WP8-D82

Efficient Abductive Reasoning Technologies

Project full title:	Knowledge Driven Data Exploitation
Project acronym:	K-Drive
Grant agreement no.:	286348
Project instrument:	EU FP7/Maria-Curie IAPP/PEOPLE WP 2011
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	UNIABDN, IBM/WP8-D82/D/PU/b1
Responsible editors:	Jeff Z. Pan, Akanimo Samuel Okure, Yuting Zhao and Alessandro Faraotti
Reviewers:	Marco Monti
Contributing participants:	UNIABDN, IBM
Contributing workpackages:	WP8
Contractual date of deliverable:	30 June 2016
Actual submission date:	30 June 2016

Abstract

Traditionally queries ask for what happens and what does not happen in the knowledge base. While hypothesises take one step forward, tell why something happens and why something does not happen in the knowledge base. In addition to deductive reasoning and explanations of its results. Hypothesis generation requires abductive reasoning support. To support such a new service, the objective of this deliverable is to understand existing abductive reasoning approaches, in which we focus on explaining why certain results can be inferred. ABox abduction is an important reasoning facility in Description Logics (DLs). It finds all minimal sets of ABox axioms, called abductive solutions, which should be added to a background ontology to enforce entailment of an observation which is a specified set of ABox axioms. However, ABox abduction is far from practical by now because there lack feasible methods working in finite time for expressive DLs. To pave a way to practical ABox abduction, this paper proposes a new problem for ABox abduction and a new method for computing abductive solutions accordingly. The proposed problem guarantees finite number of abductive solutions. The proposed method works in finite time for a very expressive DL, SHOIQ , which underpins the W3C standard language OWL 2, and guarantees soundness and conditional completeness of computed results. Experimental results on benchmark ontologies show that the method is feasible and can scale to large ABoxes.

Keyword List

Abductive reasoning, Justification, ABox Abduction; Ontologies; Description Logics; Logic Programming

Project funded by the European Commission within the 7th Framework Programme/Maria Curie Industry-Academia Partnerships and Pathways schema/PEOPLE Work Programme 2011.

Efficient Abductive Reasoning Technologies

Jeff Pan[1], **Akanimo Samuel Okure** [1], **Yuting Zhao**[2] **and Alessandro Faraotti**[2]

[1] Department of Computing Science, Aberdeen University, UK
Email: {sam, jeff.z.pan}@abdn.ac.uk
[2] IBM Italy, Roma, Italy
Email: {yuting.zhao, alessandro.faraotti}@it.ibm.com

30 June 2016

Abstract
Traditionally queries ask for what happens and what does not happen in the knowledge base. While hypothesises take one step forward, tell why something happens and why something does not happen in the knowledge base. In addition to deductive reasoning and explanations of its results. Hypothesis generation requires abductive reasoning support. To support such a new service, the objective of this deliverable is to understand existing abductive reasoning approaches, in which we focus on explaining why certain results can be inferred. ABox abduction is an important reasoning facility in Description Logics (DLs). It finds all minimal sets of ABox axioms, called abductive solutions, which should be added to a background ontology to enforce entailment of an observation which is a specified set of ABox axioms. However, ABox abduction is far from practical by now because there lack feasible methods working in finite time for expressive DLs. To pave a way to practical ABox abduction, this paper proposes a new problem for ABox abduction and a new method for computing abductive solutions accordingly. The proposed problem guarantees finite number of abductive solutions. The proposed method works in finite time for a very expressive DL, SHOIQ , which underpins the W3C standard language OWL 2, and guarantees soundness and conditional completeness of computed results. Experimental results on benchmark ontologies show that the method is feasible and can scale to large ABoxes.

Keyword List
Abductive reasoning, Justification, ABox Abduction; Ontologies; Description Logics; Logic Programming

Contents

1. Introduction

Workpackage 8 of K-Drive aims to further extend the query generation (WP3, WP5, WP6), to generate not only single queries about "what can be found in the knowledge base", but also "why something happen/not happen in the knowledge base". Actually it applies not only deductive reasoning, but also abductive inferences.

In WP3, we have studied and developed technologies to generate insightful queries from large volume of use case data efficiently and allowing users to conveniently and intuitively browse and exploit data. Technically, it firstly identifies the key dimensions by the experts, or extract key features by automatic extractor, and then generate key queries and compute answers.

In WP5, we have extended the technologies developed in WP3 to generate queries dynamically. This involves the elimination of out-dated queries, and proposal of new queries.

In WP6, we have investigated stream reasoning technologies that can efficiently update query results, in which we focus on be large volume data and its high frequency updating.

The W3C organization has proposed the standard Web Ontology Language (OWL), whose newest version is OWL 2 (http://www.w3.org/TR/owl2-overview/), to model ontologies for a wide range of applications. OWL is underpinned by Description Logics (DLs) (Baader et al., 2003). For example, the two important species of OWL, namely OWL DL and OWL 2 DL, are syntactic variants of two DLs $\mathcal{SHOIN}(\mathbf{D})$ and $\mathcal{SROIQ}(\mathbf{D})$ respectively (Horrocks et al., 2003; Grau et al., 2008). With formal semantics, DLs provide a number of well-defined reasoning facilities which widen the applicability of DL ontologies, including OWL ontologies. Traditionally queries ask for what happens and what does not happen in the knowledge base. While hypothesises take one step forward, tell why something happens and why something does not happen in the knowledge base. In addition to deductive reasoning and explanations of its results. Hypothesis generation requires abductive reasoning support. So in this deliverable, we focus on abductive reasoning technologies in *Description Logic* (DL).

2. Background of Abductive Reasoning

The term "abduction" was introduced by the philosopher Charles Sanders Peirce (Peirce, 1992; Peirce, 1994). He made the first attempt to distinguish it as a basic form of logical inference together with induction and deduction. The origin of abduction appears to come from a syllogistic form of reasoning that was discussed by Aristotle in the Prior Analytics Aristotle (1967). The syllogism finds a domain that would

make a given conclusion more desirable. In English, the term " abduction" has several meanings such as forcefully taking. Peirce also refereed to this form of reasoning as retroduction and presumption.

2.1 Peirce's Concept of Abduction

Peirce's had genuine interest in logic, and this came as a result of his drive to formalise the methods of research, in accompliance with the advancement in science. He considered abduction as an attempt to provide logic with tools for hypothesis generation. Most philosophers do not agree with the idea that logic could be used in generating hypothesis, but they are rather concerned with the methods for testing such hypothesis. Peirce classified the three fundamental form of inference into two categories: *explicit inference* and *amplicit inference*. The Explicit inference are those inference that the conclusion are generated from a domain whereas the amplicite inference refers to the inference that the conclusion are not necessarily generated from the domain. Therefore deduction is a form of explicit inference whereas abduction and induction are forms of amplicite inference.

Peirce also states that the three fundamental forms of logical inference represents the three stages of scientific inquiry which includes:

- Abduction proposes hypotheses;
- Deduction derives the consequences of the hypotheses;
- Induction test or verifies hypotheses.

The factors that are considered when performing abductive reasoning are as follows:

- Generating or finding hypotheses; and
- Selecting the best hypothesis from a set of other hypothesis .

In regards to selecting the best hypothesis, Peirce proposed the major factors that should be considered in doing this, these includes:

- A hypothesis must be able to explain facts
- The verification of hypothesis via experimentation is a must; and,
- The economy cost must be put into consideration when selecting the best explanation.

Putting the economy cost into considerations when selecting the best hypothesis, Peirce highlighted some factors that plays important role in achieving this. The first one states that it would be good to select hypothesis that the verification process would cost less with respect to time, money and energy. Secondly, when choosing hypothesis one must consider the effect it will have on other projects, that is if the

chosen hypothesis turns out to be incorrect, so effort should be made to avoid such hypothesis. And last factor stresses the need to ascertain the real value of the proposed hypothesis. There are a number of methods to ascertain the real value of hypothesis, to achieve this Peirce proposed the notion of simplicity. By this it means that hypothesis that seems more natural should be regarded as simpler hypothesis. Also the likelihood of a hypothesis is another factor, Peirce suggested that special care should be taken when adopting measures of likelihood as in most cases it is subjective. Finally, the hypothesis breath is an important point that should also considered very carefully as this shows volume of facts that an hypothesis can explain in a sense the hypothesis that could explain more facts are more useful.

2.2 Abduction based explanation

Description logics based knowledge representation systems have gained growing popularity especially in the context of Semantic Web application, where there is a raising need for tools that could offer non standard reasoning services, and one of such reasoning service which is interesting from both human perspective as well as the knowledge engineering commuting is the idea of abduction. Abductive reasoning (Abduction) is a novel reasoning method that have been successfully devised for explanation. In recent decade, the importance of abduction have been demonstrated in the field of of logic, artificial intelligence and sciences. It has been acknowledged in the field of semantic web community that deductive reasoning is insufficient when dealing with a new class of problem that involves explanation, especially in health care and life sciences where these applications seeks to explain the observations.

The significant feature of abductive reasoning tool is that they provide explanations for causes or justifications for the occurrence of an event. Generating explanations for a observations is non trivial task because not all the possible explanations generated are valid or very useful, hence it thus an important task for the abductive system to provide the best explanation from a set of possible explanations and this is often challenging. Another important point to consider when computing solution for abductive problem is that is the space of such solutions can sometimes be infinite, therefore there is need to narrow the solution space by removing the unwanted solutions, so that only solutions with high pragmatic values should be allowed. There exist some universally acceptable criteria which are taken into considerations when computing solutions for abductive problems, some of the important criteria are as follows:

Consistency: This helps to discard solutions which are inconsistent with the knowledge base;

Relevance: This condition removes those solutions that entails the observation by them selves without considering the information in the background knowledge. Such solutions makes the problem of little importance instead of actually solving it.

Minimality: This criterion ensures that the solution to abductive problem does not hypothesis(abduce) more than is necessary, that is it should not contain superfluous or vague information.

2.3 ABox-Abdcution in Description Logic

Besides standard reasoning facilities proposed in the DL handbook (Baader et al., 2003) such as checking whether a DL ontology is consistent and checking whether an axiom is entailed by a DL ontology, some non-standard reasoning facilities have been proposed as well. A well-known non-standard reasoning facility, called axiom pinpointing (Baader & Peñaloza, 2007; Schlobach & Cornet, 2003) or justification computing (Kalyanpur et al., 2007), is to compute minimal sets of axioms responsible for an entailment of a DL ontology. This facility is used to explain why some axioms are entailed by a DL ontology and suggest solutions to remove these entailments. Corresponding to this facility, another well-known non-standard reasoning facility, usually referred to as abduction or abductive reasoning (Elsenbroich et al., 2006), is to compute minimal sets of axioms that should be added to a background ontology to enforce entailment of an observation which is a set of axioms. This facility is used to explain why some axioms are not entailed by a DL ontology and suggest solutions to enforce these entailments.

Since a DL ontology is composed of a TBox, which stores intensional knowledge, as well as an ABox, which stores extensional knowledge, there are two sub-facilities for abductive reasoning in DLs. One is TBox abduction, the other is ABox abduction. They differ from each other on the kinds of information that is allowed to appear in computed results. For TBox abduction, only concepts, roles or TBox axioms (e.g. concept or role inclusion axioms) are allowed. For ABox abduction, only ABox axioms (e.g. concept or role assertions) are allowed. ABox abduction has its unique characteristics and cannot be treated as axiom pinpointing or solved by existing methods for TBox abduction. See this in the following example.

Example 1. Let the background ontology $\mathcal{O}$ consist of the following four TBox axioms

$$\mathsf{Clever} \sqcap \mathsf{Diligent} \sqsubseteq \exists \mathsf{isRewarded}.\mathsf{Competition},$$

$$\exists \mathsf{isRewarded}.\mathsf{Competition} \sqsubseteq \mathsf{Extraordinary},$$

$$\mathsf{Extraordinary} \sqsubseteq \mathsf{Person},$$

$$\mathsf{Person} \sqsubseteq \mathsf{Extraordinary} \sqcup \mathsf{Ordinary},$$

and the following two ABox axioms

$$\mathsf{Person}(\mathsf{Tom}), \qquad \mathsf{Clever}(\mathsf{Tom}).$$

The first TBox axiom says that someone who is clever and diligent will be rewarded in some competition. The second TBox axiom says that someone rewarded in some competition is extraordinary. The third TBox axiom says that someone extraor-

dinary is a person. The last TBox axiom says that a person is extraordinary or ordinary. The first ABox axiom says that Tom is a person, while the second one says that Tom is clever. When we are informed that Tom is extraordinary, we may want to know why this happens. However, the current ontology $\mathcal{O}$ does not entail $\mathsf{Extraordinary(Tom)}$, so we cannot find explanations in $\mathcal{O}$ through axiom pinpointing. In this situation, we need to introduce a hypothesis which is a set of axioms absent in $\mathcal{O}$ such that the union of it and $\mathcal{O}$ entails $\mathsf{Extraordinary(Tom)}$. For example, we can introduce a hypothesis $\{\mathsf{Diligent(Tom)}\}$, then $\mathcal{O} \cup \{\mathsf{Diligent(Tom)}\}$ entails $\mathsf{Extraordinary(Tom)}$. However, existing methods for TBox abduction cannot directly be applied to compute this hypothesis, because these methods do not consider nominals and the hypothesis involves a nominal $\{\mathsf{Tom}\}$. This example shows that we need particular methods for ABox abduction that are different from existing methods for axiom pinpointing or TBox abduction.

By now there are only few methods for ABox abduction. One known method is based on backward inference (Peraldi et al., 2007). It restricts axioms in the given ontology to some special forms. Moreover, it does not guarantee any minimality for computed results. Another known method is based on some complex tableaux and resolution techniques (Klarman et al., 2011). It works on the DL $\mathcal{ALC}$ which is a fragment of $\mathcal{SHOIN}$, the DL corresponding to OWL DL. $\mathcal{ALC}$ is obtained from $\mathcal{SHOIN}$ by disallowing number restrictions, nominals, inverse roles, role inclusion axioms and transitivity axioms. Moreover, the method proposed in (Klarman et al., 2011) does not guarantee termination because it allows arbitrarily many nested existential/value restrictions appearing in computed results. Consider an ontology consisting of only the following axiom, which says that something has a person as its parent is a person.

$$\exists \mathsf{hasParent}.\mathsf{Person} \sqsubseteq \mathsf{Person}$$

The method will compute infinitely many results for the observation that Amy is a person (i.e. $\{\mathsf{Person(Amy)}\}$). Each result consists of a single concept assertion of the form $\exists \mathsf{hasParent}.\exists \mathsf{hasParent}.\ldots.\mathsf{Person(Amy)}$, in which the concept is an existential restriction having arbitrarily many nested $\exists \mathsf{hasParent}$. Note that the ultimate results computed by the method should have certain minimality, while the method always computes all candidate results that may not be minimal before selecting out ultimate ones. Hence the method will not terminate even when there are finitely many ultimate results but infinitely many candidate results. The present situation for ABox abduction urges us to develop practical methods, which should be able to efficiently (at least in finite time) compute minimal results for expressive DLs.

To ensure all minimal results to be computed in finite time, we need to guarantee that there are only finitely many minimal results. Thus, we first propose a new problem for ABox abduction. This problem aims to compute minimal sets of ABox axioms, called abductive solutions, which should be added to a DL ontology to make a given observation entailed by the ontology, where all ABox axioms in an abductive solution are composed of individual names in the ontology and user-specified predicates. The user-specified predicates, called abducible predicates, can be arbitrary concepts or roles, but the number of abducible predicates that can be used should be finite so that the number of abductive solutions is finite. The introduction of abducible predicates will give users flexibility to formulate the explanations for an observation.

To seek methods to solve the proposed problem, we consider successful tools on abductive reasoning in logic programming (Kakas et al., 1998), such as two state-of-the-art abduction systems CIFF (Mancarella et al., 2009) and $\mathcal{A}$ -system (Kakas et al., 2001). These tools are built on modern Prolog engines, but they only allow the background theory to be a normal logic program, which corresponds to a plain datalog program extended with negation-as-failure. However, the DLs that underpin OWL, such as $\mathcal{SHOIN}$ (OWL DL without datatypes) and $\mathcal{SROIQ}$ (OWL 2 DL without datatypes), do not contain negation-as-failure and cannot be directly translated to plain datalog due to the presence of existential restrictions. For example, the axiom $A \sqsubseteq \exists r.B$ can only be translated to a first-order rule $\forall x : A(x) \rightarrow \exists y : r(x, y) \wedge B(y)$ which is not in plain datalog, because plain datalog programs do not contain function symbols or existentially quantified variables, while function symbols must be introduced when eliminating the existentially quantified variable y . Hence, we propose a reduction based method for ABox abduction. It first reduces the proposed problem for ABox abduction to a traditional abduction problem in logic programming in which the background theory is a plain datalog program, then extracts true results from the abductive solutions for the reduced abduction problem. This method can not only work for very expressive DLs including $\mathcal{SHOIN}$ and $\mathcal{SROIQ}$ but also make use of efficient techniques in modern Prolog engines. Since the reduction is approximate and cannot guarantee semantic equivalence, the method cannot guarantee completeness, i.e., some abductive solutions may be missed, but it still guarantees soundness, i.e., all output results are actually abductive solutions. We present the method with $\mathcal{SHOIQ}$ which underpins both OWL DL and OWL 2 DL.

To verify the practicality of the proposed method, we conduct experiments on a series of benchmark ontologies that have large ABoxes, including those previously used to compare modern DL reasoners (Motik & Sattler, 2006) and those coming from the well-known University Benchmark (UOBM) (Ma et al., 2006). Experimental results on these ontologies show that the proposed method works well for hundreds of abducible predicates and up to half a million ABox axioms. This demonstrates that the proposed method paves a way towards practical ABox abduction in large DL ontologies.

3. Preliminaries

In this section, we introduce the DL $\mathcal{SHOIQ}$ and disjunctive datalog, both of which express background theories that we consider. Moreover, we also introduce a method for compiling $\mathcal{SHIQ}$ to disjunctive datalog (Hustadt et al., 2007) and a method for axiomatizing equality (Fitting 1996), both of which are highly related to our proposed method.

3.1 The Description Logic $\mathcal{SHOIQ}$

Description Logics (DLs) (Baader et al., 2003) are logical foundations of OWL. $\mathcal{SHOIQ}$ is a very expressive DL that underpins OWL DL and OWL 2 DL, since OWL DL is a syntactic variant of $\mathcal{SHOIN}(\mathbf{D})$ (Horrocks et al., 2003) and OWL 2 DL is a syntactic variant of $\mathcal{SROIQ}(\mathbf{D})$ (Grau et al., 2008). Throughout this deliverable, we use the DL syntax of $\mathcal{SHOIQ}$ as it is more compact.

Let N_R be a set of role names. A $\mathcal{SHOIQ}$ role (simply a role) is either some $r \in N_R$ (atomic role) or an inverse role r^- for $r \in N_R$. Let $\mathsf{Inv}(r) = r^-$ and $\mathsf{Inv}(r^-) = r$ for $r \in N_R$. Let N_C be a set of concept names and N_I a set of individual names. The sets N_R, N_C and N_I are mutually disjoint. The set of $\mathcal{SHOIQ}$ concepts is the smallest set recursively defined as follows. Each $A \in N_C$ (atomic concept) or each $\{a\}$ (nominal) where $a \in N_I$ is a $\mathcal{SHOIQ}$ concept. For $\mathcal{SHOIQ}$ concepts C and D, roles r and s, and a nonnegative integer n, the following concepts are also $\mathcal{SHOIQ}$ concepts: $\top$ (top concept), $\bot$ (bottom concept), $\neg C$ (negation), $C \sqcap D$ (conjunction), $C \sqcup D$ (disjunction), $\exists r.C$ (existential restriction), $\forall r.C$ (value restriction), $\leq_n s.C$ and $\geq_n s.C$ (qualifying number restrictions). A concept or a role is said to be literal if it is atomic or negated atomic.

A $\mathcal{SHOIQ}$ ontology consists of a $\mathcal{SHOIQ}$ TBox and a $\mathcal{SHOIQ}$ ABox. A $\mathcal{SHOIQ}$ TBox $\mathcal{T}$ is a finite set of TBox axioms, including concept inclusion axioms $C \sqsubseteq D$, role inclusion axioms $r \sqsubseteq s$ and transitivity axioms $\mathsf{Tra}(r)$, where C and D are $\mathcal{SHOIQ}$ concepts, and r and s are roles. It is required that $r \sqsubseteq s \in \mathcal{T}$ imply $\mathsf{Inv}(r) \sqsubseteq \mathsf{Inv}(s) \in \mathcal{T}$, while $\mathsf{Tra}(r) \in \mathcal{T}$ imply $\mathsf{Tra}(\mathsf{Inv}(r)) \in \mathcal{T}$, for any roles r and s. Let $\sqsubseteq^*$ denote the reflexive-transitive closure of $\sqsubseteq$. A role r is said to be transitive if $\mathsf{Tra}(s) \in \mathcal{T}$ for some role s such

that $s \sqsubseteq^* r$ and $r \sqsubseteq^* s$. r is said to be simple if there is no transitive role s such that $s \sqsubseteq^* r$. r is said to be complex if it is not simple. To guarantee decidability of $\mathcal{SHOIQ}$, it is required that any role s used in qualifying number restrictions $\leq_n s.C$ or $\geq_n s.C$ be simple. A $\mathcal{SHOIQ}$ ABox $\mathcal{A}$ is a finite set of ABox axioms, including concept assertions $C(a)$, role assertions $r(a,b)$, equality assertions $a \approx b$ and inequality assertions $a \not\approx b$, where C is a $\mathcal{SHOIQ}$ concept, r is a literal role, and a and b are individual names in N_I. When C is a literal (resp. atomic or negated atomic) concept, $C(a)$ is said to be a literal (resp. atomic or negated) concept assertion. When r is a literal (resp. atomic or negated atomic) role, $r(a,b)$ is said to be a literal (resp. atomic or negated) role assertion.

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \bullet^{\mathcal{I}})$ consists of a set $\Delta^{\mathcal{I}}$, called the domain of $\mathcal{I}$, and a function $\bullet^{\mathcal{I}}$ that maps every concept name A to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, every role name r to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and every individual name a to $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The interpretation is extended to arbitrary $\mathcal{SHOIQ}$ concepts according to the left part of Table 1, where $|S|$ denotes the cardinality of a set S, and to inverse roles by defining $\mathsf{Inv}(r)^{\mathcal{I}}$ as $\{(x,y) \mid (y,x) \in r^{\mathcal{I}}\}$. An interpretation $\mathcal{I}$ is said to satisfy an axiom ax or be a model of ax, if the corresponding condition given in the right part of Table 1 holds. By $\mathcal{M}(ax)$ (resp. $\mathcal{M}(S)$) we denote the set of models of an axiom ax (resp. a set S of axioms). Then $\mathcal{M}(S) = \bigcap_{ax \in S} \mathcal{M}(ax)$ for any set S of axioms. A $\mathcal{SHOIQ}$ ontology $\mathcal{O}$ is said to be consistent if $\mathcal{M}(\mathcal{O}) \neq \emptyset$. A set S of axioms is said to be entailed by $\mathcal{O}$, denoted by $\mathcal{O} \models S$, if $\mathcal{M}(\mathcal{O}) \subseteq \mathcal{M}(S)$.

Table 1: The syntax and semantics of $\mathcal{SHOIQ}$

Syntax	**Semantics**	**TBox**	**Conditions**
$\top$	$\Delta^{\mathcal{I}}$	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
$\bot$	$\emptyset$	$\mathsf{Tra}(r)$	$(r^{\mathcal{I}})^+ = r^{\mathcal{I}}$
$\{a\}$	$a^{\mathcal{I}}$	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$

$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$		
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	**ABox**	**Conditions**
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$	$r(a,b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$
$\forall r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y : (x,y) \in r^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}$	$\neg r(a,b$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin r^{\mathcal{I}}$
$\leq_n s.C$	$\{x \in \Delta^{\mathcal{I}} \mid \mid \{y \in C^{\mathcal{I}} \mid (x,y) \in s^{\mathcal{I}}\} \mid \leq n\}$	$a \approx b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
$\geq_n s.C$	$\{x \in \Delta^{\mathcal{I}} \mid \mid \{y \in C^{\mathcal{I}} \mid (x,y) \in s^{\mathcal{I}}\} \mid \geq n\}$	$a \not\approx b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$

3.2 Disjunctive Datalog and Plain Datalog

An atom is of the form $T(v_1, \ldots, v_n)$, where T is a predicate and the arguments v_1, …, v_n are variables or constants. When T is the equality predicate $\approx$, $T(v_1, v_2)$ is also called an equational atom, usually written as $v_1 \approx v_2$. A rule is of the form $\alpha_1 \vee \ldots \vee \alpha_n \leftarrow \beta_1, \ldots, \beta_m$, where α_i and β_i are atoms, α_1, …, α_n are called head atoms of the rule, and β_i, …, β_m are called body atoms of the rule. The set of head atoms of a rule R is denoted by $\mathsf{head}(R)$, while the set of body atoms of R is denoted by $\mathsf{body}(R)$. A rule R is called a constraint if $|\mathsf{head}(R)| = 0$; called a fact if $|\mathsf{body}(R)| = 0$; called definite if $|\mathsf{head}(R)| = 1$. A fact $\alpha_1 \vee \ldots \vee \alpha_n \leftarrow$ can simply be written as $\alpha_1 \vee \ldots \vee \alpha_n$. A rule is said to be safe if every variable occurring in a head atom also occur in some body atom. A disjunctive datalog program (Eiter et al., 1997) is a finite set of safe rules. A disjunctive datalog program with equality is a disjunctive datalog program in which some equational atoms occur in rule heads. A plain datalog program (with equality) is a disjunctive datalog program (with equality) that has only definite rules and constraints.

An atom or a rule is ground if it has no variables. A ground instance of an atom α (resp. a rule R) is a ground atom (resp. a ground rule) obtained from α (resp. R) by replacing all variables with constants. Given a disjunctive datalog program with equality $\mathcal{P}$, the set of all ground instances of atoms in $\mathcal{P}$ obtained by replacing all variables with constants occurring in $\mathcal{P}$ is called the Herbrand base of $\mathcal{P}$, denoted by $\mathsf{HB}(\mathcal{P})$. The set of all ground instances of rules in $\mathcal{P}$ obtained by replacing all variables with constants occurring in $\mathcal{P}$ is denoted by $\mathcal{G}(\mathcal{P})$.

A Herbrand interpretation (simply interpretation) M of $\mathcal{P}$ is a subset of $\mathsf{HB}(\mathcal{P})$. M is called a Herbrand model (simply model) of $\mathcal{P}$ if (i) $\mathsf{body}(r) \subseteq M$ implies $\mathsf{head}(r) \cap M \neq \varnothing$ for every ground rule $r \in \mathcal{G}(\mathcal{P})$, and (ii) the equality predicate $\approx$ can be interpreted as a congruence relation in M, i.e., $\approx$ is reflexive ($a \approx a \in M$ for all constants a occurring in M), symmetric ($a \approx b \in M$ implies $b \approx a \in M$) and transitive ($a \approx b \in M$ and $b \approx c \in M$ imply $a \approx c \in M$), and $T(a_1,\ldots,a_i,\ldots,a_n) \in M$ and $a_i \approx b_i \in M$ imply $T(a_1,\ldots,b_i,\ldots,a_n) \in M$ for every predicate T occurring in $\mathcal{P}$. $\mathcal{P}$ is said to be satisfiable if it admits at least one model. A ground atom α is said to be entailed by $\mathcal{P}$, denoted by $\mathcal{P} \models \alpha$, if α is in all models of $\mathcal{P}$. A set S of ground atoms is said to be entailed by $\mathcal{P}$, denoted by $\mathcal{P} \models S$, if $\mathcal{P} \models \alpha$ for all $\alpha \in S$.

3.3 Compiling from $\mathcal{SHIQ}$ to Disjunctive Datalog

The DL $\mathcal{SHIQ}$ is almost as expressive as $\mathcal{SHOIQ}$ except that nominals are disallowed. There is a well-known method (Hustadt et al., 2007) for compiling an extensionally reduced $\mathcal{SHIQ}$ ontology to a disjunctive datalog program with equality, where a $\mathcal{SHIQ}$ ontology is said to be extensionally reduced if for all concept assertions $C(a)$ in the ABox, C is a literal concept, and for all role assertions $r(a,b)$ in the ABox, r is not an inverse role or the negation of some inverse role. Since this method has been implemented in the KAON2 system (http://kaon2.semanticweb.org/), we call it the KAON2 method.

Given an extensionally reduced $\mathcal{SHIQ}$ ontology $\mathcal{O}$ whose TBox is $\mathcal{T}$ and whose ABox is $\mathcal{A}$, the KAON2 method compiles $\mathcal{O}$ to a disjunctive datalog program with equality, denoted by $\mathsf{DD}(\mathcal{O})$, through the following six steps.

In the first step, every transitivity axiom $\mathsf{Tra}(s) \in \mathcal{T}$ is removed and concept inclusion axioms of the form $\forall r.C \sqsubseteq \forall s.(\forall s.C)$ are added to $\mathcal{T}$, for all roles r

such that $s \sqsubseteq^* r$ and all concepts C appearing in $\mathcal{T}$. This step is the standard method for eliminating transitivity axioms and will yield an $\mathcal{ALCHIQ}$ ontology $\Omega(\mathcal{O})$, such that $\Omega(\mathcal{O})$ is consistent if $\mathcal{O}$ is consistent, and when $\mathcal{O}$ has no negated role assertions on complex roles, $\mathcal{O}$ is consistent if $\Omega(\mathcal{O})$ is consistent (Hustadt et al., 2007).

In the second step, the TBox of $\Omega(\mathcal{O})$ is translated into a set of first-order clauses, using standard transformation methods from first-order logic. This step involves eliminating existential quantifiers by Skolemization and may introduce function symbols.

In the third step, the set of clauses obtained in the second step is saturated by adding non-redundant logical consequences. This step takes up to exponential time w.r.t. the size of $\mathcal{T}$. For an arbitrary atom (possibly an equational atom) in the saturated set of non-redundant clauses, its arguments can be variables or functional terms of the form $f(x)$, where f is a function symbol introduced in the second step.

In the fourth step, any functional term $f(x)$ occurring in the resulting set of clauses in the third step is rewritten to a new variable x_f The resulting set of clauses is then syntactically transformed to a set of rules. To make the resulting rules safe, auxiliary atoms of the form $HU(x)$, $HU(x_f)$ or $S_f(x, x_f)$ are added to rule bodies if necessary. For example, the rule $B(x_f) \leftarrow A(x), S_f(x, x_f)$ is rewritten from $B(f(x)) \leftarrow A(x)$, while the rule $A(x) \vee B(x) \leftarrow HU(x)$ is rewritten from $A(x) \vee B(x)$. We denote the set of rules computed in this step by $\Gamma(\mathcal{T})$, which has no functional terms.

In the fifth step, a set of ground facts of the form $HU(a)$, $HU(a_f)$ or $S_f(a, a_f)$ is constructed, which are instantiated for all individual names a occurring in $\mathcal{A}$ and all function symbols f introduced in the second step. We denote this set by $\Delta(\mathcal{O})$.

In the last step, $\mathcal{A}$ is directly translated to a set of ground facts or ground constraints. More precisely, ABox axioms of the form $A(a)$ (resp. $r(a,b)$ or $a \approx b$) are translated to ground facts $A(a)$ (resp. $r(a,b)$ or $a \approx b$), while ABox axioms of the form $\neg A(a)$ (resp. $\neg r(a,b)$ or $a \approx b$) are translated to ground constraints $\leftarrow A(a)$ (resp. $\leftarrow r(a,b)$ or $\leftarrow a \approx b$). We denote this set by $\Xi(\mathcal{A})$.

Let $\mathsf{DD}(\mathcal{O})$ be defined as $\Gamma(\mathcal{T}) \cup \Xi(\mathcal{A}) \cup \Delta(\mathcal{O})$. We have the following theorem.

Theorem 1 ((Hustadt et al., 2007)). Let $\mathcal{O}$ be an extensionally reduced $\mathcal{SHIQ}$ ontology. Then: (1) for any literal concept assertion or literal role assertion ax, $\mathsf{DD}(\mathcal{O} \cup \{ax\}) = \mathsf{DD}(\mathcal{O}) \cup \{ax\}$; (2) when $\mathcal{O}$ has no negated role assertions on complex roles, $\mathcal{O}$ is consistent if and only if $\mathsf{DD}(\mathcal{O})$ is satisfiable.

In the following, an example for the KAON2 method is shown.

Example 2 Consider the ontology $\mathcal{O}$ given in Example 1. By compiling $\mathcal{O}$ through the KAON2 system, we obtain $\mathsf{DD}(\mathcal{O})$ which consists of the following rules (1)–(6).

$$\mathsf{Extraordinary}(x) \leftarrow \mathsf{Clever}(x), \mathsf{Diligent}(x) \quad (1)$$

$$\mathsf{Extraordinary}(x) \leftarrow \mathsf{isRewarded}(x, y), \mathsf{Competition}(y) \quad (2)$$

$$\mathsf{Person}(x) \leftarrow \mathsf{Extraordinary}(x) \quad (3)$$

$$\mathsf{Extraordinary}(x) \vee \mathsf{Ordinary}(x) \leftarrow \mathsf{Person}(x) \quad (4)$$

$$\mathsf{Person}(\mathsf{Tom}) \leftarrow \quad (5)$$

$$\mathsf{Clever}(\mathsf{Tom}) \leftarrow \quad (6)$$

3.4 Equality Axiomatization

Our proposed method needs to call a Prolog engine to solve the reduced abduction problem in which the background theory is a plain datalog program. Since equational atoms occurring in rule heads have special semantics and existing Prolog engines do not particularly handle this semantics, we need to treat the equality predicate as an ordinary predicate through a standard method for axiomatizing equality (Fitting, 1996). This method is described below.

Let $\pi(\mathcal{P})$ denote the disjunctive datalog program obtained from a disjunctive datalog program with equality $\mathcal{P}$ by replacing the equality predicate $\approx$ with a new

ordinary predicate eq, and $\mathcal{P}_{\approx}$ denote the plain datalog program consisting of the following rules.

$$eq(a,a) \leftarrow . \text{ for every constant } a \text{ occurring in } \mathcal{P} \tag{7}$$

$$eq(y,x) \leftarrow eq(x,y). \tag{8}$$

$$eq(x,z) \leftarrow eq(x,y), eq(y,z). \tag{9}$$

$$T(x_1,\ldots,y_i,\ldots,x_n) \leftarrow T(x_1,\ldots,x_i,\ldots,x_n), eq(x_i,y_i). \text{ for every predicate } T \text{ occurring in } \mathcal{P} \text{ except } \approx \text{ and every position } i \text{ in } T \tag{10}$$

The group of rules (7) ensures that eq is reflexive. Rule (8) ensures that eq is symmetric. Rule (9) ensures that eq is transitive. The group of rules (10) ensures that for every model M of $\pi(\mathcal{P})$ and every predicate T occurring in $\mathcal{P}$ except $\approx$, $T(a_1,\ldots,a_i,\ldots,a_n) \in M$ and $eq(a_i,b_i) \in M$ imply $T(a_1,\ldots,b_i,\ldots,a_n) \in M$. It is clear that M is a model of $\mathcal{P}$ if and only if M is an interpretation of $\pi(\mathcal{P}) \cup \mathcal{P}_{\approx}$ such that $\mathsf{body}(r) \subseteq M$ implies $\mathsf{head}(r) \cap M \neq \emptyset$ for all rules $r \in \mathcal{G}(\pi(\mathcal{P}) \cup \mathcal{P}_{\approx})$. The disjunctive datalog program without equality $\pi(\mathcal{P}) \cup \mathcal{P}_{\approx}$, in which the equality predicate does not appear, is said to be obtained from $\mathcal{P}$ by axiomatizing equality.

4. A New Problem for ABox Abduction

We derive a new problem for ABox abduction from the area of logic-based abduction (Eiter & Gottlob, 1995; Kakas et al., 1998). In this area, an abduction problem is usually defined as a problem of computing all minimal sets Δ of sentences w.r.t. a background theory T and an observation G, such that Δ does not entail G, but $T \cup \Delta$ entails G and $T \cup \Delta$ is consistent. A computed set Δ is often restricted to a special pre-specified class of sentences called abducibles, so as to provide appropriate modes to enforce entailment of an observation (Kakas et al., 1998). Inspired from this idea, we propose the following problem for ABox abduction, in which abducible predicates are introduced to give users flexibility to formulate the explanations for an observation.

Definition 1 (ABox Abduction). Given a DL ontology $\mathcal{O}$, a finite set A of abducible predicates which are arbitrary concepts or roles, and an observation G which is a finite set of concept or role assertions, an abductive solution for $(\mathcal{O}, A, G)$ is a subset-minimal (simply minimal) set Δ of ABox axioms such that

all ABox axioms in Δ are directly composed of individual names in $\mathcal{O}$ and concepts or roles in A, $\Delta \not\models G$, $\mathcal{O} \cup \Delta \models G$ and $\mathcal{O} \cup \Delta$ is consistent. The ABox abduction problem defined by $(\mathcal{O}, A, G)$ is to compute all abductive solutions for $(\mathcal{O}, A, G)$.

A simple example of the proposed problem is shown below.

Example 3. Consider the ontology $\mathcal{O}$ given in Example 1. Let the set of abducible predicates be $A = \{\mathsf{Clever}, \mathsf{Diligent}, \mathsf{Extraordinary}\}$ and the observation be $G = \{\mathsf{Extraordinary}(\mathsf{Tom})\}$. Then there is only one abductive solution for $(\mathcal{O}, A, G)$, namely $\{\mathsf{Diligent}(\mathsf{Tom})\}$. $\{\mathsf{Extraordinary}(\mathsf{Tom})\}$ is not an abductive solution because $\{\mathsf{Extraordinary}(\mathsf{Tom})\} \not\models G$. $\{\mathsf{Clever}(\mathsf{Tom}), \mathsf{Diligent}(\mathsf{Tom})\}$ is not either because it is not minimal.

The proposed problem (simply called problem A) mainly differs from the problem proposed in (Klarman et al., 2011) (simply called problem B) in using a finite set of abducible predicates. In problem B, abductive solutions can be on an infinite set of $\mathcal{ALE}$ concepts or roles, where $\mathcal{ALE}$ is a DL obtained from $\mathcal{ALC}$ by disallowing non-atomic negation and disjunction. In problem A, abductive solutions can be on more expressive DL concepts. The current problem A has extended its original one proposed in the conference paper (Du et al., 2011) by allowing arbitrary concepts or roles as abducible predicates and negated role assertions as observations.

Problem A may be inferior to problem B when abducible predicates are not appropriately set. On the one hand, when an instance of problem B has abductive solutions, its counterpart of problem A may not have. For example, given $\mathcal{O} = \{\forall \mathsf{hasChild}.\mathsf{Good} \sqsubseteq \mathsf{Happy}\}$ and $G = \{\mathsf{Happy}(\mathsf{Amy})\}$, the corresponding instance of problem B has an abductive solution in $\mathcal{O}$, namely $\{\forall \mathsf{hasChild}.\mathsf{Good}(\mathsf{Amy})\}$, but its counterpart of problem A has not unless $\forall \mathsf{hasChild}.\mathsf{Good}$ is set as an abductive predicate. On the other hand, the representation of abductive solutions in problem A may be less concise than its counterpart in problem B. For example, given $\mathcal{O} = \{\exists \mathsf{hasFather}.\top \sqsubseteq \mathsf{Person}$, $\neg \mathsf{hasFather}(a_1, a_1)$, $\mathsf{Person}(a_2)$, ..., $\mathsf{Person}(a_n)\}$, $A = \{\mathsf{hasFather}\}$ and $G = \{\mathsf{Person}(a_1)\}$, the corresponding instance of problem A has $n-1$ abductive solutions $\{\mathsf{hasFather}(a_1, a_2)\}$, ..., $\{\mathsf{hasFather}(a_1, a_n)\}$ in $\mathcal{O}$, while its counterpart of problem B has only one, i.e. $\{\exists \mathsf{hasFather}.\top (a_1)\}$.

Despite of the above potential disadvantages, problem A has intrinsic merits that are lacking in problem B. First, the number of abductive solutions is finite because the

number of possible axioms in an abductive solution is at most $|A_c||N_I|+|A_r||N_I|^2$, where A_c, A_r and N_I are respectively the set of concepts in A, the set of roles in A, and the set of individual names in $\mathcal{O}$. Second, the minimality of candidate abductive solutions can be simply determined by set-inclusion checking, rather than by the complex renaming and entailment checking which are used in the method for problem B (Klarman et al., 2011). Since termination and efficiency are crucial for practical ABox abduction, we propose problem A as the fundamental problem for practical ABox abduction.

5. Computing All Abductive Solutions

Although the number of abductive solutions is finite, a brute-force search method for computing all abductive solutions is impractical because the search space, namely the set of candidate solutions, has a size exponential in $|A_c||N_I|+|A_r||N_I|^2$. Hence we consider state-of-the-art abduction systems, such as CIFF (Mancarella et al., 2009) and $\mathcal{A}$-system (Kakas et al., 2001). These systems compute minimal results in a top-down manner, recursively using goals to direct the search and prune search space. To adapt the top-down manner to computing abductive solutions in ABox abduction, we need to confine the background ontology as a syntactic variant of a plain datalog program, because existing practical methods for computing minimal results, such as the ones implemented in CIFF and $\mathcal{A}$-system, only work on plain datalog programs with negation-as-failure. To make the adaptation work for common DLs (e.g. $\mathcal{SHOIQ}$) that cannot be translated to plain datalog, we consider approximate translations which are derived from the KAON2 method (Hustadt et al., 2007). As described in subsection 2.3, the KAON2 method compiles an extensionally reduced $\mathcal{SHIQ}$ ontology $\mathcal{O}$ to a disjunctive datalog program with equality $\mathsf{DD}(\mathcal{O})$.

Let $\mathcal{P}$ be a disjunctive datalog program (possibly with equality), A be a set of atomic concepts or atomic roles, and G be a set of atomic concept assertions or atomic role assertions. Corresponding to Definition 1, we also define an abductive solution for $(\mathcal{P}, A, G)$ as a minimal set of Δ of ground atoms such that all ground atoms in Δ are directly composed of constants in $\mathcal{P}$ and predicates in A, $\Delta \not\models G$, $\mathcal{P} \cup \Delta \models G$ and $\mathcal{P} \cup \Delta$ is consistent, where atomic concepts and atomic roles are treated as predicates, and atomic concept assertions and atomic role assertions are treated as ground atoms. By Theorem 1, we have a correspondence between abductive solutions for $(\mathsf{DD}(\mathcal{O}), A, G)$ and abductive solutions for $(\mathcal{O}, A, G)$, as shown in Theorem 2. Note that an abductive solution for $(\mathsf{DD}(\mathcal{O}), A, G)$ may contain constants not corresponding to individual names in $\mathcal{O}$, i.e. the constants of the form a_f introduced in the fifth step of the KAON2 method (see subsection

2.3), thus Theorem 2 only considers abductive solutions for $(\mathsf{DD}(\mathcal{O}), A, G)$ that are ABox axioms; this means that all constants appearing in these abductive solutions correspond to individual names in $\mathcal{O}$.

Theorem 2. Let $\mathcal{O}$ be an extensionally reduced $\mathcal{SHIQ}$ ontology without negated role assertions on complex roles, A a set of atomic concepts or atomic roles, and G a set of atomic concept assertions or atomic role assertions on simple roles, then for any set Δ of ABox axioms, Δ is an abductive solution for $(\mathsf{DD}(\mathcal{O}), A, G)$ if and only if it is an abductive solution for $(\mathcal{O}, A, G)$.

Proof. We show that (*) for any set Δ of ABox axioms on concepts or roles in A, $\mathsf{DD}(\mathcal{O}) \cup \Delta$ is satisfiable and entails $G \Leftrightarrow \mathcal{O} \cup \Delta$ is consistent and entails G. ($\Leftarrow$) Since $\mathcal{O} \cup \Delta$ is consistent, $\mathsf{DD}(\mathcal{O} \cup \Delta)$ is satisfiable, so $\mathsf{DD}(\mathcal{O}) \cup \Delta = \mathsf{DD}(\mathcal{O} \cup \Delta)$ is also satisfiable. Let ax be any ABox axiom in G. Since $\mathcal{O} \cup \Delta \models \{ax\}$, $\mathcal{O} \cup \Delta \cup \{\neg ax\}$ is inconsistent. Since $\mathcal{O} \cup \Delta \cup \{\neg ax\}$ does not contain any negated role assertion on complex roles, $\mathsf{DD}(\mathcal{O} \cup \Delta \cup \{\neg ax\})$ is unsatisfiable. Hence, $\mathsf{DD}(\mathcal{O} \cup \Delta) \cup \{\neg ax\} = \mathsf{DD}(\mathcal{O} \cup \Delta \cup \{\neg ax\})$ is also unsatisfiable, and thus $\mathsf{DD}(\mathcal{O} \cup \Delta) \models \{ax\}$. It follows that $\mathsf{DD}(\mathcal{O} \cup \Delta) \models G$. ($\Rightarrow$) Since $\mathsf{DD}(\mathcal{O}) \cup \Delta$ is satisfiable, $\mathsf{DD}(\mathcal{O} \cup \Delta) = \mathsf{DD}(\mathcal{O}) \cup \Delta$ is also satisfiable, so $\mathcal{O} \cup \Delta$ is satisfiable. Let ax be any ABox axiom in G. Since $\mathsf{DD}(\mathcal{O}) \cup \Delta \models \{ax\}$, $\mathsf{DD}(\mathcal{O}) \cup \Delta \cup \{\neg ax\}$ is unsatisfiable, so $\mathsf{DD}(\mathcal{O} \cup \Delta \cup \{\neg ax\}) = \mathsf{DD}(\mathcal{O}) \cup \Delta \cup \{\neg ax\}$ is also unsatisfiable. Since $\mathcal{O} \cup \Delta \cup \{\neg ax\}$ does not contain any negated role assertion on complex roles, $\mathcal{O} \cup \Delta \cup \{\neg ax\}$ is inconsistent, so $\mathcal{O} \cup \Delta \models \{ax\}$. It follows that $\mathcal{O} \cup \Delta \models G$.

(1) Let Δ be an abductive solution for $(\mathcal{O}, A, G)$, then $\Delta \not\models G$ and $\mathcal{O} \cup \Delta$ is consistent and entails G. By (*), $\mathsf{DD}(\mathcal{O}) \cup \Delta$ is consistent and entails G. Suppose Δ is not an abductive solution for $(\mathsf{DD}(\mathcal{O}), A, G)$, then there is an abductive solution Δ' for $(\mathsf{DD}(\mathcal{O}), A, G)$ such that $\Delta' \subset \Delta$. By (*) again, $\mathcal{O} \cup \Delta'$ is consistent and entails G. Moreover, $\Delta' \not\models G$ and all concepts or roles occurring in Δ' are in A, contradicting that Δ is an abductive solution for $(\mathcal{O}, A, G)$.

(2) Let Δ be a set of ABox axioms and an abductive solution for $(\mathsf{DD}(\mathcal{O}), A, G)$, then $\Delta \not\models G$ and $\mathsf{DD}(\mathcal{O}) \cup \Delta$ is consistent and entails G. By (*), $\mathcal{O} \cup \Delta$ is consistent and entails G. Suppose Δ is not an abductive solution for $(\mathcal{O}, A, G)$,

then there is an abductive solution Δ' for $(\mathcal{O}, A, G)$ such that $\Delta' \subset \Delta$. By (*) again, $\mathsf{DD}(\mathcal{O}) \cup \Delta'$ is consistent and entails G. Moreover, $\Delta' \not\models G$ and all concepts or roles occurring in Δ' are in A, contradicting that Δ is an abductive solution for $(\mathsf{DD}(\mathcal{O}), A, G)$. □

The above theorem shows that for some restricted class of the proposed problem, the original problem can be reduced to the problem of computing all abductive solutions in the reduced disjunctive datalog program with equality. Hence, we first propose a method for this restricted class, and then extend it to address the full class of the proposed problem.

5.1. The Method for the Restricted Class

Throughout this subsection, let $\mathcal{O}$ denote an extensionally reduced $\mathcal{SHIQ}$ ontology without negated role assertions on complex roles, A a set of atomic concepts or atomic roles, and G a set of atomic concept assertions or atomic role assertions on simple roles. In order to compute abductive solutions for $(\mathcal{O}, A, G)$, by Theorem 2 we seek methods for computing abductive solutions for $(\mathsf{DD}(\mathcal{O}), A, G)$.

Considering that two state-of-the-art abduction systems CIFF (Mancarella et al., 2009) and $\mathcal{A}$-system (Kakas et al., 2001) are built on Prolog engines, we intend to encode the problem of computing all abductive solutions for $(\mathsf{DD}(\mathcal{O}), A, G)$ into a Prolog program and solve it with Prolog engines. Note that we do not directly apply CIFF or $\mathcal{A}$-system to solve the abduction problem on $(\mathsf{DD}(\mathcal{O}), A, G)$, because currently CIFF and $\mathcal{A}$-system cannot guarantee termination in handling cyclic logic programs, whereas $\mathsf{DD}(\mathcal{O})$ can have cycles between predicates (e.g., the disjunctive datalog program given in Example 2 has a cycle on predicates Person and $\mathsf{Extraordinary}$). Hence, we turn to encode the abduction problem on $(\mathsf{DD}(\mathcal{O}), A, G)$ into a Prolog program and apply a state-of-the-art Prolog engine, B-Prolog (http://www.probp.com/), to solve it. B-Prolog supports linear tabling (Shen et al., 2001), which is an efficient way to guarantee termination in handling cycles.

The equality predicate $\approx$ should be axiomatized when it occurs in some rule heads in $\mathsf{DD}(\mathcal{O})$, because B-Prolog does not treat it as a congruence relation. As described in subsection 2.4, the equality predicate $\approx$ can be axiomatized by using a standard method (Fitting, 1996). Let $\mathsf{DD}'(\mathcal{O})$ be obtained from $\mathsf{DD}(\mathcal{O})$ by axiomatizing equality if necessary. That is, $\mathsf{DD}'(\mathcal{O})$ is converted from $\mathsf{DD}(\mathcal{O})$ using the method described in subsection 2.4, if the equality predicate $\approx$ occurs in some rule heads in $\mathsf{DD}(\mathcal{O})$, or is directly copied from $\mathsf{DD}(\mathcal{O})$ otherwise. Consider

Example 2. Since the equality predicate does not occurs in any rule head in $\mathsf{DD}(\mathcal{O})$, $\mathsf{DD}'(\mathcal{O})$ is the same as $\mathsf{DD}(\mathcal{O})$.

In the following, we present a method for encoding $(\mathsf{DD}'(\mathcal{O}), A, G)$ into a Prolog program, which consists of four steps.

In the first step, the observation G is encoded into a Prolog rule with a nullary head atom go, where every ground atom $\alpha \in G$ is encoded as a body atom with an extra argument, which is a list storing a set of ground atoms that are added to $\mathsf{DD}'(\mathcal{O})$ to enforce entailment of α. A list L is of the form $[t_1,...,t_n]$, where t_i is of the form $(a,\text{"rdf:type"},\text{p\_A})$ or $(a,\text{p\_r},b)$. It can be decoded into a set of ABox axioms $\{t'_1,...,t'_n\}$, denoted by $\mathsf{decode}(L)$, where t'_i is rewritten from t_i by rewriting $(a,\text{"rdf:type"},\text{p\_T})$ to a concept assertion $T(a)$ and $(a,\text{p\_r},b)$ to a role assertion $r(a,b)$. Note that every predicate in $\mathsf{DD}'(\mathcal{O})$ is rewritten to a Prolog predicate with the prefix "p_", because the Prolog syntax capitalizes the first letter for variables only. Following the first body atom encoded from ground atoms in G, a body atom of the form $check(L)$ is added to the encoded Prolog rule, where $check(L)$ returns true if and only if none of the subsets of L has been output, which is used to prune non-minimal sets of added ground atoms. Then, following every other body atom encoded from ground atoms in G, two body atoms of the form $union(L_1,L_2,L)$ and $check(L)$ are added to the encoded Prolog rule, where $union(L_1,L_2,L)$ sets L as the union of L_1 and L_2 and returns true, which is used to yield the union of all sets of added ground atoms. Following all the above body atoms, two body atoms $output(L)$ and $fail$ are also added to the encoded Prolog rule, where $output(L)$ outputs L and returns true, while $fail$ forces the Prolog engine to enumerate all possible instantiations of extra arguments when go is called. For example, the observation $\{\mathsf{Extraordinary(Tom)}, \mathsf{Person(Tom)}\}$ is encoded into the following Prolog rule.

$$go \text{ :- } \text{p\_Extraordinary}(\text{"Tom"},L_1), check(L_1),$$
$$\text{p\_Person}(\text{"Tom"},L_2), union(L_1,L_2,L), check(L), output(L), fail$$

.

In the second step, every definite rule in $\mathsf{DD}'(\mathcal{O})$ is encoded into a Prolog rule. In more details, every atom α occurring in $\mathsf{DD}'(\mathcal{O})$ is encoded into a Prolog atom with an extra argument, which is a list storing a set of ground atoms that are added to

$\mathsf{DD}'(\mathcal{O})$ to enforce entailment of a ground instance of α. When α has variables, the extra argument is written as a variable because the corresponding list is different for different ground instances of α, otherwise the extra argument is written as the empty list $[\,]$ since $\alpha \leftarrow$ is a ground fact in $\mathsf{DD}'(\mathcal{O})$ and α is entailed by $\mathsf{DD}'(\mathcal{O})$. Likewise, following the first body atom (resp. every other body atom) encoded from body atoms in the original rule in $\mathsf{DD}'(\mathcal{O})$, a body atom of the form $check(L)$ (resp. two body atoms of the form $union(L_1, L_2, L)$ and $check(L)$) is added to the encoded Prolog rule. Consider Example 2, all rules except rule (4) are encoded in this step. For example, rules (2), (3) and (6) are encoded into the following Prolog rules.

$$\text{p\_Extraordinary}(X, L) \text{ :- p\_isRewarded}(X, Y, L_1), check(L_1),$$
$$\text{p\_Competition}(Y, L_2), union(L_1, L_2, L), check(L)$$
.

$$\text{p\_Person}(X, L) \text{ :- p\_Extraordinary}(X, L), check(L).$$

$$\text{p\_Clever("Tom",[\,])}.$$

In the third step, every predicate T in A is encoded into a Prolog rule, which consists of a head atom and n body atoms, specifying that adding a ground atom to $\mathsf{DD}'(\mathcal{O})$ enforces entailment of this ground atom, where $n = 1$ if T is an atomic concept, or $n = 2$ if T is an atomic role. The head atom is composed by T and $n+1$ arguments, where the last argument is a singleton list storing an atom on T. Every body atom is of the form $dom(X)$, which ensures X to be a constant in $\mathsf{DD}'(\mathcal{O})$. For example, the abducible predicates Diligent and isRewarded are encoded into the following two rules.

$$\text{p\_Diligent}(X,[(X,\text{"rdf:type"},\text{"Diligent"})]) \text{ :- } dom(X).$$

$$\text{p\_isRewarded}(X, Y, [(X,\text{"isRewarded"}, Y)]) \text{ :- } dom(X),\ dom(Y).$$

In the last step, all Prolog predicates occurring in cycles in the set of generated Prolog rules are declared to be tabled predicates. Declaring a Prolog predicate to be tabled means that any Prolog atom on this predicate is prevented from calling multiple times. The declaration of tabled predicates is supported by B-Prolog. This is a crucial step for guaranteeing termination when calling Prolog atoms in the encoded Prolog program.

There are two remarks on the aforementioned encoding method. First, rules in $\mathsf{DD}'(\mathcal{O})$ that have more than one head atom cannot be encoded into Prolog rules, thus they are ignored. Second, although constraints in $\mathsf{DD}'(\mathcal{O})$ can be encoded into Prolog rules with some special treatments, they are only used to determine whether an output solution is consistent with the background theory. Since this consistency checking implemented in B-Prolog is based on brute-force search and is generally less efficient than consistency checking in modern DL reasoners, constraints in $\mathsf{DD}'(\mathcal{O})$ are also ignored and consistency checking is performed by calling external DL reasoners.

Let $\mathsf{prolog}(\mathsf{DD}'(\mathcal{O}), A, G)$ denote the set of lists output by the Prolog program encoded from $(\mathsf{DD}'(\mathcal{O}), A, G)$ when calling go. The following theorem shows that all abductive solutions for $(\mathsf{DD}(\mathcal{O}), A, G)$ can be extracted from $\mathsf{prolog}(\mathsf{DD}'(\mathcal{O}), A, G)$ when $\mathsf{DD}'(\mathcal{O})$ is a plain datalog program.

Theorem 3. If $\mathsf{DD}(\mathcal{O})$ is a plain datalog program (possibly with equality), then the set of abductive solutions for $(\mathsf{DD}(\mathcal{O}), A, G)$ is the set of minimal sets in $\{\mathsf{decode}(L) \mid L \in \mathsf{prolog}(\mathsf{DD}'(\mathcal{O}), A, G), \mathsf{decode}(L) \not\models G, \mathcal{O} \cup \mathsf{decode}(L)$ is consistent$\}$.

Proof. Note that $\mathsf{DD}'(\mathcal{O})$ is a plain datalog program without equality. The encoded Prolog program searches and only outputs all lists L such that $\mathsf{DD}'(\mathcal{O}) \cup \mathsf{decode}(L)$ entails G and every ground atom in $\mathsf{decode}(L)$ is on atomic concepts or atomic roles in A. It must output all minimal ones among all these lists. Let $S = \{\mathsf{decode}(L) \mid L \in \mathsf{prolog}(\mathsf{DD}'(\mathcal{O}), A, G), \mathsf{decode}(L) \not\models G, \mathcal{O} \cup \mathsf{decode}(L)$ is consistent$\}$.

(1) Let Δ be a minimal set in S, then Δ is a set of ground atoms on atomic concepts or atomic roles in A such that $\mathsf{DD}'(\mathcal{O}) \cup \Delta \models G$, i.e., $\mathsf{DD}(\mathcal{O}) \cup \Delta \models G$. Since $\mathcal{O} \cup \Delta$ is consistent, by Theorem 1, $\mathsf{DD}(\mathcal{O}) \cup \Delta = \mathsf{DD}(\mathcal{O} \cup \Delta)$ is satisfiable. Hence, Δ is an abductive solution for $(\mathsf{DD}(\mathcal{O}), A, G)$.

(2) Let Δ be an abductive solution for $(\mathsf{DD}(\mathcal{O}), A, G)$, then $\mathsf{DD}'(\mathcal{O}) \cup \Delta \models G$ and there is not any proper subset Δ' of Δ such that $\mathsf{DD}'(\mathcal{O}) \cup \Delta' \models G$. Hence, there is a list L output by the encoded Prolog program such that $\mathsf{decode}(L) = \Delta$. By Theorem 2, Δ is also an abductive solution for $(\mathcal{O}, A, G)$, thus $\mathcal{O} \cup \Delta$ is consistent. Suppose Δ is not a minimal set in S. Since

$\Delta \not\models G$ and $\mathcal{O} \cup \Delta$ is consistent, there must be a minimal set Δ' in S such that $\Delta' \subset \Delta$. But then by (1), Δ' is an abductive solution for $(\mathsf{DD}(\mathcal{O}), A, G)$, contradiction. □

Based on Theorem 2 and Theorem 3, we can obtain a method which computes the set of all minimal sets in $\{\mathsf{decode}(L) \mid L \in \mathsf{prolog}(\mathsf{DD}'(\mathcal{O}), A, G), \mathsf{decode}(L) \not\models G, \mathcal{O} \cup \mathsf{decode}(L)$ is consistent$\}$ that are also sets of ABox axioms. The resulting set is actually the complete set of abductive solutions. However, this method only guarantees sound and complete results for a restricted class of the proposed problem. Moreover, it is impractical when $\mathsf{DD}(\mathcal{O})$ has equational head atoms. In this case, axiomatizing equality is needed, implying that $\mathsf{DD}'(\mathcal{O})$ will have some rules like $T(y) \leftarrow T(x), x \approx y$. These rules are hard to handle by the encoded Prolog program since every predicate occurring in them appear in cycles. Our experimental results also confirm that these rules easily make ABox abduction fail. In the next subsection, we propose a general method to tackle all the above issues.

5.2. The Method for the Full Class

Throughout this subsection, let $\mathcal{O}$ denote an arbitrary $\mathcal{SHOIQ}$ ontology, A a finite set of arbitrary concepts or roles, and G a finite set of concept assertions or role assertions. That is, $(\mathcal{O}, A, G)$ represents the full class of the proposed problem where $\mathcal{SHOIQ}$ is treated as the most expressive DL.

The key idea for applying Prolog engines to compute abductive solutions for $(\mathcal{O}, A, G)$ is to transform $(\mathcal{O}, A, G)$ to some $(\mathcal{P}, A', G')$ which can be encoded to a Prolog program using the method described in the previous subsection, i.e., $\mathcal{P}$ is a plain datalog program, A' is a set of atomic concepts or atomic roles, and G' is a set of atomic concept assertions or atomic role assertions. Suppose there is a one-to-one mapping function f that maps concepts or roles in A' to concepts or roles in A. Given a set Δ of ABox axioms on concepts or roles in A', by $f(\Delta)$ we simply denote the set of ABox axioms obtained from Δ by replacing every concept or role T occurring in Δ with $f(T)$. Suppose $\mathcal{P} \cup \Delta \models G'$ implies $\mathcal{O} \cup f(\Delta) \models G$ for all sets Δ of ABox axioms on concepts or roles in A', then for every set Δ of ABox axioms on concepts or roles in A' such that $\mathcal{P} \cup \Delta \models G'$, some subsets of $f(\Delta)$ can possibly be abductive solutions for $(\mathcal{O}, A, G)$. Hence, we can first compute all sets Δ of ABox axioms on concepts or roles in A' such that $\mathcal{P} \cup \Delta \models G'$ by the Prolog program encoded from $(\mathcal{P}, A', G')$, then extract abductive solutions for $(\mathcal{O}, A, G)$ from $f(\Delta)$.

Based on the above idea, we develop a method for computing abductive solutions for $(\mathcal{O}, A, G)$. In order to transform $(\mathcal{O}, A, G)$ to $(\mathcal{P}, A', G')$ such that (*) $\mathcal{P} \cup \Delta \models G'$ implies $\mathcal{O} \cup f(\Delta) \models G$ for all sets Δ of ABox axioms on concepts or roles in A', we need to normalize $(\mathcal{O}, A, G)$ to $(\mathcal{O}', A', G')$ first, where $\mathcal{O}'$ is an extensionally reduced ontology, A' is a set of atomic concepts or atomic roles, and G' is a set of atomic concept assertions or atomic role assertions. Since this is a normalization step, we also need to ensure that for all sets Δ of ABox axioms on concepts or roles in A', Δ is an abductive solution for $(\mathcal{O}', A', G')$ if and only if $f(\Delta)$ is an abductive solution for $(\mathcal{O}, A, G)$, where f is a one-to-one mapping function from concepts or roles in A' to concepts or roles in A. Then, in order to convert $(\mathcal{O}', A', G')$ to $(\mathcal{P}, A', G')$, we consider existing methods for transforming DLs to plain datalog. The KAON2 method is the best choice because it has efficient implementation and preserves consequences when compiling very expressive DLs to disjunctive datalog. To apply the KAON2 method, we need to weaken $\mathcal{O}'$ to a $\mathcal{SHIQ}$ ontology. Moreover, we require that the disjunctive datalog program compiled by the KAON2 method should have no equational head atoms; otherwise the axiomatization of equality is needed and will introduce many cyclic rules that heavily impair the efficiency of subsequent steps. Hence, we weaken $\mathcal{O}'$ to $\mathcal{O}''$ such that $\mathsf{DD}(\mathcal{O}'')$ can be computed by the KAON2 method and has no equational head atoms. As a weakening step, we need to ensure that $\mathcal{O}'' \cup \Delta \models G'$ implies $\mathcal{O}' \cup \Delta \models G'$ for all sets Δ of ABox axioms on concepts or roles in A'. Afterwards, we compile $\mathsf{DD}(\mathcal{O}'')$ from $\mathcal{O}''$ and then modify it to a plain datalog program $\mathcal{P}$ by removing non-definite rules and adding more definite rules to make the ultimate results more complete. To achieve the aforementioned condition (*), we need to ensure that $\mathcal{P} \cup \Delta \models G'$ implies $\mathcal{O}' \cup \Delta \models G'$ for all sets Δ of ABox axioms on concepts or roles in A'.

To summarize, the proposed method for computing all abductive solutions for $(\mathcal{O}, A, G)$ consists of five steps. In the first step, $(\mathcal{O}, A, G)$ is normalized to $(\mathcal{O}', A', G')$. In the second step, $\mathcal{O}'$ is weakened to $\mathcal{O}''$. In the third step, $\mathcal{O}''$ is compiled to $\mathsf{DD}(\mathcal{O}'')$ and $\mathsf{DD}(\mathcal{O}'')$ is then modified to $\mathcal{P}$. In the fourth step, a Prolog program is encoded from $(\mathcal{P}, A', G')$ using the method described in the previous subsection, and then go is called. In the last step, abductive solutions for $(\mathcal{O}, A, G)$ are extracted from $\mathsf{decode}(L)$ for every list L output by the encoded Prolog program. More details on these steps are given in the following.

5.2.1 Normalizing $(\mathcal{O}, A, G)$

In the first step, we need to normalize $(\mathcal{O}, A, G)$ to $(\mathcal{O}', A', G')$ such that for all sets Δ of ABox axioms on concepts or roles in A', Δ is an abductive solution for $(\mathcal{O}', A', G')$ if and only if $f(\Delta)$ is an abductive solution for $(\mathcal{O}, A, G)$, where f is a one-to-one mapping function from concepts or roles in A' to concepts or roles in A. To achieve this goal, $\mathcal{O}'$ may not be kept as a $\mathcal{SHOIQ}$ ontology. For example, suppose $\neg r$ is a role in A and $\neg s(a,b)$ is a role assertion in G, where both r and s are atomic roles. To obtain A' and G', we introduce a fresh atomic role $Q^{\dagger}_{\neg r}$ for $\neg r$ and another fresh atomic role $Q^{\|}_{\neg s}$ for $\neg s$. To make G' hold, we need to guarantee the traditional forward inference from ABox axioms on A' to G'. This inference involves an inference from ABox axioms on A' to ABox axioms on A, an inference from ABox axioms on A to G, and an inference from G to G'. Hence, we need to introduce two axioms $Q^{\dagger}_{\neg r} \sqsubseteq \neg r$ and $\neg s \sqsubseteq Q^{\|}_{\neg s}$ for $Q^{\dagger}_{\neg r}$ and $Q^{\|}_{\neg s}$, respectively, to make the inference from ABox axioms on A' to G' work.

We call an axiom of the form $s \sqsubseteq \neg r$ or $\neg s \sqsubseteq r$ (where s and r are non-negated roles) a negated role inclusion axiom. A $\mathcal{SHOIQ}$ ontology does not include negated role inclusion axioms, thus $\mathcal{O}'$ may not be expressed in $\mathcal{SHOIQ}$. We extend the semantics of $\mathcal{SHOIQ}$ to the semantics of $\mathcal{SHOIQ}$ with negated role inclusion axioms. We say an interpretation $\mathcal{I}$ satisfies $s \sqsubseteq \neg r$ if $s^{\mathcal{I}} \cap r^{\mathcal{I}} = \varnothing$; satisfies $\neg s \sqsubseteq r$ if $s^{\mathcal{I}} \cup r^{\mathcal{I}} = \Delta^{\mathcal{I}}$. Then a model of $\mathcal{O}'$ is still defined as an interpretation that satisfies all axioms in $\mathcal{O}'$.

The pseudo-code for this step is given in Algorithm 1 below. Line 1 eliminates all inverse roles in $\mathcal{O}$ and G. Lines 3–16 normalize A to A' by introducing a set of fresh predicates (which are atomic concepts or roles) and adding axioms that maintain the correspondence between fresh predicates and original predicates. Lines 17–24 normalize G to G' in a similar way as normalizing A. Lines 25–28 compute $\mathcal{O}'$ that is the union of an ontology extensionally reduced from $\mathcal{O}$ and the set of previously added axioms.

Algorithm 1. Normalize($\mathcal{O}$, A, G)

Input: A $\mathcal{SHOIQ}$ ontology $\mathcal{O}$, a set A of concepts or roles, and a set G of concept or role assertions.

Output: An extensionally reduced $\mathcal{SHOIQ}$ ontology $\mathcal{O}'$ possibly with negated role inclusion axioms, a set A' of atomic concepts or atomic roles, and a set G' of atomic concept or role assertions.

1: **for** each role assertion of the form $r^{-}(a,b)$ or $\neg r^{-}(a,b)$ in G or $\mathcal{O}$ where r is an atomic role **do** Replace $r^{-}(a,b)$ with $r(b,a)$ and $\neg r^{-}(a,b)$ with $\neg r(b,a)$;

2: $\mathcal{O}' \leftarrow \varnothing$; $A' \leftarrow \varnothing$; $G' \leftarrow \varnothing$;

3: **for** each concept or role T in A **do**

4: **if** T is of the form r^{-} where r is an atomic role **then**

5: $A' \leftarrow A' \cup \{Q^{\dagger}_{r^{-}}\}$ where $Q^{\dagger}_{r^{-}}$ is a fresh atomic role;

6: $\mathcal{O}' \leftarrow \mathcal{O}' \cup \{Q^{\dagger}_{r^{-}} \sqsubseteq r^{-}\}$;

7: **else if** T is of the form $\neg r$ where r is an atomic role **then**

8: $A' \leftarrow A' \cup \{Q^{\dagger}_{\neg r}\}$ where $Q^{\dagger}_{\neg r}$ is a fresh atomic role;

9: $\mathcal{O}' \leftarrow \mathcal{O}' \cup \{Q^{\dagger}_{\neg r} \sqsubseteq \neg r\}$;

10: **else if** T is of the form $\neg r^{-}$ where r is an atomic role **then**

11: $A' \leftarrow A' \cup \{Q^{\dagger}_{\neg r^{-}}\}$ where $Q^{\dagger}_{\neg r^{-}}$ is a fresh atomic role;

12: $\mathcal{O}' \leftarrow \mathcal{O}' \cup \{Q^{\dagger}_{\neg r^{-}} \sqsubseteq \neg r^{-}\}$;

13: **else if** T is of the form C where C is not an atomic concept **then**

14: $A' \leftarrow A' \cup \{Q^{\dagger}_{C}\}$ where $Q^{\dagger}_{C}$ is a fresh atomic concept;

15: $\mathcal{O}' \leftarrow \mathcal{O}' \cup \{Q^{\dagger}_{C} \sqsubseteq C\}$;

16: **else** $A' \leftarrow A' \cup \{T\}$;

17: **for** each concept assertion or role assertion ax in G **do**

18: **if** ax is of the form $\neg r(a,b)$ where r is an atomic role **then**

19: $G' \leftarrow G' \cup \{Q^{\|}_{\neg r}(a,b)\}$ where $Q^{\|}_{\neg r}$ is a fresh atomic role;

20: $\mathcal{O}' \leftarrow \mathcal{O}' \cup \{\neg r \sqsubseteq Q^{\|}_{\neg r}\}$;

21: **else if** ax is of the form $C(a)$ where C is not an atomic concept **then**

22: $G' \leftarrow G' \cup \{Q^{\|}_{C}(a)\}$ where $Q^{\|}_{C}$ is a fresh atomic concept;

23: $\mathcal{O}' \leftarrow \mathcal{O}' \cup \{C \sqsubseteq Q^{\|}_{C}\}$;

24: **else** $G' \leftarrow G' \cup \{ax\}$;

25: **for** each ABox axiom ax in $\mathcal{O}$ **do**

26: **if** ax is of the form $C(a)$ where C is not a literal concept **then**

27: $\mathcal{O}' \leftarrow \mathcal{O}' \cup \{Q_C(a)\} \cup \{Q_C \sqsubseteq C\}$ where Q_C is a globally unique fresh atomic concept for C;

28: **else** $\mathcal{O}' \leftarrow \mathcal{O}' \cup \{ax\}$;

29: **return** $(\mathcal{O}', A', G')$;

Let f be a one-to-one mapping function on all concepts or roles $T \in A'$ such that $f(T) = r^-$ if T is of the form $Q^{\dagger}_{r^-}$, $f(T) = \neg r$ if T is of the form $Q^{\dagger}_{\neg r}$, $f(T) = \neg r^-$ if T is of the form $Q^{\dagger}_{\neg r^-}$, $f(T) = C$ if T is of the form $Q^{\dagger}_{C}$, or $f(T) = T$ otherwise. By $f(\Delta)$ we simply denote the set of ABox axioms obtained from a set Δ of ABox axioms by replacing every concept or role T occurring in Δ with $f(T)$. Let $(\mathcal{O}', A', G')$ be returned by Normalize($\mathcal{O}$, A, G), then we have the following lemma.

Lemma 1 For any set Δ of ABox axioms on concepts or roles in A', Δ is an abductive solution for $(\mathcal{O}', A', G')$ if and only if $f(\Delta)$ is an abductive solution for $(\mathcal{O}, A, G)$.

Proof. Let $\mathcal{O}^{\dagger}$ and $G^{\dagger}$ be obtained from $\mathcal{O}$ and G by replacing $r^-(a,b)$ with $r(b,a)$ and $\neg r^-(a,b)$ with $\neg r(b,a)$ for every atomic role r, then clearly $(\mathcal{O}^{\dagger}, A, G^{\dagger})$ has the same set of abductive solutions as $(\mathcal{O}, A, G)$ has. Let Δ be an

arbitrary set of ABox axioms on concepts or roles in A'. We only need to show that Δ is an abductive solution for $(\mathcal{O}', A', G')$ if and only if $f(\Delta)$ is an abductive solution for $(\mathcal{O}^\dagger, A, G^\dagger)$.

Let h be a one-to-one mapping function on all concepts or roles T appearing in $G^\dagger$ such that $h(T) = \neg r$ if T is of the form $Q^{\|}_{\neg r}$, $h(T) = C$ if T is of the form $Q^{\|}_{C}$, or $h(T) = T$ otherwise. We first show that (*) for any axiom $T(\vec{t}) \in G'$ and any set Δ of ABox axioms on concepts or roles in A', $\mathcal{O}' \cup \Delta \models T(\vec{t}) \Leftrightarrow \mathcal{O}^\dagger \cup f(\Delta) \models h(T)(\vec{t})$. For any interpretation $\mathcal{I}$, by $\vec{t}^{\,\mathcal{I}}$ we simply denote $(a^{\mathcal{I}}, b^{\mathcal{I}})$ when $\vec{t}$ is a pair made up of a and b, or denote $a^{\mathcal{I}}$ when $\vec{t}$ is a singleton a. ($\Rightarrow$) Suppose $\mathcal{O}' \cup \Delta \models T(\vec{t})$. Consider an arbitrary model $\mathcal{I}$ of $\mathcal{O}^\dagger \cup f(\Delta)$. $\mathcal{I}$ can be expanded to a model $\mathcal{I}'$ of $\mathcal{O}' \cup \Delta$ such that $T^{\mathcal{I}'} = h(T)^{\mathcal{I}'} = h(T)^{\mathcal{I}}$ for every concept or role T appearing in G', and $a^{\mathcal{I}'} = a^{\mathcal{I}}$ for every individual a in $\mathcal{O}$. Since $\mathcal{O}' \cup \Delta \models T(\vec{t})$, we have $\vec{t}^{\,\mathcal{I}'} \in T^{\mathcal{I}'}$ and thus $\vec{t}^{\,\mathcal{I}} \in h(T)^{\mathcal{I}}$. It follows that $\mathcal{O}^\dagger \cup f(\Delta) \models h(T)(\vec{t})$. ($\Leftarrow$) Suppose $\mathcal{O}^\dagger \cup f(\Delta) \models h(T)(\vec{t})$. Consider an arbitrary model $\mathcal{I}$ of $\mathcal{O}' \cup \Delta$. Let $\mathcal{I}'$ be the projection of $\mathcal{I}$ on the signature of $\mathcal{O}^\dagger \cup f(\Delta)$, then $\mathcal{I}'$ is a model of $\mathcal{O}^\dagger \cup f(\Delta)$, $h(T)^{\mathcal{I}'} = h(T)^{\mathcal{I}} \subseteq T^{\mathcal{I}}$ for every concept or role T appearing in G', and $a^{\mathcal{I}'} = a^{\mathcal{I}}$ for every individual a in $\mathcal{O}$. Since $\mathcal{O}^\dagger \cup f(\Delta) \models h(T)(\vec{t})$, we have $\vec{t}^{\,\mathcal{I}'} \in h(T)^{\mathcal{I}'}$ and thus $\vec{t}^{\,\mathcal{I}} \in T^{\mathcal{I}}$. It follows that $\mathcal{O}' \cup \Delta \models T(\vec{t})$.

Suppose Δ is an abductive solution for $(\mathcal{O}', A', G')$, then $\mathcal{O}' \cup \Delta \models ax$ for all $ax \in G'$. By (*) we have $\mathcal{O}^\dagger \cup f(\Delta) \models ax$ for all $ax \in G^\dagger$. $f(\Delta)$ must be an abductive solution for $(\mathcal{O}^\dagger, A, G^\dagger)$. Otherwise, since $\mathcal{O}' \cup \Delta$ is consistent and so is $\mathcal{O}^\dagger \cup f(\Delta)$, there must exist $\Delta' \subset f(\Delta)$ such that $\mathcal{O}^\dagger \cup \Delta' \models ax$ for all $ax \in G^\dagger$. By (*) we have $\mathcal{O}' \cup f^-(\Delta') \models ax$ for all $ax \in G'$. But then $f^-(\Delta') \subset \Delta$, contradicting that Δ is an abductive solution for $(\mathcal{O}', A', G')$.

Suppose Δ is an abductive solution for $(\mathcal{O}^\dagger, A, G^\dagger)$, then $\mathcal{O}^\dagger \cup \Delta \models ax$ for all $ax \in G$. By (*) we have $\mathcal{O}' \cup f^-(\Delta) \models ax$ for all $ax \in G'$. $f^-(\Delta)$ must be an abductive solution for $(\mathcal{O}', A', G')$. Otherwise, since $\mathcal{O}^\dagger \cup \Delta$ is consistent and so is $\mathcal{O}' \cup f^-(\Delta)$, there must exist $\Delta' \subset f^-(\Delta)$ such that $\mathcal{O}' \cup \Delta' \models ax$ for

all $ax \in G'$. By (*) we have $\mathcal{O}^\dagger \cup f(\Delta') \models ax$ for all $ax \in G^\dagger$. But then $f(\Delta') \subset \Delta$, contradicting that Δ is an abductive solution for $(\mathcal{O}^\dagger, A, G^\dagger)$. □

An example for this step is given below.

Example 4 Consider computing all abductive solutions for $(\mathcal{O}, A, G)$, where $\mathcal{O}$ is the ontology given in Example 1, $A = \{\neg\mathsf{Extraordinary}\}$ and $G = \{\mathsf{Ordinary}(\mathsf{Tom})\}$. $(\mathcal{O}, A, G)$ is normalized to $(\mathcal{O}', A', G')$, where $A' = \{\mathsf{Q}^\dagger_{\neg\mathsf{Extraordinary}}\}$, $G' = G$ and $\mathcal{O}' = \mathcal{O} \cup \{\mathsf{Q}^\dagger_{\neg\mathsf{Extraordinary}} \sqsubseteq \neg\mathsf{Extraordinary}\}$.

5.2.2 Weakening $\mathcal{O}'$

In the second step, we need to weaken $\mathcal{O}'$ to $\mathcal{O}''$ such that $\mathsf{DD}(\mathcal{O}'')$ can be computed by the KAON2 method and has no equational head atoms. To do this, we first eliminate nominals, negated role inclusion axioms and equality assertions, then standardize every concept inclusion axiom $C \sqsubseteq D$ to $\top \sqsubseteq \mathsf{NNF}(\neg C \sqcup D)$, and finally remove all maximum number restrictions from every standardized axiom, where $\mathsf{NNF}(E)$ denotes the negation normal form of a concept E, which can be computed by standard methods e.g. given in (Hustadt et al., 2007). Note that there will be no equational head atom introduced when translating $\top \sqsubseteq \mathsf{NNF}(\neg C \sqcup D)$ to first-order rules, if $\mathsf{NNF}(\neg C \sqcup D)$ has no maximum number restrictions.

The pseudo-code for this step is given in Algorithm 2 below. Lines 1–3 eliminate nominals by introducing fresh atomic concepts. Line 4 eliminates negated role inclusion axioms. By now $\mathcal{O}'$ becomes a $\mathcal{SHIQ}$ ontology. Line 5 eliminates equality assertions. Lines 6–8 further rewrite every concept inclusion axiom $C \sqsubseteq D$ to a semantically equivalent axiom $\top \sqsubseteq \mathsf{NNF}(\neg C \sqcup D)$ and eliminate all maximum number restrictions $\leq_n R.E$ occurring in the right hand side of the resulting axiom, so that $\mathcal{O}'$ becomes a $\mathcal{SHIQ}$ ontology such that $\mathsf{DD}(\mathcal{O}')$ does not contain any equational head atom.

Algorithm 2. Weaken($\mathcal{O}'$)

Input: An extensionally reduced $\mathcal{SHOIQ}$ ontology $\mathcal{O}'$ possibly with negated role inclusion axioms.

Output: A $\mathcal{SHIQ}$ ontology $\mathcal{O}''$.

1: **for** each nominal $\{a\}$ occurring in $\mathcal{O}'$ **do**

2: Replace $\{a\}$ with C_a where C_a is a globally unique fresh atomic concept;

3: $\mathcal{O}' \leftarrow \mathcal{O}' \cup \{C_a(a)\}$;

4: **for** each negated role inclusion axiom ax in $\mathcal{O}'$ **do** $\mathcal{O}' \leftarrow \mathcal{O}' \setminus \{ax\}$;

5: **for** each equality assertion ax in $\mathcal{O}'$ **do** $\mathcal{O}' \leftarrow \mathcal{O}' \setminus \{ax\}$;

6: **for** each concept inclusion axiom $C \sqsubseteq D$ in $\mathcal{O}'$ **do**

7: Replace it with $\top \sqsubseteq \mathsf{NNF}(\neg C \sqcup D)$;

8: Replace $\leq_n R.E$ with $\top$ for every maximum number restriction $\leq_n R.E$ occurring in the right hand side of $\top \sqsubseteq \mathsf{NNF}(\neg C \sqcup D)$;

9: return $\mathcal{O}'$;

Let $\mathsf{norm}(\mathcal{O})$ denote the ontology obtained from an ontology $\mathcal{O}$ by replacing every concept inclusion axiom $C \sqsubseteq D$ with $\top \sqsubseteq \mathsf{NNF}(\neg C \sqcup D)$. We call a $\mathcal{SHIQ}$ ontology $\mathcal{O}$ a $\mathcal{SHIQ}_{\leq}$ ontology if $\mathsf{norm}(\mathcal{O})$ has no equality assertions and contains no maximum number restrictions in the right hand side of any concept inclusion axiom. Let $\mathcal{O}''$ be returned by Weaken($\mathcal{O}'$), then $\mathcal{O}''$ is a $\mathcal{SHIQ}_{\leq}$ ontology. We have the following lemma.

Lemma 2. For any set Δ of ABox axioms on concepts or roles in A', $\mathcal{O}' \cup \Delta \models G'$ if $\mathcal{O}'' \cup \Delta \models G'$.

Proof. Let ax be an arbitrary atomic concept or role assertion in G'. When $\mathcal{O}'' \cup \Delta \models G'$, $\mathcal{O}'' \cup \Delta \models \{ax\}$ and thus $\mathcal{O}'' \cup \Delta \cup \{\neg ax\}$ is inconsistent. Let $\mathcal{O}^{\#}$ be the ontology obtained before line 4 and $\mathcal{O}^{\|}$ be the ontology obtained before line 6, then $\mathcal{M}(\mathcal{O}^{\#})$ and $\mathcal{M}(\mathcal{O}')$ coincide on the signature of $\mathcal{O}'$, and $\mathcal{M}(\mathcal{O}^{\#}) \subseteq \mathcal{M}(\mathcal{O}^{\|})$. For any concept inclusion axiom ax' in $\mathsf{norm}(\mathcal{O}^{\|})$, let $w(ax')$ be obtained from ax' by replacing $\leq_n R.C$ with $\top$ for every maximum number restriction $\leq_n R.C$ occurring in the right hand side of ax', then

$\mathcal{M}(ax') \subseteq \mathcal{M}(w(ax'))$. Hence, $\mathcal{M}(\mathsf{norm}(\mathcal{O}^{\|})) = \bigcap_{ax' \in \mathsf{norm}(\mathcal{O}^{\|})} \mathcal{M}(ax') \subseteq \bigcap_{ax' \in \mathcal{O}''} \mathcal{M}(ax') = \mathcal{M}(\mathcal{O}'')$ and thus $\mathsf{norm}(\mathcal{O}^{\|}) \cup \Delta \cup \{\neg ax\}$ is inconsistent. Since $\mathcal{M}(\mathcal{O}^{\#}) \subseteq \mathcal{M}(\mathcal{O}^{\|}) = \mathcal{M}(\mathsf{norm}(\mathcal{O}^{\|}))$, $\mathcal{O}^{\#} \cup \Delta \cup \{\neg ax\}$ is inconsistent. Since $\mathcal{M}(\mathcal{O}^{\#})$ and $\mathcal{M}(\mathcal{O}')$ coincide on the signature of $\mathcal{O}'$, $\mathcal{O}' \cup \Delta \cup \{\neg ax\}$ is also inconsistent and thus $\mathcal{O}' \cup \Delta \models \{ax\}$. It follows that $\mathcal{O}' \cup \Delta \models G'$. □

5.2.3 Modifying $\mathsf{DD}(\mathcal{O}'')$

In the third step, we need to compute $\mathsf{DD}(\mathcal{O}'')$ and modify it to a plain datalog program $\mathcal{P}$ such that $\mathcal{P} \cup \Delta \models G'$ implies $\mathcal{O}' \cup \Delta \models G'$ for all sets Δ of ABox axioms on concepts or roles in A'. As mentioned before, $\mathsf{DD}(\mathcal{O}'')$ is compiled from $\mathcal{O}''$ by applying the KAON2 method (Hustadt et al., 2007). However, $\mathsf{DD}(\mathcal{O}'')$ may not contain all entailed definite rules, because the KAON2 method eliminates all redundant rules that do not impact the results of the subsequent resolution operations. The elimination of redundant definite rules may make the Prolog program encoded subsequently output nothing when calling go, as shown in the following example.

Example 5. Consider the normalized problem $(\mathcal{O}', A', G')$ given in Example 4. The step for weakening $\mathcal{O}'$ yields a semantically equivalent ontology $\mathcal{O}''$ since $\mathcal{O}'$ is already a $\mathcal{SHIQ}_{\leq}$ ontology. By compiling $\mathcal{O}''$ through the KAON2 system, we obtain $\mathsf{DD}(\mathcal{O}'')$ which consists of the rules (1)–(6) given in Example 2 and the following rule.

$$\leftarrow \mathsf{Extraordinary}(x), \mathsf{Q}^{\dagger}_{\neg \mathsf{Extraordinary}}(x). \qquad (11)$$

The predicate $\mathsf{Ordinary}$ does not occur in the head of any definite rule in $\mathsf{DD}(\mathcal{O}'')$, thus the Prolog program encoded from $(\mathsf{DD}(\mathcal{O}''), A', G')$ does not output any list when called go, i.e., $\mathsf{prolog}(\mathsf{DD}(\mathcal{O}''), A', G')$ is empty.

The above example shows that the results of some resolution operations that involve a rule translated from an axiom of the form $\neg P \sqsubseteq Q^{\|}_{\neg P}$, $Q^{\dagger}_{\neg P} \sqsubseteq \neg P$ or $Q_{\neg P} \sqsubseteq \neg P$ may be treated as redundant rules in the KAON2 method. These redundant rules are entailed by $\mathsf{DD}(\mathcal{O}'')$, thus re-adding them to $\mathsf{DD}(\mathcal{O}'')$ does not impact the models of $\mathsf{DD}(\mathcal{O}'')$. In other words, we can add to $\mathsf{DD}(\mathcal{O}'')$ any rules that are entailed by $\mathsf{DD}(\mathcal{O}'')$ while still keeping that $\mathsf{DD}(\mathcal{O}'') \cup \Delta \models G'$ implies

$\mathcal{O}' \cup \Delta \models G'$ for all sets Δ of ABox axioms on concepts or roles in A'. In this way $\mathcal{P} \cup \Delta \models G'$ still implies $\mathcal{O}' \cup \Delta \models G'$, where $\mathcal{P}$ is the set of definite rules in $\mathsf{DD}(\mathcal{O}'')$.

To make the presentation concise, we only present simple resolution operations that involve new concept names introduced in the normalization step for adding redundant rules that are entailed by $\mathsf{DD}(\mathcal{O}'')$. These resolution operations can make many concept names appear in heads of definite rules, thus can compensate abductive solutions in many cases. To add more redundant rules that are entailed by $\mathsf{DD}(\mathcal{O}'')$, we can apply other resolution operations exploited in the KAON2 method (Hustadt et al., 2007).

The pseudo-code for this step is given in Algorithm 3. Line 1 initializes the resulting plain datalog program $\mathcal{P}$ as $\mathsf{DD}(\mathcal{O}'')$. For every rule R in $\mathsf{DD}(\mathcal{O}'')$, lines 3–9 add to $\mathcal{P}$ the hyper-resolution result between R and as many as possible rules translated from axioms of the form $Q^{\dagger}_{\neg P} \sqsubseteq \neg P$ or $Q_{\neg P} \sqsubseteq \neg P$. For every constraint R in $\mathcal{P}$, lines 12–16 add to $\mathcal{P}$ every resolution result between R and a rule translated from axioms of the form $\neg P \sqsubseteq Q^{\|}_{\neg P}$. Line 17 keeps only definite rules in $\mathcal{P}$ and returns it.

Algorithm 3. Modify($\mathsf{DD}(\mathcal{O}'')$)

Input: A disjunctive datalog program without equality $\mathsf{DD}(\mathcal{O}'')$.

Output: A plain datalog program without equality.

1: $\mathcal{P} \leftarrow \mathsf{DD}(\mathcal{O}'')$;

2: **for** each rule R in $\mathsf{DD}(\mathcal{O}'')$ such that $|\mathsf{head}(R)| > 0$ **do**

3: **for** every atom of the form $P(x)$ in $\mathsf{head}(R)$ **do**

4: **if** P is a concept name and $Q^{\dagger}_{\neg P}$ appears in $\mathsf{DD}(\mathcal{O}'')$ **then**

5: Remove $P(x)$ from the head of R and add $Q^{\dagger}_{\neg P}(x)$ to the body of R;

6: **else if** P is a concept name and $Q_{\neg P}$ appears in $\mathsf{DD}(\mathcal{O}'')$ **then**

7: Remove $P(x)$ from the head of R and add $Q^{\dagger}_{\neg P}(x)$ to the

body of R;

8: **else if** P is of the form $Q^{\dagger}_{\neg T}$ or $Q_{\neg T}$ where T is a concept name **then**

9: Remove $P(x)$ from the head of R and add $T(x)$ to the body of R;

10: Add to P the finally updated R;

11: **for** each constraint R in P **do**

12: **for** every atom of the form $P(x)$ in $\mathsf{body}(R)$ **do**

13: **if** P is a concept name and $Q^{\|}_{\neg P}$ appears in $\mathsf{DD}(\mathcal{O}'')$ **then**

14: Add to P the rule obtained from R by removing $P(x)$ from the body and adding $Q^{\|}_{\neg P}(x)$ to the head;

15: **else if** P is of the form $Q^{\|}_{\neg T}$ where T is a concept name **then**

16: Add to P the rule obtained from R by removing $P(x)$ from the body and adding $T(x)$ to the head;

17: **return** $\{R \in \mathcal{P} \mid \mid \mathsf{head}(R) \mid = 1\}$;

The following example shows the effectiveness of this step.

Example 6. Consider the disjunctive datalog program $\mathsf{DD}(\mathcal{O}'')$ given in Example 5. This step will yield a plain datalog program $\mathcal{P}$ having the following rule (12), which is the resolution result between rule (4) and rule (11).

$$\mathsf{Ordinary}(x) \leftarrow \mathsf{Person}(x), \mathsf{Q}^{\dagger}_{\neg \mathsf{Extraordinary}}(x). \quad (12)$$

We can see that now the predicate $\mathsf{Ordinary}$ occurs in the head of rule (12) and the Prolog program encoded from $(\mathcal{P}, A', G')$ will output a list [("Tom", "rdf:type", p_QDagNegExtraordinary)] when executing go, where QDagNegExtraordinary stands for $\mathsf{Q}^{\dagger}_{\neg \mathsf{Extraordinary}}$.

Let $\mathcal{P}$ be returned by Modify($\mathsf{DD}(\mathcal{O}'')$), then we have the following lemma.

Lemma 3. For any set Δ of ABox axioms on concepts or roles in A', $\mathcal{O}' \cup \Delta \models G'$ if $\mathcal{P} \cup \Delta \models G'$.

Proof. Let ax be an abitrary atomic concept or role assertion in G'. When $\mathcal{P} \cup \Delta \models G'$, $\mathcal{P} \cup \Delta \models \{ax\}$ and thus $\mathcal{P} \cup \Delta \cup \{\neg ax\}$ is unsatisfiable. Since every model of $\mathsf{DD}(\mathcal{O}'')$ is also a model of $\mathcal{P}$, $\mathsf{DD}(\mathcal{O}'') \cup \Delta \cup \{\neg ax\}$ is also unsatisfiable. Let $\Omega(\mathcal{O}'')$ be the $\mathcal{ALCHIQ}$ ontology obtained from $\mathcal{O}''$ in the course of the KAON2 method, then by Theorem 1, $\mathsf{DD}(\Omega(\mathcal{O}'') \cup \Delta \cup \{\neg ax\}) = \mathsf{DD}(\Omega(\mathcal{O}'')) \cup \Delta \cup \{\neg ax\} = \mathsf{DD}(\mathcal{O}'') \cup \Delta \cup \{\neg ax\}$ is unsatisfiable and thus $\Omega(\mathcal{O}'') \cup \Delta \cup \{\neg ax\}$ is inconsistent. Since $\mathcal{M}(\mathcal{O}'') \subseteq \mathcal{M}(\Omega(\mathcal{O}''))$, $\mathcal{O}'' \cup \Delta \cup \{\neg ax\}$ is also inconsistent and thus $\mathcal{O}'' \cup \Delta \models \{ax\}$. It follows that $\mathcal{O}'' \cup \Delta \models G'$. By Lemma 2, $\mathcal{O}' \cup \Delta \models G'$. □

5.2.4 Extracting Abductive Solutions from $\mathsf{prolog}(\mathcal{P}, A', G')$

In the last two steps, we encode $(\mathcal{P}, A', G')$ to a Prolog program, and then extract abductive solutions for $(\mathcal{O}, A, G)$ from $\mathsf{prolog}(\mathcal{P}, A', G')$, namely the set of lists output by the encoded Prolog program when calling go.

There is a remark on the extraction step. Consider an arbitrary set Δ of ABox axioms on concepts or roles in A. It can be seen that, all minimal subsets Δ' of Δ such that $\mathcal{O} \cup \Delta'$ is consistent and entails G are abductive solutions for $(\mathcal{O}, A, G)$. However, it is unlikely that such a subset Δ' of Δ exists when $\mathcal{O} \cup \Delta \not\models G$. In contrast, consider a list $L \in \mathsf{prolog}(\mathcal{P}, A', G')$, since $\mathcal{P} \cup \mathsf{decode}(L) \models G'$, by Lemma 3, we have $\mathcal{O}' \cup \mathsf{decode}(L) \models G'$ and thus $\mathcal{O} \cup f(\mathsf{decode}(L)) \models G$. This implies that there probably exist some subsets Δ' of $f(\mathsf{decode}(L))$ such that $\mathcal{O} \cup \Delta'$ is consistent and entails G. Hence, we do not extract abductive solutions from arbitrary hypotheses but only from lists L in $\mathsf{prolog}(\mathcal{P}, A', G')$ such that $\mathsf{decode}(L) \not\models G'$.

The following theorem shows that this method guarantees the soundness of results.

Theorem 4. Let L be a list in $\mathsf{prolog}(\mathcal{P}, A', G')$ such that $\mathsf{decode}(L)$ is a set of ABox axioms not entailing G', and Δ be a minimal subset of

$f(\mathsf{decode}(L))$ such that $\mathcal{O} \cup \Delta$ is consistent and entails G, then Δ is an abductive solution for $(\mathcal{O}, A, G)$.

Proof. Since $\mathsf{decode}(L) \not\models G'$ and all ABox axioms in $\mathsf{decode}(L)$ are only on concepts or roles in A', for all subsets Δ' of $f(\mathsf{decode}(L))$, obviously $\Delta' \not\models G$ and all ABox axioms in Δ' are only on concepts or roles in A. By the definition of abductive solutions, this theorem follows. □

By Theorem 2, Theorem 3 and Lemma 1, we see that this method also guarantees the completeness of results in some restricted class of the proposed problem. This conclusion is shown in the following theorem.

Theorem 5. If $\mathcal{O}'$ is a Horn-$\mathcal{SHIQ}_{\leq}$ ontology without negated role assertions on complex roles and G' has no atomic role assertions on complex roles, then for every abductive solution Δ for $(\mathcal{O}, A, G)$, there is a list L in $\mathsf{prolog}(\mathcal{P}, A', G')$ such that $\mathsf{decode}(L)$ is a set of ABox axioms not entailing G' and Δ is a minimal subset of $f(\mathsf{decode}(L))$ such that $\mathcal{O} \cup \Delta$ is consistent and entail G.

Proof. Let Δ' be a set of ABox axioms such that $f(\Delta') = \Delta$, then by Lemma 1, Δ' is an abductive solution for $(\mathcal{O}', A', G')$. Since G has no role assertions on complex roles, G' is a set of atomic concept assertions or atomic role assertions on simple roles. Since $\mathcal{O}'$ has no negated role assertions on complex roles and A' is a set of atomic concepts or atomic roles, by Theorem 2, Δ' is also an abductive solution for $(\mathsf{DD}(\mathcal{O}'), A', G')$. Since $\mathcal{O}'$ is a $\mathcal{SHIQ}_{\leq}$ ontology, we have $\mathsf{DD}(\mathcal{O}'') = \mathsf{DD}(\mathcal{O}')$. Since $\mathcal{O}'$ is a Horn-$\mathcal{SHIQ}_{\leq}$ ontology, $\mathsf{DD}(\mathcal{O}'')$ is a plain datalog program without equational head atoms, thus every definite rule in $\mathsf{DD}(\mathcal{O}'')$ is also in $\mathcal{P}$. It follows that $\mathsf{prolog}(\mathsf{DD}(\mathcal{O}'), A', G') = \mathsf{prolog}(\mathsf{DD}(\mathcal{O}''), A', G') \subseteq \mathsf{prolog}(\mathcal{P}, A', G')$. Since Δ' is an abductive solution for $(\mathsf{DD}(\mathcal{O}'), A', G')$ and $\mathsf{DD}(\mathcal{O}')$ does not contain any equational head atom, by Theorem 3, Δ' is a minimal set in $\{\mathsf{decode}(L) \mid L \in \mathsf{prolog}(\mathsf{DD}(\mathcal{O}'), A', G'), \mathsf{decode}(L) \not\models G', \mathcal{O}' \cup \mathsf{decode}(L)$ is consistent$\}$. Since $\mathsf{prolog}(\mathsf{DD}(\mathcal{O}'), A', G') \subseteq \mathsf{prolog}(\mathcal{P}, A', G')$, there must be a list $L \in \mathsf{prolog}(\mathcal{P}, A', G')$ such that $\mathsf{decode}(L) = \Delta'$. Then $f(\mathsf{decode}(L)) = \Delta$. Since Δ is an abductive solution for $(\mathcal{O}, A, G)$, Δ is the unique minimal subset of $f(\mathsf{decode}(L))$ such that $\mathcal{O} \cup \Delta$ is consistent and entails G. □

The remaining problem is how to efficiently compute all minimal subsets Δ of $f(\mathsf{decode}(L))$ such that $\mathcal{O} \cup \Delta$ is consistent and entails G, where L is a list in $\mathsf{prolog}(\mathcal{P}, A', G')$ such that $\mathsf{decode}(L) \not\models G'$. We tackle this problem by using a set-enumeration tree whose root is $f(\mathsf{decode}(L))$. A set-enumeration tree stores all subsets of a given set and is constructed by recursively expanding nodes from the root corresponding to the given set. Suppose each element has a sequence number from 1 to m and we use a subset S of $\{1,...,m\}$ to represent a node in the tree, where $i \in S$ means that the i^{th} element is in the node represented by S. An example set-enumeration tree whose root is represented by $\{1, 2, 3\}$ is shown in Figure 1 (a). A node represented by $\{1,2,...,i,i+j,...,n\}$ (where $0 \le i \le n \le m$ and $j > 1$) has exactly i children, where the k^{th} $(1 \le k \le i)$ child is obtained from its parent by deleting the $(i+1-k)^{th}$ element, as shown in Figure 1 (b).

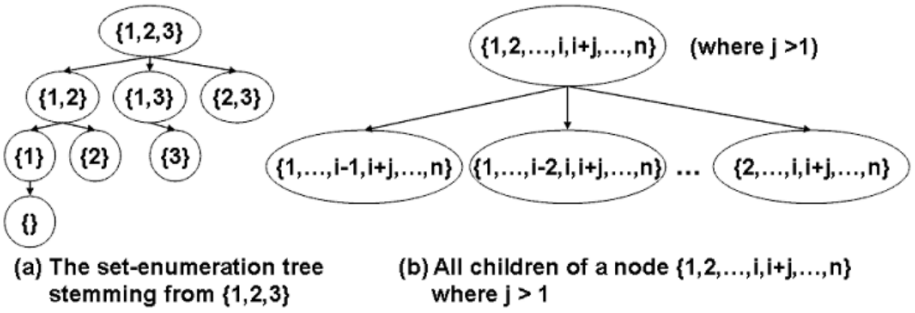

Figure 1: Illustrations for set-enumeration trees

To find abductive solutions for $(\mathcal{O}, A, G)$ among all subsets of $f(\mathsf{decode}(L))$, the set-enumeration tree stemming from $f(\mathsf{decode}(L))$ is traversed in a depth-first manner. The pseudo-code is given in Algorithm 4, where $Ch(\Delta)$ returns the set of children of a subset Δ of $f(\mathsf{decode}(L))$ in the set-enumeration tree stemming from $f(\mathsf{decode}(L))$, Traverse($f(\mathsf{decode}(L))$, $\mathcal{O}$, G, S) returns the union of S and the set of all minimal subsets Δ of $f(\mathsf{decode}(L))$ such that $\mathcal{O} \cup \Delta$ is consistent and entails G.

Algorithm 4. Traverse(Δ, $\mathcal{O}$, G, S)

Input: A set Δ of ABox axioms, a $\mathcal{SHOIQ}$ ontology $\mathcal{O}$, a set G of ABox axioms, and a set S of abductive solutions previously found.

Output: A updated set of abductive solutions.

1: **if** $\mathcal{O} \cup \Delta$ is consistent but does not entail G **then return** S;

2: $S' \leftarrow S$;

3: **for** each Δ' in $Ch(\Delta)$ **do** $S' \leftarrow$ Traverse(Δ', $\mathcal{O}$, G, S');

4: **if** $S' = S$ and $\mathcal{O} \cup \Delta$ is consistent and $\mathcal{O} \cup \Delta$ entails G and Δ has no subsets in S **then** $S' \leftarrow S' \cup \{\Delta\}$;

5: **return** S';

Algorithm 4 can be explained as follows. Suppose Δ is the current node to be processed in Algorithm 4. In case $\mathcal{O} \cup \Delta$ is consistent but does not entail G, since any descendant of Δ cannot entail G, no descendants of Δ can be abductive solutions (see line 1). In other cases, all children of Δ in the set-enumeration tree are processed recursively (see line 3). Note that, during the traversal of a set-enumeration tree, all subsets of Δ must have been processed after all descendants of Δ are processed. Hence, whether Δ is an abductive solution can be decided after all its descendants are processed (see line 4).

6. EXPERIMENTAL EVALUATION

We implemented both the method for the restricted class (simply called the restricted method) and the one for the full class (simply called the general method), where Pellet (Sirin et al., 2007) API is used to realize consistency checking and entailment checking in the course of extracting abductive solutions from the output of B-Prolog. We conducted experiments on thirteen benchmark ontologies that have large ABoxes. The first two ontologies are Semintec (http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm), which is an ontology about financial services, and Vicodi (http://www.vicodi.org/), which is an ontology on European history. The next five are the Lehigh University Benchmark (LUBM) (Guo et al., 2005) ontologies LUBMn ($n = 1$, ..., 5), where LUBMn denotes the LUBM ontology containing the data of n universities. The above ontologies have been used as benchmark ones in comparing different DL reasoners (Motik & Sattler, 2006). The last six ontologies are the University Benchmark (UOBM) (Ma et al., 2006) ontologies UOBM-Lite$n$ and UOBM-DLn ($n = 1, 2, 3$), where UOBM-Lite$n$ and

UOBM-DLn denote the UOBM ontologies (OWL Lite version and OWL DL version, respectively) containing the data of n universities. We could not test larger UOBM ontologies that involve more universities, because B-Prolog ran out of memory when loading the Prolog programs encoded from these ontologies. The characteristics of all test ontologies are shown in Table 2. All experiments were conducted on a PC with Pentium Dual Core 2.60GHz CPU and 2GB RAM, running Windows XP, where the maximum Java heap size was set to 1GB. Note that B-Prolog does not work in the Java Virtual Machine and its memory usage is not limited by the maximum Java heap size. Our implemented system for ABox abduction, accessorial tools and test ontologies are all available at http://jfdu.limewebs.com/abduction/.

Table 2: The characteristics of test ontologies

Ontology	**#C**	**#R**	**#TA**	**#AA**	**#I**
Semintec	60	16	219	65,240	17,941
Vicodi	194	12	223	116,181	33,238
LUBM 1-5	43	32	93	100,543 ~ 624,532	17,174 ~ 102,368
UOBM-Lite 1-3	51	43	145	245,740 ~ 575,380	37,704 ~ 71,901
UOBM-DL 1-3	69	44	206	260,900 ~ 607,248	37,927 ~ 72,059

Note: “#C", “#R", “#TA", “#AA" and “#I" are the numbers of concept names, role names, TBox axioms, ABox axioms and individual names, respectively.

6.1 Results on the Restricted Method

We first compared the general method with the restricted method on handling test ontologies for which the results obtained from the KAON2 method are plain datalog programs with equality. These ontologies include Semintec and all UOBM-Lite$n$. We randomly generated forty atomic concept assertions. We set all atomic concepts as abducible predicates and every singleton set made up of a generated concept assertion as an observation. The general method finishes in half an hour for all observations. But the restricted method always exceeds half a day when handling an observation that is known to have abductive solutions from the results of the general method. It shows that the rules added to axiomatize equality heavily impair the efficiency of ABox abduction. This can be explained by the fact that these rules introduce cycles in the encoded Prolog program, diffusing the search space to a huge one. Thus we do not recommend the restricted method even when the given problem is in the corresponding restricted class.

6.2 Preparation for the General Method

We implemented the general method in a way that all observations can be handled without starting from scratch as long as the observations are made up of literal concept assertions. Given a test ontology $\mathcal{O}$, a set A of abducible predicates and some observations made up of literal concept assertions, the implementation works in two phases. In the first phase, $\mathcal{O}$, A and the set of all literal concepts in $\mathcal{O}$ are encoded into a Prolog program, which is then loaded to B-Prolog. Since this phase is independent from specific observations made up of literal concept assertions, we call it the preprocess phase. In the second phase, when given an observation G made up of literal concept assertions, the implementation encodes G into a Prolog rule and combines it with the loaded Prolog program to compute abductive solutions for $(\mathcal{O}, A, G)$. We call this phase the query phase.

Using hypothesis is a new feature in ABox abduction which is not in traditional ABox reasoning. The performance of the general method depends on the size of the hypothesis space which is determined by the number of abductive predicates. To see how the performance changes against different numbers of abductive predicates, we designed four suites of experiments on the general method, each of which uses different numbers of abductive predicates. For the first suite, called the allAC suite, we set all atomic concepts as abducible predicates. For the second suite, called the allLC suite, we set all literal concepts as abducible predicates. For the third suite, called the allEAC suite, we set as abducible predicates all atomic concepts and all existential restrictions of the form $\exists r.P$ where r is an atomic concept and P is an atomic concept subsumed by the domain of r in the test ontology. For the last suite, called the allELC suite, we set as abducible predicates all literal concepts and all existential restrictions of the form $\exists r.C$ where r is an atomic concept and C is a literal concept subsumed by the domain of r in the test ontology. For all suites of experiments on a test ontology $\mathcal{O}$, we randomly generated forty concept assertions $C(a)$ such that $\mathcal{O} \not\models C(a)$ and $\mathcal{O} \not\models \neg C(a)$, out of which twenty are atomic concept assertions and twenty are negative ones, and set every singleton set made up of a generated concept assertion as an observation. We did not generate $C(a)$ such that $\mathcal{O} \models C(a)$ because there is only one trivial abductive solution $\varnothing$ for $(\mathcal{O}, A, \{C(a)\})$. We also did not generate $C(a)$ such that $\mathcal{O} \models \neg C(a)$ because there is no abductive solution for $(\mathcal{O}, A, \{C(a)\})$. To see how the general method scales with increasing sizes of ABoxes, we generated the same set of observations for different LUBM n (resp. different UOBM-Lite n or different UOBM-DL n).

The aim of these experiments is to verify the general method in terms of efficiency and scalability against different numbers of abducible predicates and different sizes of ABoxes. Note that the set of abducible predicates in the allAC suite is a subset of that in the allLC or allEAC suite, while the set of abducible predicates in the allLC or allEAC suite is a subset of that in the allELC suite. Hence we have the fol-

lowing partial order on the complexity of abducible predicates: allAC < allLC, allAC < allEAC, allLC < allELC, allEAC < allELC. So far we cannot verify the completeness of the general method, because the baseline method which generates and tests all candidate abductive solutions is infeasible in traversing such a huge search space for any test ontology. Nevertheless, we can still provide some information on the completeness. Since Vicodi and all LUBM n are Horn-$\mathcal{SHIQ}_{\leq}$ ontologies, by Theorem 5, the general method must compute the complete set of abductive solutions for an observation made up of atomic concept assertions, in all suites of experiments.

6.3 Results on the General Method

Table 3: The statistics for Semintec and Vicodi

Ontology	Suite	#Abd	Pre. Time	Max. Time	Avg. Time	Max. Num	Avg. Num
Semintec	allAC	59	22.1	5.4	4.2	3	2.5
Semintec	allLC	118	22.2	7.4	5.4	3	3.0
Semintec	allEAC	119	24.2	4.5	3.4	3	2.5
Semintec	allELC	178	24.2	4.7	4.1	3	3.0
Vicodi	allAC	194	76.3	22.4	3.8	11	1.8
Vicodi	allLC	388	76.5	22.3	11.2	11	3.3
Vicodi	allEAC	673	78.7	480.3	16.0	212	7.2
Vicodi	allELC	867	78.9	480.8	19.9	212	8.6

Note: “#Abd" is the number of abducible predicates. “Pre.Time" is the execution time (sec) of the preprocess phase, “Max.Time" (resp. “Avg.Time") is the maximum (resp. average) execution time (sec) for computing abductive solutions for an observation in the query phase. “Max.Num" (resp. “Avg.Num") is the maximum (resp. average) number of computed abductive solutions for an observation.

The test results for Semintec and Vicodi are shown in Table 3. For all suites of experiments, the execution time of the preprocess phase is almost the same, except that the execution time for the allEAC or allELC suite is slightly longer. Both the execution time for computing abductive solutions for an observation and the number of computed abductive solutions increase when the complexity of abducible predicates increases. For each suite and each observation, the computation of abductive solutions is accomplished without running out of memory. In particular, the maximum execution time for computing abductive solutions for an observation is less than half a minute for almost all suites, except that for Vicodi and two suites (allEAC and allELC), the maximum execution time is about six minutes. The reason why compu-

ting abductive solutions takes a rather long time in some cases is that there are many abductive solutions in these cases.

Table 4: A portion of statistics for LUBM n , UOBM-Lite n and UOBM-DL n

Ontology	Suite	#Abd	#Succ	Max.Num	Avg.Num
LUBM n	allAC	43	40	3	0.3
LUBM n	allEC	86	40	4	1.2
LUBM n	allEAC	373	40	53	5.1
LUBM n	allELC	588	40	96	9.6
UOBM-Lite n	allAC	51	24	9	1.5
UOBM-Lite n	allEC	102	24	9	1.8
UOBM-Lite n	allEAC	659	24	148	17.2
UOBM-Lite n	allELC	1067	24	250	21.8
UOBM-DL n	allAC	68	16	15	1.5
UOBM-DL n	allEC	136	16	15	1.8
UOBM-DL n	allEAC	862	16	101	9.4
UOBM-DL n	allELC	1406	15	169	14.7

Note: "#Abd" is the number of abducible predicates. "#Succ" is the number of successful observations for which computing abductive solutions is accomplished in finite time. "Max.Num" (resp. "Avg.Num") is the maximum (resp. average) number of computed abductive solutions for a successful observation.

The test results for LUBM n , UOBM-Lite n and UOBM-DL n (excluding execution time) are shown in Table 4. Since we used the same set of observations for all LUBM n (resp. all UOBM-Lite n or all UOBM-DL n), we got the same results on all aspects except execution time for different n . Due to limited memory, some observations cannot be properly handled in some test cases. We call an observation a successful one if the computation of abductive solutions for it is accomplished without running out of memory. For all suites of experiments on LUBM n , all 40 observations are successful ones. For all suites of experiments on UOBM-Lite n , 24 observations are successful ones. For almost all suites of experiments on UOBM-DL n , 16

observations are successful ones except that 15 are successful for the allELC suite. All failures are caused by B-Prolog, which ran out of memory during executing the encoded Prolog program.

The execution time of the preprocess phase (simply preprocessing time) against different n is shown in Figure 2. For all LUBMn or all UOBM-Lite$n$, the preprocessing time is almost the same for different suites. For all UOBM-DLn, the preprocessing time for the allELC suite and the allEAC suite is significantly longer than the preprocessing time for the allLC suite and the allAC suite. The main reason why UOBM-DLn have different results on preprocessing time is that they are expressed in the most expressive language among all test ontologies, while the resolution operations between the clauses translated from complex axioms in UOBM-DLn and the clauses used to normalize abducible predicates from existential restrictions to atomic concepts result in much more rules in the compiled disjunctive datalog program. Regarding the scalability against different sizes of ABoxes, the preprocessing time for LUBMn, UOBM-Lite$n$ or UOBM-DLn increases smoothly when n increases. The general method shows a near linear scalability on preprocessing time.

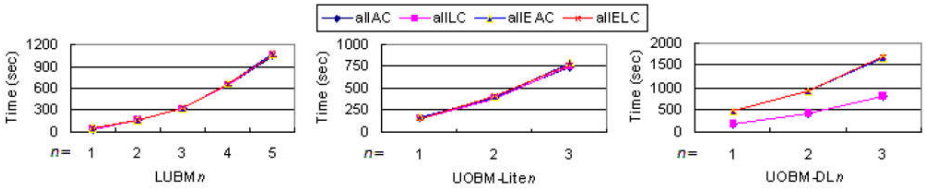

Figure 2: The execution time of the preprocess phase

The maximum/average execution time (in the query phase) for computing abductive solutions for a successful observation is shown in Figure 3 and Figure 4, respectively. For all LUBMn, UOBM-Lite$n$ or UOBM-DLn, the execution time for computing abductive solutions for a successful observation increases when n increases and when the complexity of abducible predicates increases. The general method shows a near linear scalability on the execution time for computing abductive solutions against different sizes of ABoxes. For the allAC suite and the allLC suite, the maximum execution time for computing abductive solutions for a successful observation is relatively short and is less than eight minutes for all test ontologies. For the other two suites, the maximum execution time for computing abductive solutions for a successful observation is relatively long, but the average execution time is only about one tenth of the maximum execution time. Table 4 has shown some hints for explaining why computing abductive solutions takes a long time in some cases. The main reason is that there are many abductive solutions in these cases.

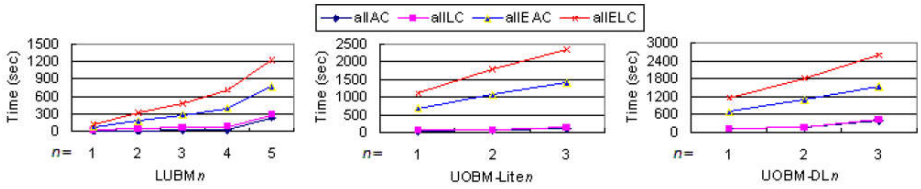

Figure 3: The maximum execution time for computing abductive solutions for a successful observation

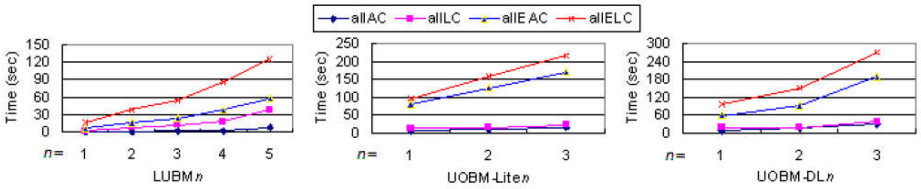

Figure 4: The average execution time for computing abductive solutions for a successful observation

6.4 Discussion

The general method for ABox abduction provides an effective way to search over $O(2^{|A_C||N_I|+|A_R||N_I|^2})$ candidate abductive solutions, where $|A_C|$, $|A_R|$ and $|N_I|$ are respectively the number of abducible concepts, the number of abducible roles and the number of individual names. That is, it localizes the search in small areas, each of which contains a portion of abductive solutions. With this manner the method can efficiently compute abductive solutions in benchmark ontologies that have large ABoxes. In particular, experimental results show that the method works well for hundreds of abducible predicates that are concepts more complex than literal ones. The results also show that the method scales well against different sizes of ABoxes. We believe that the method is able to scale to much larger ABoxes provided that it works with more memory.

It should be mentioned that we did not show the experimental results about the cases where roles are used as abducible predicates. But we had actually conducted some of such experiments. The results are not promising because there are usually too many abductive solutions. Recall a simple example given in Section 3: $\mathcal{O} = \{\exists \mathsf{hasFather}.\top \sqsubseteq \mathsf{Person}$, $\neg\mathsf{hasFather}(a_1, a_1)$, $\mathsf{Person}(a_2)$, ..., $\mathsf{Person}(a_n)\}$, $A = \{\mathsf{hasFather}\}$ and $G = \{\mathsf{Person}(a_1)\}$. There are $n-1$ abductive solutions for $(\mathcal{O}, A, G)$, i.e. $\{\mathsf{hasFather}(a_1, a_2)\}$, ..., $\{\mathsf{hasFather}(a_1, a_n)\}$. This example has similar nature as the test cases where

atomic roles are used as abducible predicates, thus it is not a surprise when we saw the general method did not finish and continued outputting abductive solutions after several hours. Although the general method does not handle abducible roles well, it can still be of practical use because abducible roles can often be substituted by abducible predicates that are existential restrictions. Consider the aforementioned example. If the abducible role $\mathsf{hasFather}$ is replaced with $\exists \mathsf{hasFather}.\top$ in A, then there is only one abductive solution for $(\mathcal{O}, A, G)$, i.e. $\{\exists \mathsf{hasFather}.\top(\mathsf{a}_1)\}$, which essentially generalizes $\{\mathsf{hasFather}(a_1, a_2)\}$, ..., $\{\mathsf{hasFather}(a_1, a_n)\}$. Hence, we recommend using existential restrictions rather than roles as abducible predicates.

7. RELATED WORK

Abductive reasoning in DLs was initiated by Elsenbroich et al. (2006). They classified the tasks of abductive reasoning into two categories, namely TBox abduction and ABox abduction, and described specific tasks in these two categories using case studies. The necessity of abductive reasoning in DLs was reemphasized by Bada et al. (2008) to support ontology quality control.

Although abductive reasoning in DLs is important, there is still not much work in this area, probably due to the high complexity of abductive reasoning. Computing a set-minimal abductive solution for propositional Horn theories is already NP-hard (Selman & Levesque, 1990). It is even harder for more general propositional theories (Eiter & Gottlob, 1995). Bienvenu (2008) adapted this complexity result to the $\mathcal{EL}$ family (Baader et al., 2005) and showed that the problem of computing a minimal set of atomic concepts $\{A_1, ..., A_n\}$ such that $A_1 \sqcap ... \sqcap A_n$ is satisfiable and subsumed by an observed atomic concept C in an $\mathcal{EL}^{++}$ TBox is NP-hard. Considering that $\mathcal{EL}^{++}$ is a rather inexpressive DL, the complexity should be at least as high for general DLs. Since the problem considered by Bienvenu (2008) can be treated as a problem for ABox abduction by defining abducible predicates as atomic concepts and the observation as $\{C(a)\}$ where a is a fresh individual, the complexity for ABox abduction is at least NP-hard.

The work on methods for TBox abduction has a longer history than that for ABox abduction. Before the use of abductive reasoning in DLs was comprehensively discussed by Elsenbroich et al. (2006), Colucci et al. (2004) have proposed a tableaux-based method for concept abduction in $\mathcal{ALN}$ TBoxes, which computes an $\mathcal{ALN}$ concept H such that $C \sqcap H$ is satisfiable and subsumed by D in a given $\mathcal{ALN}$ TBox, for two given $\mathcal{ALN}$ concepts C and D. This method has only been empirically verified in small-scale applications with a few hundreds of concepts (Colucci et al., 2004; Noia et al., 2007). To support existential restrictions that are not allowed in $\mathcal{ALN}$, Noia et al. (2009) also proposed a tableaux-based method for

concept abduction in $\mathcal{SH}$ TBoxes. No evaluation results are available for this method. Targeting a different problem for TBox abduction, which computes a set of concept inclusion axioms to enforce entailment of a given concept inclusion axiom, Hubauer et al. (2010) proposed an automata-based method for TBox abduction in $\mathcal{EL}$ TBoxes. Also, there are no evaluation results available for this method. Considering that $\mathcal{ALN}$, $\mathcal{SH}$ and $\mathcal{EL}$ do not support nominals, the above methods cannot directly be applied to ABox abduction. Moreover, there is no empirical evidence that these methods are practically feasible in handling a large number of axioms that involve nominals. Hence, we do not consider adapting existing methods for TBox abduction to ABox abduction.

As mentioned in section 1, the work on ABox abduction is rare. Peraldi et al. (2007) proposed a method, based on backward inference, to compute abductive solutions in a DL ontology accompanying rules. The method has the following limitations: the axioms that can be used are restricted to some special forms; the computed abductive solutions may not be subset-minimal. Recently, Klarman et al. (2011) proposed a method, based on tableaux and resolution techniques, to compute all abductive solutions in an $\mathcal{ALC}$ ontology. It is still unclear how to extend this method to support more expressive DLs. Furthermore, the method does not guarantee termination. In contrast, our proposed method guarantees termination and set-inclusion minimality of abductive solutions; moreover, it works for $\mathcal{SHOIQ}$ which is much more expressive than $\mathcal{ALC}$. Currently, we are unable to empirically compare our proposed method with the above two methods, because for the first one, neither the ontology nor the system they used is publicly accessible, while for the second one, no evaluation results are available.

Abductive reasoning in logic programming (Kakas et al., 1998) is a relatively prolific area. There exist mature proof procedures for abductive reasoning in logic programming. The premier proof procedure is the SLDA procedure (Kakas & Mancarella, 1990), which extends the well-known SLD resolution (Selective Linear Definite clause resolution) with abduction. This procedure has been extended to the SLDNFA procedure (Denecker & Schreye, 1992) to support normal logic programs that may contain negation-as-failure. The SLDNFA procedure has also been extended or refined to other proof procedures such as the IFF (if-and-only-if) procedure (Fung & Kowalski, 1997). The two state-of-the-art abduction systems CIFF (Mancarella et al., 2009) and $\mathcal{A}$-system (Kakas et al., 2001), mentioned in this deliverable, are built on the above proof procedures, where CIFF is built on an extension of IFF and $\mathcal{A}$-system is built on SLDNFA. However, these abduction systems cannot solve our proposed program for ABox abduction because they do not work for expressive DLs. Although we provide a method for reducing our proposed problem to an abduction problem on plain datalog programs (see subsection 4.2), these systems are still inapplicable because they currently do not guarantee termination in handling cyclic plain datalog programs. Hence, we implement the SLDA procedure on a Prolog engine B-Prolog, through an encoding method proposed in subsection 4.1, to solve the reduced abduction problem. This implementation uses linear tabling (Shen et al., 2001) supported by B-Prolog to solve the reduced problem in finite time.

8. CONCLUSION

ABox abduction is an indispensable non-standard reasoning facility in DLs, but the work on ABox abduction is rare. What is even worse, currently no method for ABox abduction works for very expressive DLs and computes minimal solutions in finite time. Under this situation, this deliverable made the following contributions so as to pave a way to practical ABox abduction.

Firstly, the paper proposed a new problem for ABox abduction. This problem follows some ideas from abductive reasoning in logic programming, e.g., an abductive solution, namely a result of ABox abduction, should be subset-minimal, and introduces the notion of abducible predicate to guarantee finite number of abductive solutions. That is, all ABox axioms in an abductive solution should be on a finite set of abducible predicates which can be arbitrary concepts or roles.

Secondly, the paper accordingly proposed a method for the above problem. The method is based on a reduction from DL $\mathcal{SHOIQ}$ to plain datalog. That is, the abductive solutions for the original problem which is expressed in $\mathcal{SHOIQ}$ are computed by reducing the original problem to an abduction problem in plain datalog programs, and then extracting true results from abductive solutions for the reduced abduction problem. Although the reduction may not guarantee semantic equivalence, the proposed method still guarantees soundness and conditional completeness of computed results.

At last, the paper also provided evaluation results on benchmark ontologies that have large ABoxes. The results show that the method works well for hundreds of abducible predicates and scales well against different sizes of ABoxes. To the best of our knowledge, these results are the first evaluation results for ABox abduction on large benchmark ontologies.

As shown in our experiments, the bottleneck of the proposed method lies in solving the reduced abduction problem. Hence, in future work we plan to investigate which fragments of plain datalog allow for efficient computation of abductive solutions, and develop methods for reducing the proposed problem to an abduction problem expressed in such fragments. The proposed problem has a potential issue that there may be too many abductive solutions, especially when roles are used as abducible predicates. We also plan to tackle this issue by refining the proposed problem, e.g., defining stricter minimal criteria for abductive solutions.

ACKNOWLEDGEMENTS

This research has been funded by the European Commission within the 7th~Framework Programme/Marie~Curie Industry-Academia-Partnerships-and-Pathways schema/PEOPLE Work Programme 2011 project K-Drive number~286348 (cf.\\url{http://www.kdrive-project.eu}).

REFERENCES

Baader, F., Brandt, S., & Lutz, C. (2005). Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 364–369).

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., & Patel-Schneider, P. F. (Eds.) (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Baader, F., & Peñaloza, R. (2007). Axiom pinpointing in general tableaux. In *Proceedings of the 16th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)* (pp. 11–27).

Bada, M., Mungall, C., & Hunter, L. (2008). A call for an abductive reasoning feature in owl-reasoning tools toward ontology quality control. In *Proceedings of the 5th OWLED Workshop on OWL: Experiences and Directions*.

Bienvenu, M. (2008). Complexity of abduction in the $\mathcal{EL}$ family of lightweight description logics. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR)* (pp. 220–230).

Colucci, S., Noia, T. D., Sciascio, E. D., Donini, F. M., & Mongiello, M. (2004). A uniform tableaux-based approach to concept abduction and contraction in $\mathcal{ALN}$. In *Proceedings of the 17th International Workshop on Description Logics*.

Denecker, M., & Schreye, D. D. (1992). SLDNFA: An abductive procedure for normal abductive programs. In *Proceedings of the Joint International Conference and Symposium on Logic Programming (JICSLP)* (pp. 686–700).

Du, J., Qi, G., Shen, Y., & Pan, J. Z. (2011). Towards practical abox abduction in large OWL DL ontologies. In *Proceedings of the 25th National Conference on Artificial Intelligence (AAAI)* (pp. 1160–1165).

Eiter, T., & Gottlob, G. (1995). The complexity of logic-based abduction. *Journal of the ACM*, *42*, 3–42.

Eiter, T., Gottlob, G., & Mannila, H. (1997). Disjunctive datalog. *ACM Transactions on Database Systems*, *22*, 364–418.

Elsenbroich, C., Kutz, O., & Sattler, U. (2006). A case for abductive reasoning over ontologies. In *Proceedings of the 3rd OWLED Workshop on OWL: Experiences and Directions*.

Fitting, M. (1996). *First-order Logic and Automated Theorem Proving (2nd ed.)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Fung, T. H., & Kowalski, R. A. (1997). The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, *33*, 151–165.

Grau, B. C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P. F., & Sattler, U. (2008). OWL 2: The next step for OWL. *Journal of Web Semantics*, *6*, 309–322.

Guo, Y., Pan, Z., & Heflin, J. (2005). LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, *3*, 158–182.

Horrocks, I., Patel-Schneider, P. F., & van Harmelen, F. (2003). From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, *1*, 7–26.

Hubauer, T., Lamparter, S., & Pirker, M. (2010). Automata-based abduction for tractable diagnosis. In *Proceedings of the 23rd International Workshop on Description Logics*.

Hustadt, U., Motik, B., & Sattler, U. (2007). Reasoning in description logics by a reduction to disjunctive datalog. *Journal of Automated Reasoning*, *39*, 351–384.

Kakas, A. C., Kowalski, R. A., & Toni, F. (1998). The role of abduction in logic programming. In Dov M. Gabbay, C.J. Hogger, and J. A. Robinson (Ed.), *Handbook of Logic in Artificial Intelligence and Logic Programming Volume 5: Logic Programming*. (pp. 235–324). Oxford University Press.

Kakas, A. C., & Mancarella, P. (1990). Database updates through abduction. In *Proceedings of the 16th International Conference on Very Large Data Bases (VLDB)* (pp. 650–661).

Kakas, A. C., Nuffelen, B. V., & Denecker, M. (2001). $\mathcal{A}$ -System: Problem solving through abduction. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 591–596).

Kalyanpur, A., Parsia, B., Horridge, M., & Sirin, E. (2007). Finding all justifications of OWL DL entailments. In *Proceedings of the 6th International Semantic Web Conference (ISWC)* (pp. 267–280).

Klarman, S., Endriss, U., & Schlobach, S. (2011). Abox abduction in the description logic $\mathcal{ALC}$. *Journal of Automated Reasoning*, *46*, 43–80.

Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., & Liu, S. (2006). Towards a complete OWL ontology benchmark. In *Proceedings of the 3rd European Semantic Web Conference (ESWC)* (pp. 125–139).

Mancarella, P., Terreni, G., Sadri, F., Toni, F., & Endriss, U. (2009). The CIFF proof procedure for abductive logic programming with constraints: Theory, implementation and experiments. *Theory and Practice of Logic Programming*, *9*, 691–750.

Motik, B., & Sattler, U. (2006). A comparison of reasoning techniques for querying large description logic aboxes. In *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)* (pp. 227–241).

Noia, T. D., Sciascio, E. D., & Donini, F. M. (2007). Semantic matchmaking as non-monotonic reasoning: A description logic approach. *Journal of Artificial Intelligence Research*, *29*, 269–307.

Noia, T. D., Sciascio, E. D., & Donini, F. M. (2009). A tableaux-based calculus for abduction in expressive description logics: Preliminary results. In *Proceedings of the 22nd International Workshop on Description Logics*.

Peirce, C. S. (1992). *Reasoning and the logic of things: The Cambridge conferences lectures of 1898*. Harvard University Press.

Peirce, C. S., Hartshorne, C., Weiss, P., and Burks, A. W. (1994). The Collected Papers of Charles Sanders Peirce. Electronic Edition. *Volume 8: Reviews, Correspondence, and Bibliography*. InteLex Corporation.

Peraldi, I., Kaya, A., Melzer, S., Möller, R., & Wessel, M. (2007). Towards a media interpretation framework for the semantic web. In *Proceedings of 2007 IEEE/WIC/ACM International Conference on Web Intelligence (WI)* (pp. 374–380).

Schlobach, S., & Cornet, R. (2003). Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 355–362).

Selman, B., & Levesque, H. J. (1990). Abductive and default reasoning: A computational core. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI)* (pp. 343–348).

Shen, Y., Yuan, L., You, J., & Zhou, N. (2001). Linear tabulated resolution based on prolog control strategy. *Theory and Practice of Logic Programming*, *1*, 71–103.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, *5*, 51–53.