N_{\text{bn(m)}})$ is defined in (5.8), and N_{node} is the number of nodes. We plot $E(N_{\text{bn(m)}}) \times N_{\text{node}}$ in Fig. 6.16 and compare it with the experimental results shown in Figs. 6.12, 6.13, and 6.14. We set the size of the matched data $|G_{\text{ext}}|$ and the number of solutions

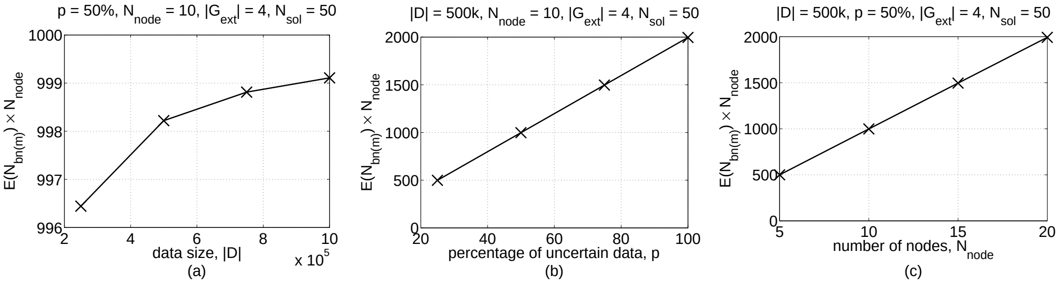


Figure 6.16: The quantity $E(N_{\text{bn(m)}}) \times N_{\text{node}}$, which is proportional to the expected running time of calculating the probabilities of the solutions to a query.

N_{sol} in $E(N_{\text{bn(m)}})$ to 4 and 50 respectively. See Table 6.9 for the sizes of matched data of the tested queries, which range between 1 and 7. See Table 6.8 and Fig. 6.6 for the number of solutions to the tested queries.

Fig. 6.16a shows that $E(N_{\text{bn(m)}}) \times N_{\text{node}}$ increases very slightly with the data size. This agrees with the experimental results shown in Fig. 6.12, which do not show any trend in the running time of probability calculation.

Fig. 6.16b shows that $E(N_{\text{bn(m)}}) \times N_{\text{node}}$ increases linearly with the percentage of uncertain data. This result matches the experimental results shown in Fig. 6.13 that the running time of probability calculation increases linearly with the percentage of uncertain data.

Fig. 6.16c shows that $E(N_{\text{bn(m)}}) \times N_{\text{node}}$ increases linearly with the number of nodes. This agrees with the experimental results shown in Fig. 6.14 that the running time of probability calculation increases linearly with the number of nodes.

The running time of probability calculation would increase with the in-degree N_{ideg} or out-degree N_{odeg} of nodes because of the processing of increased parents of nodes. Fig. 6.15 shows that the running time of probability calculation increases linearly with N_{ideg} or N_{odeg}.

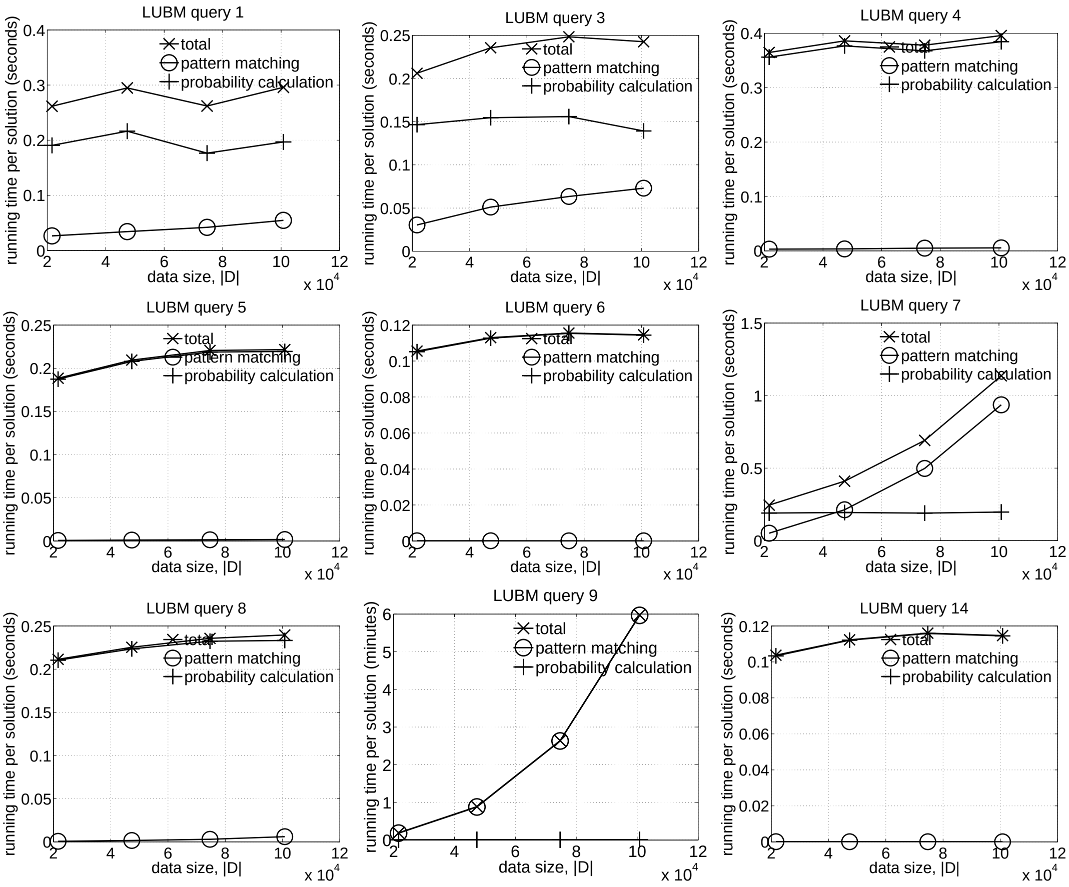


Figure 6.17: Running time of answering queries with RDFS reasoning as a function of the data size $|D|$ (p = 50%, N_{node} = 10, N_{ideg} = N_{odeg} = 2).

6.7 Query Evaluation with RDFS Reasoning

In this section, we examine the running time performance of query evaluation with RDFS reasoning. In this set of experiments, the LUBM data are used because they produce derived data. Moreover, RDFS reasoning is required for some LUBM queries to produce non-empty results. The data sizes tested are 10 times smaller than those in the previous experiments because of the long query time.

Query evaluation time as functions of the data size $|D|$, the percentage of uncertain data p, the number of nodes per Bayesian network N_{node}, and the in-degree N_{ideg} or out-degree N_{odeg} of nodes is plotted in Figs. 6.17, 6.18, 6.19, and 6.20 respectively.

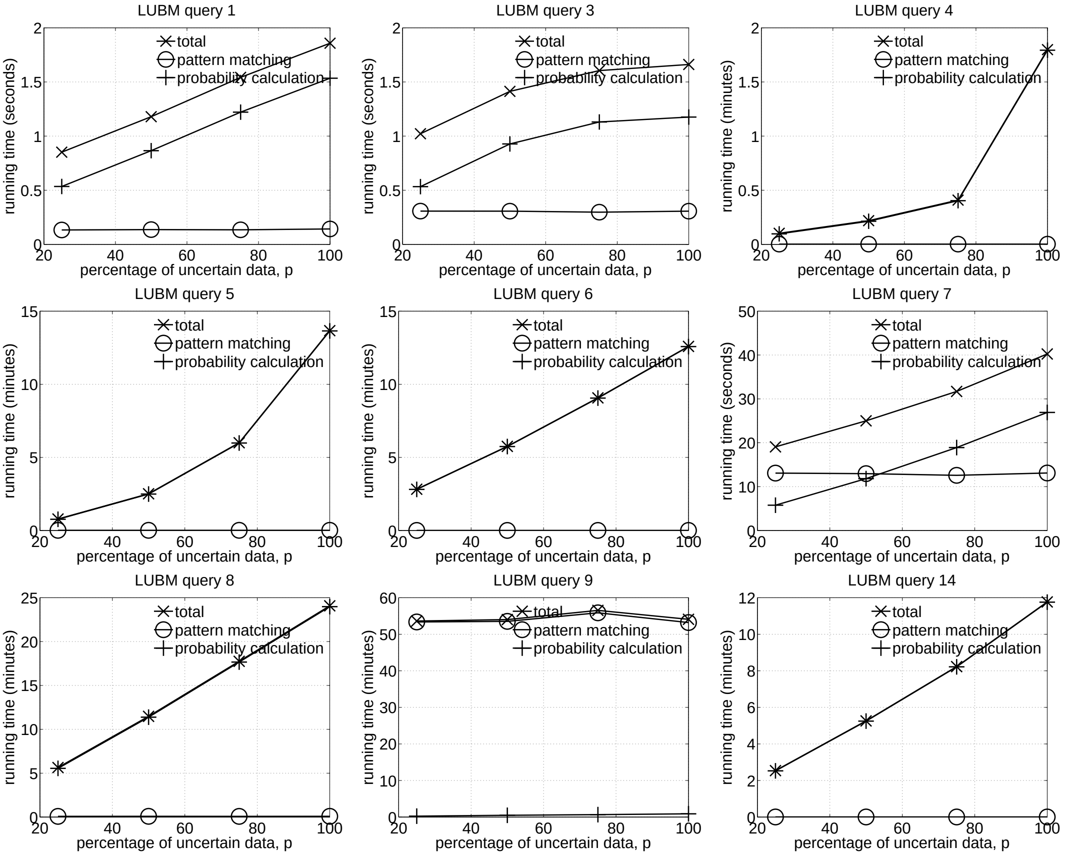


Figure 6.18: Running time of answering queries with RDFS reasoning as a function of the percentage of uncertain data p ($|D|$ = 50k, N_{node} = 10, $N_{\text{ideg}} = N_{\text{odeg}}$ = 2).

These figures also show the running time of pattern matching and probability calculation, which are the major components of the query evaluation. As mentioned in Section 6.2.2, the numbers of solutions to some queries do not vary with $|D|$ while the numbers of solutions to other queries increase linearly with $|D|$. To eliminate the effect of the number of solutions, we plot the running time per solution against $|D|$ in Fig. 6.17.

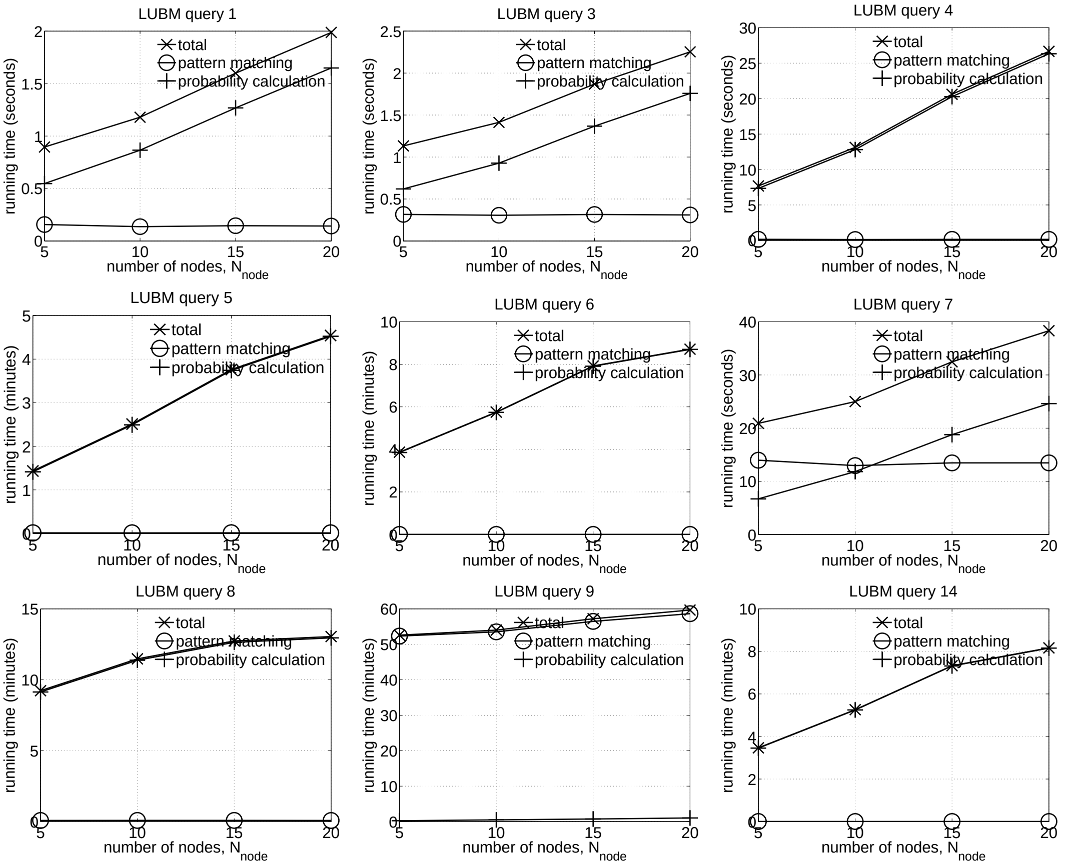


Figure 6.19: Running time of answering queries with RDFS reasoning as a function of the number of nodes N_{node} ($|D|$ = 50k, p = 50%, N_{ideg} = N_{odeg} = 2).

6.7.1 Pattern Matching

Fig. 6.17 shows that the pattern matching time per solution has different orders of growth for different queries. It does not grow with $|D|$ for the LUBM queries 6 and 14, grows sublinearly for the LUBM query 3, grows linearly for the LUBM queries 1, 4, and 5, grows polynomially for the LUBM queries 7 and 9, and grows exponentially for the LUBM query 8.

The pattern matching time does not depend on the percentage of uncertain data, the number of nodes, and the degree of nodes. This agrees with the experimental results shown in Figs. 6.18, 6.19, and 6.20.

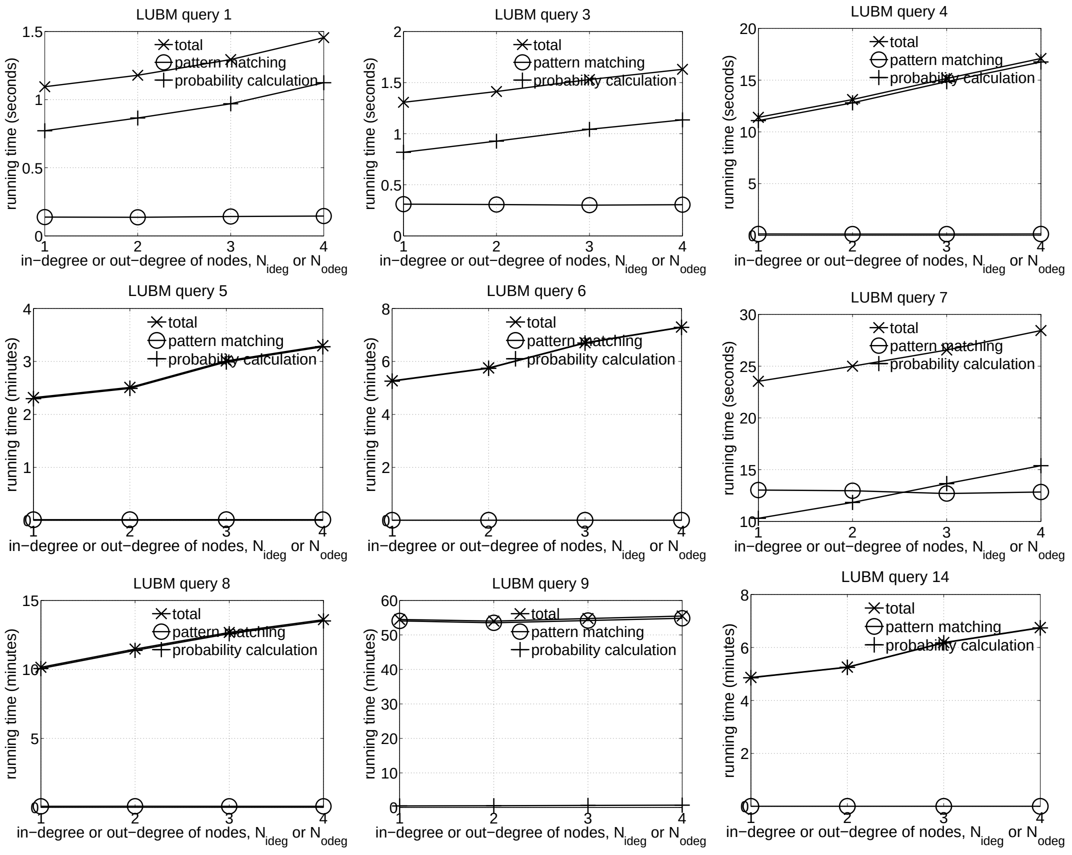


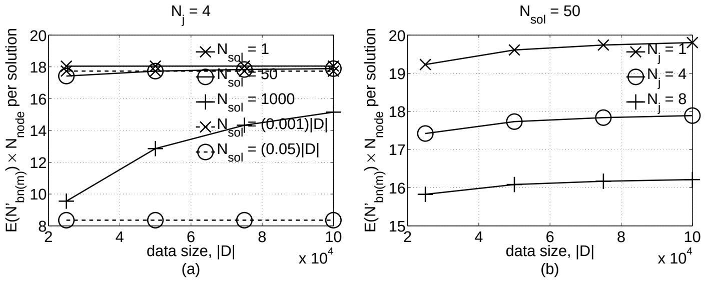
Figure 6.20: Running time of answering queries with RDFS reasoning as a function of the in-degree N_{ideg} or out-degree N_{odeg} of nodes ($|D|$ = 50k, p = 50$\%$, N_{node} = 10).

6.7.2 Probability Calculation

The quantity $E(N'_{\text{bn(m)}}) \times N_{\text{node}}$ is proportional to the expected running time of the probability calculation, where $E(N'_{\text{bn(m)}})$ is defined in (5.9) and N_{node} is the number of nodes per Bayesian network. We set the values of N_j, q, $|G_{\text{ext}}|$, and N_{sol} as follows, where N_j is the number of minimal justifications for a derived triple, q is the percentage of derived data, $|G_{\text{ext}}|$ is the size of matched data, and N_{sol} is the number of solutions. From Fig. 6.4, the value of N_j in the LUBM data is around 2.8. From Table 6.9, the values of N_j in the matched data of the solutions to the LUBM queries range between 1 and 8.5. We set N_j to be 1, 4, or 8. From Fig. 6.5, the value of q is between 19$\%$ and

Parameter	Setting
Number of minimal justifications for a derived triple, N_j	1, **4**, 8
Percentage of derived data, q	**19.5%**
Size of matched data, $\|G_{\text{ext}}\|$	**4**
Number of solutions, N_{sol}	1, **50**, 1000, 0.001$\|D\|$, 0.05$\|D\|$

Table 6.10: Settings for N_j, q, $|G_{\text{ext}}|$, and N_{sol} in the computation of $E(N'_{\text{bn(m)}})$.



(a)

(b)

Figure 6.21: $E(N'_{\text{bn(m)}}) \times N_{\text{node}}$ per solution as a function of $|D|$, N_{sol}, and N_j (p = 50%, N_{node} = 10, q = 19.5%, $|G_{\text{ext}}|$ = 4).

20%. We set q to be 19.5%. From Table 6.9, the values of $|G_{\text{ext}}|$ range between 1 and 6. We set $|G_{\text{ext}}|$ to be 4. From Table 6.8 and Fig. 6.6, the value of N_{sol} either does not vary with the data size $|D|$ or increases linearly with $|D|$. The settings for N_{sol} include small and large fixed values (1, 50, and 1000) and values that increase linearly with $|D|$ at slow and fast rates ($0.001|D|$ and $0.05|D|$). Table 6.10 summaries the settings for N_j, q, $|G_{\text{ext}}|$, and N_{sol}. The default values are in bold.

To study how the running time of probability calculation per solution varies with the data size $|D|$, we plot $E(N'_{\text{bn(m)}}) \times N_{\text{node}}$ per solution against $|D|$ for different values of N_{sol} and N_j in Fig. 6.21 and compare it with the experimental results shown in Fig. 6.17. From Fig. 6.21, $E(N'_{\text{bn(m)}}) \times N_{\text{node}}$ per solution changes very little except when N_{sol} = 50 or 1000. It increases sublinearly with $|D|$ when N_{sol} = 50 or 1000. This agrees with the experimental results that the running time of probability calculation per solution either changes very little or increases sublinearly with $|D|$.

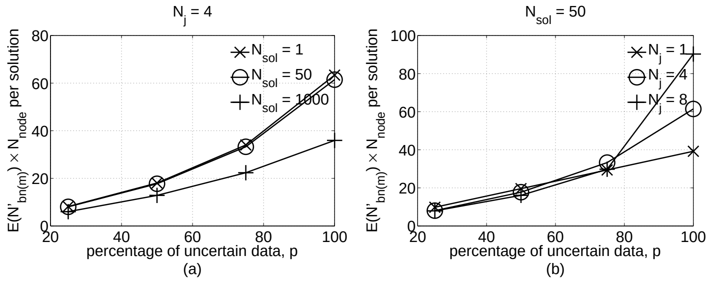


Figure 6.22: $E(N'_{\text{bn(m)}}) \times N_{\text{node}}$ per solution as a function of p, N_{sol}, and N_j ($|D|$ = 50k, N_{node} = 10, q = 19.5%, $|G_{\text{ext}}|$ = 4).

To study how the running time of probability calculation varies with the percentage of uncertain data p, we plot $E(N'_{\text{bn(m)}}) \times N_{\text{node}}$ per solution against p for different values of N_{sol} and N_j in Fig. 6.22 and compare it with the experimental results shown in Fig. 6.18. From Fig. 6.22, $E(N'_{\text{bn(m)}}) \times N_{\text{node}}$ increases linearly with p when N_j = 1. The order of its growth increases with N_j and decreases when N_{sol} is large like 1000.

For LUBM queries 1, 6, 7, 8, and 14, either the percentage of derived data q in the matched data is 0% or the number of minimal justifications for a derived triple N_j in the matched data is close to 1. Our model predicts that the running time of probability calculation for these queries increases linearly with p, and this agrees with the experimental results shown in Fig. 6.18.

For LUBM query 3, the value of q in the matched data is 0%. Our model predicts that the running time of probability calculation increases linearly with p. However, the running time increases sublinearly with p because of the entailment relationships between the matched data of the solutions. The graph pattern of query 3 consists of two triple patterns, (?X, rdf:type, ub:Publication) and (?X, ub:publicationAuthor, d0u0:AssistantProfessor0). The namespace prefixes in these two patterns are described in Table 6.4. The second triple pattern (?X, ub:publicationAuthor, d0u0:AssistantProfessor0) and the schema triple (ub:publicationAuthor, rdfs:domain, ub:Publication) together entail the first triple pattern (?X, rdf:type, ub:Publication) no matter what the value of the variable ?X is.

One solution to query 3 is $\{$(?X, d0u0ap0:Publication0)$\}$. If the matched data (d0u0ap0: Publication0, ub:publicationAuthor, d0u0: AssistantProfessor0) and (d0u0ap0: Publication0, rdf:type, ub:Publication) are both uncertain, they have to be in the same Bayesian network because of the entailment relationship between them. To compute the probability of the matched data, we perform inference on only one Bayesian network. However, in our model, the matched data are assumed to be drawn randomly from the data and no entailment relationship among the matched data is assumed. Inference on one or two Bayesian networks may be needed. Hence, our model overestimates the amount of inference performed, and the overestimate increases with p. This explains the sublinear growth of probability calculation time with respect to p for query 3.

For LUBM queries 4 and 5, N_j in the matched data is large. Our model predicts that the probability calculation time increases rapidly with p, and this agrees with the experimental result shown in Fig. 6.18.

For LUBM query 9, the value of N_j in the matched data is large. Our model predicts that the running time of probability calculation increases rapidly with p. However, the probability calculation time increases linearly with p because of the entailment relationships among the matched data of the solutions. The graph pattern of query 9 consists of six triple patterns. See Table 6.7a for the triple patterns. The namespace prefixes in the triple patterns are described in Table 6.4. The schema data of

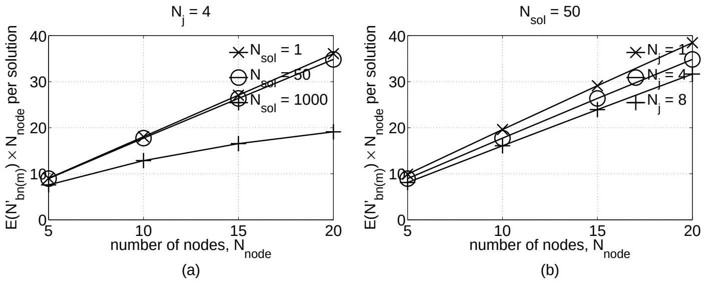


Figure 6.23: $E(N'_{\text{bn(m)}}) \times N_{\text{node}}$ per solution as a function of N_{node}, N_{sol}, and N_j ($|D|$ = 50k, p = 50%, q = 19.5%, $|G_{\text{ext}}|$ = 4).

the LUBM data state that the domain and range of ub:teacherOf are ub:Faculty and ub:Course respectively. Hence, the triple pattern (?Y, ub:teacherOf, ?Z), together with the schema data, entails both the triple patterns (?Y, rdf:type, ub:Faculty) and (?Z, rdf:type, ub:Course) no matter what the values of variables ?Y and ?Z are.

One solution to query 9 is $\{$(?X, d1u0: GraduateStudent53), (?Y, d1u0: AssociateProfessor1), (?Z, d1u0: GraduateCourse14)$\}$. The matched data of this solution are $\{d_1, d_2, d_3, r_1, r_2, r_3\}$. r_1, r_1, and r_1 are declared data, and d_1, d_2, and d_3 are derived data. They are described in Tables 6.11b and 6.11c respectively. The minimal justifications for the derived data d_1, d_2, and d_3 are shown in Tables 6.11d, and the numbers of minimal justifications for them are 1, 13, and 2 respectively. The probability bounds of the matched data are shown in Table 6.11e. The lower bound of the probability appears to contain a lot of RDF triples because there are 13 minimal justifications for the derived triple d_2. However, the expression for the lower bound of the probability can be simplified to contain four RDF triples because of the entailment relationship among the matched data. This explains why the running time of probability calculation does not increase rapidly with p for query 9.

To study how the running time of probability calculation varies with N_{node}, we plot $E(N'_{\text{bn(m)}}) \times N_{\text{node}}$ per solution against N_{node} for different values of N_{sol} and N_j in Fig. 6.23 and compare it with the experimental results shown in Fig. 6.19. From Fig. 6.23, $E(N'_{\text{bn(m)}}) \times N_{\text{node}}$ increases linearly with N_{node} except when N_{sol} is large like 1000. It increases sublinearly with N_{node} when N_{sol} is large. This agrees with the experimental results shown in Fig. 6.19. For LUBM queries 1, 3, 4, 7, and 9, the values of N_{sol} are small, and the time of probability calculation increases linearly with N_{node}. For LUBM queries 5, 6, 8, and 14, the values of N_{sol} are large, and the time of probability calculation increases sublinearly with N_{node}.

The running time of probability calculation would increase with the in-degree N_{ideg} or out-degree N_{odeg} of nodes because of the processing of increased parents of nodes. Fig. 6.20 shows that the probability calculation time increases linearly with N_{ideg} or N_{odeg}.

Symbol	RDF triple
h_1	(ub:teacherOf, rdfs:domain, ub:Faculty)
h_2	(ub:teacherOf, rdfs:range, ub:Course)
h_3	(ub:ResearchAssistant, rdfs:subClassOf, ub:Student)
h_4	(ub:AssociateProfessor, rdfs:subClassOf, ub:Professor)
h_5	(ub:Professor, rdfs:subClassOf, ub:Faculty)
h_6	(ub:advisor, rdfs:range, ub:Professor)
h_7	(ub:GraduateCourse, rdfs:subClassOf, ub:Course)

(a) Schema triples.

Symbol	RDF triple
r_1	(d1u0:GraduateStudent53, ub:advisor, d1u0:AssociateProfessor1)
r_2	(d1u0:GraduateStudent53, ub:takesCourse, d1u0:GraduateCourse14)
r_3	(d1u0:AssociateProfessor1, ub:teacherOf, d1u0:GraduateCourse14)
r_4	(d1u0:AssociateProfessor1, rdf:type, ub:AssociateProfessor)
r_5	(d1u0:UndergraduateStudent312, ub:advisor, d1u0:AssociateProfessor1)
r_6	(d1u0:GraduateStudent53, ub:advisor, d1u0:AssociateProfessor1)
r_7	(d1u0:UndergraduateStudent152, ub:advisor, d1u0:AssociateProfessor1)
r_8	(d1u0:UndergraduateStudent116, ub:advisor, d1u0:AssociateProfessor1)
r_9	(d1u0:GraduateStudent89, ub:advisor, d1u0:AssociateProfessor1)
r_{10}	(d1u0:GraduateStudent31, ub:advisor, d1u0:AssociateProfessor1)
r_{11}	(d1u0:UndergraduateStudent106, ub:advisor, d1u0:AssociateProfessor1)
r_{12}	(d1u0:GraduateStudent100, ub:advisor, d1u0:AssociateProfessor1)
r_{13}	(d1u0:AssociateProfessor1, ub:teacherOf, d1u0:GraduateCourse13)
r_{14}	(d1u0:AssociateProfessor1, ub:teacherOf, d1u0:Course16)
r_{15}	(d1u0:AssociateProfessor1, ub:teacherOf, d1u0:Course15)
r_{16}	(d1u0:GraduateCourse14, rdf:type, ub:GraduateCourse)
r_{17}	(d1u0:GraduateStudent53, rdf:type, ub:ResearchAssistant)

(b) Instance triples.

Symbol	RDF triple
d_1	(d1u0:GraduateStudent53, rdf:type, ub:Student)
d_2	(d1u0:AssociateProfessor1, rdf:type, ub:Faculty)
d_3	(d1u0:GraduateCourse14, rdf:type, ub:Course)

(c) Derived triples.

Symbol	Minimal justifications
d_1	$\{h_3, r_{17}\}$
d_2	$\{h_1, r_3\}$, $\{h_4, h_5, r_4\}$, $\{h_5, h_6, r_5\}$ $\{h_5, h_6, r_6\}$, $\{h_5, h_6, r_7\}$, $\{h_5, h_6, r_8\}$ $\{h_5, h_6, r_9\}$, $\{h_5, h_6, r_{10}\}$, $\{h_5, h_6, r_{11}\}$, $\{h_5, h_6, r_{12}\}$ $\{h_1, r_{13}\}$, $\{h_1, r_{14}\}$, $\{h_1, r_{15}\}$
d_3	$\{h_2, r_3\}$, $\{h_7, r_{16}\}$

(d) Minimal justifications for the derived triples d_1, d_2, and d_3.

$\mathcal{F}$	$d_1 \wedge d_2 \wedge d_3 \wedge r_1 \wedge r_2 \wedge r_3$
$Pr_{\text{lower}}(\mathcal{F})$	$Pr(r_{17} \wedge \bigvee_{i=3,4,\ldots,15} r_i \wedge (r_3 \vee r_{16}) \wedge r_1 \wedge r_2 \wedge r_3) = Pr(r_{17} \wedge r_1 \wedge r_2 \wedge r_3)$
$Pr_{\text{upper}}(\mathcal{F})$	$Pr(r_1 \wedge r_2 \wedge r_3)$

(e) Probability of the matched data $\{d_1, d_2, d_3, r_1, r_2, r_3\}$.

Table 6.11: Probability calculation of the matched data of a solution to LUBM query 9.

Chapter 7

Conclusion

This chapter presents a summary of our work and suggests three areas for future studies.

7.1 Summary and Contributions

In this thesis, we propose a probabilistic model for RDF called probabilistic RDFS (pRDFS), which models the uncertainties of correlated RDF data. We argue that being able to model correlated RDF data is necessary. First, RDF data using the RDFS vocabulary are correlated. Second, correlated data occur in practice. The syntax and semantics of pRDFS are defined in Chapter 3. Representing and performing probabilistic inference on correlated data are expensive. We use Bayesian networks to represent the correlated data and probabilistic logic sampling to perform approximate inference.

A pRDFS theory is inconsistent if it does not have any satisfying interpretation. In Proposition 2, we prove that if the probabilities of all inconsistent truth value assignments are zeros, then the pRDFS theory is consistent. In Propositions 3 and 4, we prove that it is sufficient to check the inconsistent truth value assignments created from the minimal justifications so that the computation of consistency checking can be reduced. We describe a consistency algorithm in Chapter 4. Moreover, for data that are frequently updated, we provide an incremental consistency checking algorithm, which performs fast rechecking each time the data are updated.

We extend a common form of SPARQL query to pRDFS by adding the truth value to each triple pattern of the query. We define an answer to the extended SPARQL query on pRDFS, which includes the solutions to the query pattern and the probabilities of the solutions. For solutions that contain derived data, their probabilities cannot be specified using a single probability value because the probability distribution of the derived data is not fully specified. In this case, the probability bounds of the solutions are computed. In Proposition 5, we prove that the computation of the probability bounds can be reduced using the minimal justifications.

We present an experimental evaluation of the running time performance of the consistency checking and query evaluation algorithms with respect to the data size $|D|$, the percentage of uncertain data p, the number of nodes in a Bayesian network N_{node}, and the degrees of nodes N_{ideg} and N_{odeg} in a network. The algorithms were tested on the Berlin SPARQL Benchmark (BSBM), the Lehigh University Benchmark (LUBM), and random uncertain data. For the consistency checking algorithm, the running time scales linearly with $|D|\log|D|$ and at most linearly with p, N_{node}, N_{ideg}, and N_{odeg}. For the incremental consistency checking algorithm, the running time in all test cases is under 160ms. The running time scales at most linearly with $|D|$, p, N_{node}, N_{ideg}, and

N_{odeg}. The only exception is that the running time scales exponentially with p in the LUBM data when a schema triple is added to the theory. For the query evaluation algorithm without RDFS reasoning, the running time scales at most linearly with $|D|$, p, N_{node}, N_{ideg}, and N_{odeg}. For the query evaluation algorithm with RDFS reasoning, the running time could scale exponentially with $|D|$ for some queries because of the pattern matching step. It scales at most linearly with p, N_{node}, N_{ideg}, and N_{odeg}.

Finally, we present three models for predicting the average-case running time of probability calculations in consistency checking, query evaluation without RDFS reasoning, and query evaluation with RDFS reasoning. They are $E(N_{\text{bn(e)}}) \times N_{\text{node}}$, $E(N_{\text{bn(m)}}) \times N_{\text{node}}$, and $E(N'_{\text{bn(m)}}) \times N_{\text{node}}$ respectively. The computations of $E(N_{\text{bn(e)}})$, $E(N_{\text{bn(m)}})$, and $E(N'_{\text{bn(m)}})$ are shown in (4.9), (5.8), and (5.9) respectively. The first two models match the experimental results. The last model overestimates the running time for queries, the triple patterns of which are correlated through the RDFS semantics no matter what the values of the variables in the triple patterns are. The discrepancy is due to the fact that our model assumes that the matched data of the solutions to the queries are drawn randomly from the RDFS closure of the data and may not be correlated.

7.2 Future Research

In this section, we outline three areas to enhance this work.

- The first one is the uncertainty supported. pRDFS assumes that the probability information of the uncertain data is fully specified. However, the probability information may be incomplete in practice. Moreover, the probabilities of the derived data from another pRDFS theories are intervals. Hence, pRDFS would have wider applications if it is able to handle partially specified probability information and uncertain data specified using interval probabilities.

- The second one is the inference supported. pRDFS now supports the RDFS inference. pRDFS could extend its support to other inferences like the inference of application-specific transitive properties, which is supported and used in [59], [37], and [56].

- The third one is the query supported. pRDFS currently supports SPARQL queries with basic graph patterns. pRDFS could extend its support to queries with complex graph patterns, which are formed by combining smaller patterns using SPARQL keywords OPTIONAL and UNION. However, solutions found by ignoring the truth values of the complex graph patterns do not cover all solutions to all instances of a pRDFS theory. Pre-processing and post-processing are required.

Bibliography

[1] AUER, S., BIZER, C., KOBILAROV, G., LEHMANN, J., CYGANIAK, R., AND IVES, Z. Dbpedia: a nucleus for a web of open data. In *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference* (Berlin, Heidelberg, 2007), ISWC'07/ASWC'07, Springer-Verlag, pp. 722–735.

[2] AUER, S., DEMTER, J., MARTIN, M., AND LEHMANN, J. Lodstats — an extensible framework for high-performance dataset analytics. In *Proceedings of the 18th international conference on Knowledge Engineering and Knowledge Management* (Berlin, Heidelberg, 2012), EKAW'12, Springer-Verlag, pp. 353–362.

[3] BARBARÁ, D., GARCIA-MOLINA, H., AND PORTER, D. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering 4*, 5 (1992), 487–502.

[4] BELLEAU, F., NOLIN, M.-A., TOURIGNY, N., RIGAULT, P., AND MORISSETTE, J. Bio2rdf: Towards a mashup to build bioinformatics knowledge systems. *Journal of Biomedical Informatics 41*, 5 (2008), 706–716.

[5] BERNERS-LEE, T. Linked data - design issues. Retrieved August 21, 2013, http://www.w3.org/DesignIssues/LinkedData.html, 2006.

[6] BERNERS-LEE, T., FIELDING, R., AND MASINTER, L. Uniform resource identifier (URI): Generic syntax. http://www.ietf.org/rfc/rfc3986.txt, 2005.

[7] BERNERS-LEE, T., HENDLER, J., AND LASSILA, O. The semantic web. *Scientific American 284*, 5 (May 2001), 34–43.

[8] BIEDERMANN, A., AND TARONI, F. Bayesian networks for evaluating forensic DNA profiling evidence: A review and guide to literature. *Forensic Science International: Genetics 6*, 2 (2012), 147 – 157.

[9] BIZER, C., CYGANIAK, R., AND GAUSS, T. The RDF book mashup: From web APIs to a web of data. In *SFSW* (2007), S. Auer, C. Bizer, T. Heath, and G. A. Grimnes, Eds., vol. 248 of *CEUR Workshop Proceedings*, CEUR-WS.org.

[10] BIZER, C., AND SCHULTZ, A. The Berlin SPARQL benchmark. *International Journal on Semantic Web and Information Systems 5*, 2 (2009), 1–24.

[11] BRICKLEY, D., AND GUHA, R. RDF vocabulary description language 1.0: RDF schema W3C recommendation. http://www.w3.org/TR/rdf-schema/, 2004.

[12] BROEKSTRA, J., AND KAMPMAN, A. SeRQL: A second generation RDF query language. SWAD-Europe Workshop on Semantic Web Storage and Retrieval, http://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/, 2003.

[13] CARROLL, J. J., DICKINSON, I., DOLLIN, C., REYNOLDS, D., SEABORNE, A., AND WILKINSON, K. Jena: implementing the semantic web recommendations. In *WWW (Alternate Track Papers & Posters)* (2004), S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, Eds., ACM, pp. 74–83.

[14] CHICKERING, D. M. Learning bayesian networks is NP-complete. In *Learning from Data: Artificial Intelligence and Statistics V* (1996), Springer-Verlag, pp. 121–130.

[15] COOPER, G. F. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence 42*, 2-3 (Mar. 1990), 393–405.

[16] COOPER, G. F., AND HERSKOVITS, E. A bayesian method for the induction of probabilistic networks from data. *Machine Learning 9*, 4 (Oct. 1992), 309–347.

[17] CYGANIAK, R., AND WOOD, D. Rdf 1.1 concepts and abstract syntax w3c last call working draft 23 july 2013. http://www.w3.org/TR/rdf11-concepts/, 2013.

[18] DAGUM, P., AND LUBY, M. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence 60*, 1 (Mar. 1993), 141–153.

[19] DALVI, N., AND SUCIU, D. Efficient query evaluation on probabilistic databases. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30* (2004), VLDB '04, VLDB Endowment, pp. 864–875.

[20] DALVI, N. N., RÉ, C., AND SUCIU, D. Probabilistic databases: diamonds in the dirt. *Communications of the ACM 52*, 7 (2009), 86–94.

[21] DE CAMPOS, L. M., FERNÁNDEZ-LUNA, J. M., AND HUETE, J. F. Bayesian networks and information retrieval: an introduction to the special issue. *Information Processing and Management: an International Journal 40*, 5 (Sept. 2004), 727–733.

[22] DESHPANDE, A., GUESTRIN, C., MADDEN, S. R., HELLERSTEIN, J. M., AND HONG, W. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30* (2004), VLDB '04, VLDB Endowment, pp. 588–599.

[23] DONG, X. L., BERTI-EQUILLE, L., AND SRIVASTAVA, D. Integrating conflicting data: The role of source dependence. *PVLDB 2*, 1 (2009), 550–561.

[24] DUERST, M., AND SUIGNARD, M. Internationalized resource identifiers (IRIs). http://www.ietf.org/rfc/rfc3987.txt, 2005.

[25] FAGIN, R. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci. 58*, 1 (1999), 83–99.

[26] FUHR, N., AND RÖLLEKE, T. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems 15*, 1 (1997), 32–66.

[27] FUKUSHIGE, Y. Representing probabilistic knowledge in the semantic web. Retrieved August 21, 2013, http://www.w3.org/2004/09/13-Yoshio/PositionPaper.html, 2004.

[28] FUKUSHIGE, Y. Representing probabilistic relations in RDF. In *ISWC-URSW* (2005), P. C. G. da Costa, K. B. Laskey, K. J. Laskey, and M. Pool, Eds., pp. 106–107.

[29] FUNG, R. M., AND CHANG, K.-C. Weighing and integrating evidence for stochastic simulation in bayesian networks. In *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence* (Amsterdam, The Netherlands, The Netherlands, 1990), UAI '89, North-Holland Publishing Co., pp. 209–220.

[30] GUO, Y., PAN, Z., AND HEFLIN, J. An evaluation of knowledge base systems for large OWL datasets. In *International Semantic Web Conference* (2004), S. A. McIlraith, D. Plexousakis, and F. van Harmelen, Eds., vol. 3298 of *Lecture Notes in Computer Science*, Springer, pp. 274–288.

[31] GUPTA, R., AND SARAWAGI, S. Creating probabilistic databases from information extraction models. In *VLDB* (2006), U. Dayal, K.-Y. Whang, D. B. Lomet, G. Alonso, G. M. Lohman, M. L. Kersten, S. K. Cha, and Y.-K. Kim, Eds., ACM, pp. 965–976.

[32] HECKERMAN, D. E., HORVITZ, E. J., AND NATHWANI, B. N. Toward normative expert systems: Part I. the pathfinder project. *Methods of information in medicine 31*, 2 (1992), 90–105.

[33] HECKERMAN, D. E., AND NATHWANI, B. N. An evaluation of the diagnostic accuracy of pathfinder. *Computers and Biomedical Research 25*, 1 (1992), 56–74.

[34] HECKERMAN, D. E., AND NATHWANI, B. N. Toward normative expert systems: Part II. probability-based representations for efficient knowledge acquisition and inference. *Methods of information in medicine 31*, 2 (1992), 106–116.

[35] HENRION, M. Propagating uncertainty in bayesian networks by probabilistic logic sampling. In *UAI* (1986), J. F. Lemmer and L. N. Kanal, Eds., Elsevier, pp. 149–164.

[36] HOEFFDING, W. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association 58*, 301 (1963), 13–30.

[37] HUANG, H., AND LIU, C. Query evaluation on probabilistic RDF databases. In *WISE* (2009), G. Vossen, D. D. E. Long, and J. X. Yu, Eds., vol. 5802 of *Lecture Notes in Computer Science*, Springer, pp. 307–320.

[38] KALYANPUR, A. *Debugging and repair of OWL ontologies.* PhD thesis, University of Maryland at College Park, College Park, MD, USA, 2006.

[39] KLYNE, G., AND CARROLL, J. J. Resource description framework (RDF): Concepts and abstract syntax W3C recommendation. http://www.w3.org/TR/rdf-concepts/, 2004.

[40] KOBILAROV, G., SCOTT, T., RAIMOND, Y., OLIVER, S., SIZEMORE, C., SMETHURST, M., BIZER, C., AND LEE, R. Media meets semantic web — how the bbc uses dbpedia and linked data to make connections. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications* (Berlin, Heidelberg, 2009), ESWC 2009 Heraklion, Springer-Verlag, pp. 723–737.

[41] LANDUYT, D., BROEKX, S., D'HONDT, R., ENGELEN, G., AERTSENS, J., AND GOETHALS, P. L. A review of bayesian belief networks in ecosystem service modelling. *Environmental Modelling & Software 46* (2013), 1 – 11.

[42] LARRAÑAGA, P., KUIJPERS, C. M., MURGA, R. H., AND YURRAMENDI, Y. Learning bayesian network structures by searching for the best ordering with genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans 26*, 4 (July 1996), 487–493.

[43] LARRAÑAGA, P., POZA, M., YURRAMENDI, Y., MURGA, R. H., AND KUIJPERS, C. M. H. Structure learning of bayesian networks by genetic algorithms: A performance analysis of control parameters. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*, 9 (Sept. 1996), 912–926.

[44] LIAN, X., AND CHEN, L. Efficient query answering in probabilistic RDF graphs. In *SIGMOD Conference* (2011), T. K. Sellis, R. J. Miller, A. Kementsietsidis, and Y. Velegrakis, Eds., ACM, pp. 157–168.

[45] MCGUINNESS, D. L., AND VAN HARMELEN, F. OWL web ontology language overview W3C recommendation. http://www.w3.org/TR/owl-features/, 2004.

[46] NOY, N., AND RECTOR, A. Defining n-ary relations on the semantic web. Retrieved August 21, 2013, http://www.w3.org/TR/swbp-n-aryRelations/, Apr. 2006.

[47] PEARL, J. A constraint-propagation approach to probabilistic reasoning. In *UAI '85: Proceedings of the First Annual Conference on Uncertainty in Artificial Intelligence* (1985), pp. 357–370.

[48] PEARL, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[49] PÉREZ, J., ARENAS, M., AND GUTIERREZ, C. Semantics and complexity of SPARQL. *ACM Trans. Database Syst. 34*, 3 (Sept. 2009), 16:1–16:45.

[50] PRUD'HOMMEAUX, E., AND SEABORNE, A. SPARQL query language for RDF W3C recommendation. http://www.w3.org/TR/rdf-sparql-query/, 2008.

[51] SEABORNE, A. RDQL - a query language for RDF W3C member submission 9 January 2004. http://www.w3.org/Submission/RDQL/, 2004.

[52] SEN, P., AND DESHPANDE, A. Representing and querying correlated tuples in probabilistic databases. In *ICDE* (2007), IEEE, pp. 596–605.

[53] SHACHTER, R. D., AND PEOT, M. A. Simulation approaches to general probabilistic inference on belief networks. In *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence* (Amsterdam, The Netherlands, The Netherlands, 1990), UAI '89, North-Holland Publishing Co., pp. 221–234.

[54] SPIRTES, P., AND GLYMOUR, C. An algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review 9* (1991), 62–72.

[55] SPIRTES, P., GLYMOUR, C., AND SCHEINES, R. *Causation, Prediction, and Search, 2nd edition*. The MIT Press, 2000.

[56] SUCHANEK, F. M., KASNECI, G., AND WEIKUM, G. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web* (New York, NY, USA, 2007), WWW '07, ACM, pp. 697–706.

[57] SZETO, C.-C., HUNG, E., AND DENG, Y. Modeling and querying probabilistic RDFS data sets with correlated triples. In *APWeb* (2011), X. Du, W. Fan, J. Wang, Z. Peng, and M. A. Sharaf, Eds., vol. 6612 of *Lecture Notes in Computer Science*, Springer, pp. 333–344.

[58] SZETO, C.-C., HUNG, E., AND DENG, Y. SPARQL query answering with RDFS reasoning on correlated probabilistic data. In *WAIM* (2011), H. Wang, S. Li, S. Oyama, X. Hu, and T. Qian, Eds., vol. 6897 of *Lecture Notes in Computer Science*, Springer, pp. 56–67.

[59] UDREA, O., SUBRAHMANIAN, V. S., AND MAJKIC, Z. Probabilistic RDF. In *IRI* (2006), IEEE Systems, Man, and Cybernetics Society, pp. 172–177.

[60] WEBER, P., MEDINA-OLIVA, G., SIMON, C., AND IUNG, B. Overview on bayesian networks applications for dependability, risk analysis and maintenance areas. *Engineering Applications of Artificial Intelligence 25*, 4 (June 2012), 671–682.

[61] YAO, S. B. Approximating block accesses in database organizations. *Commun. ACM 20*, 4 (Apr. 1977), 260–261.

[62] ZHANG, N., AND POOLE, D. A simple approach to bayesian network computations. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence* (1994), pp. 171–178.