IJCAI Workshop 13
Ontologies and Logic Programming for Query Answering

http://ontolp.lsis.org/

Proceedings

Chairs:
Odile PAPINI
Salem BENFERHAT
Laurent GARCIA
Marie-Laure MUGNIER

July 25th 2015
Ontologies and Logic Programming for Query Answering
http://ontolp.lsis.org


Workshop description The aim of this workshop is to bridge knowledge representation and reasoning in artificial intelligence and web of knowledge communities in order to encourage the emergence of new solutions for reasoning with lightweight ontologies. The workshop focuses on languages and techniques that allow for:
- Query answering while taking ontologies into account.
- Non monotonic reasoning for inconsistency handling and exception handling and expressing default negations in ontologies.

Concerning the first point, a challenging issue is how to adapt or extend Answer Set Programming to represent ontological knowledge. In particular, can (a fragment of) ASP cover lightweight ontological languages while keeping decidability and efficiency?

Concerning the second point, a challenging issue is how to extend lightweight ontological languages with non-monotonic features, while keeping a good computational complexity. In particular,

i) how to embed exceptions-based and inconsistency tolerant-based reasoning in a tractable ontological language?

ii) how to integrate uncertainty information in lightweight ontological languages?

iii) how to define merging operations where both inputs and outputs are in lightweight ontological languages?

This workshop receives support from ANR (French National Research Agency), ASPIQ project reference ANR-12-BS02-0003. (http://www.agence-nationale-recherche.fr/)

Topics Topics of interest include but are not limited to:
- Reasoning with lightweight ontologies
- Reasoning with tractable fragments of OWL languages
- Belief change & tractable Description logics
- Reasoning on the web of data
- Ontology-based data-access
- Ontological query answering
- Belief change & ASP
- Datalog + & existential rules
- ASP & Description logics
- ASP & Uncertainty
- First order ASP
- Rulelog
Program Committee

Maria Teresa Alsinet Bernadó (University of Lleida, Spain)
Jean-François Baget (INRIA, Montpellier, France)
Chitta Baral (Arizona State University, U.S.A.)
Madalina Croitoru (University of Montpellier 2, France)
Jim Delgrande (Simon Fraser University, Vancouver, Canada)
Thomas Eiter (Vienna University of Technology, Austria)
Jérôme Euzenat (INRIA, Grenoble, France)
Eduardo Fermé (University of Madeira, Portugal)
Sergio Alejandro Gomez (Universidad Nacional del Sur Alem, Argentina)
Fabio Grandi (University of Bologna, Italy)
Guido Governatori (NICTA Queensland, Australia)
John Horty (University of Maryland, U.S.A.)
Gabriele Kern-Isberner (University of Dortmund, Germany)
Sylvain Lagrue (University of Artois, France)
Claire Lefèvre (University of Angers, France)
Joao Leite (University of Lisbon, Portugal)
Nicola Leone (University of Calabria, Italy)
Stéphane Loiseau (University of Angers, France)
Thomas Lukasiewicz (University of Oxford, U.K.)
Robert E. Mercer (University of Western Ontario, Canada)
Thomas Meyer (CSIR Meraka Institute, Pretoria, South of Africa)
Farid Nouioua (Aix Marseille University, France)
Guilin Qi (Southeast University, China)
Vincent Risch (Aix Marseille University, France)
Riccardo Rosati (University La Sapienza, Roma, Italy)
Marie-Christine Rousset (University of Grenoble, France)
Torsten Schaub (University of Potsdam, Germany)
Steven Schockaert (Cardiff University, U.K.)
Guillermo Simari (University Bahia blanca, Argentina)
Igor Stéphan (University of Angers, France)
Umberto Straccia (ISTI-CNR, Pisa, Italy)
Kewen Wang (Griffith University, Australia)
Stefan Woltran (Vienna University of Technology, Austria)
Eric Würbel (University of Toulon, France)

Additional reviewers

Michael Morak (University of Oxford, U.K.)
Ricardo Gonçalvez (University of Lisbon, Portugal)
Antonius Weinzierl (Vienna University of Technology, Austria)
Program

8:30 Welcome

8:45-9:40 Invited talk: Steven Schockaert
Extending Answer Set Programming using Generalized Possibilistic Logic

9:40-10:30 Logic Programming 1

9:40-10:05 Riccardo Zese, Elena Bellodi, Evelina Lamma, Fabrizio Riguzzi.
Logic Programming Techniques for Reasoning with Probabilistic Ontologies.

9:40-10:05 Verónica Dahl, Sergio Tessaris, Thomas Frühwirth.
Imperfect Querying through Womb Grammars plus Ontologies.

10:05-10:30 Fabien Garreau, Laurent Garcia, Claire Lefèvre, Igor Stéphan.
∃-ASP

10:30-11:00 Coffee break

11:00-12:15 Description Logic

11:00-11:25 Veronica Dahl, Sergio Tessaris, Thomas Fruehwirth.
Imperfect Querying through Womb Grammars plus Ontologies.

11:25-11:50 Roberto Confalonieri, Manfred Eppe, Marco Schorlemmer, Oliver Kutz, Rafael Peñaloza, Enric Plaza.
Upward Refinement for Conceptual Blending in Description Logic : An ASP-based Approach and Case Study in EL++.

11:50-12:15 Nuno Costa, Matthias Knorr, Joao Leite.
Extending NoHR for OWL 2 QL.

12:45-13:45 Lunch break

13:45-14:40 Invited talk: Andreas Pieris.
From Classical to Consistent Query Answering under Existential Rules.

14:40-15:30 Logic Programming 2

15:05-15:30 Ricardo Gonçalves. Combining rules and ontologies via parametrized logic programming.

15:30-16:00 Coffee break

16:00-17:30 Inconsistency Handling

16:00-16:25 Cristhian Ariel David Deagustini, Maria Vanina Martinez, Marcelo A. Falappa, Guillermo Simari. On the Influence of Incoherence in Inconsistency-tolerant Semantics for Datalog++.


16:50-17:15 Zied Bouraoui, Salem Benferhat, Sylvain Lagrue, Julien Rossit.
Merging Incommensurable Possibilistic DL-Lite Assertional Bases

17:15-17:45 Concluding discussion
Table of contents

Invited talks :

Andreas Pieris.
From Classical to Consistent Query Answering under Existential Rules.

Steven Schockaert
Extending Answer Set Programming using Generalized Possibilistic Logic.

Papers :

Riccardo Zese, Elena Bellodi, Evelina Lamma, Fabrizio Riguzzi.
Logic Programming Techniques for Reasoning with Probabilistic Ontologies.

Fabien Garreau, Laurent Garcia, Claire Lefèvre, Igor Stéphan.
∃-ASP

Veronica Dahl, Sergio Tessaris, Thom Fruehwirth.
Imperfect Querying through Womb Grammars plus Ontologies.

Roberto Confalonieri, Manfred Eppe, Marco Schorlemmer, Oliver Kutz, Rafael Peñaloza, Enric Plaza.
Upward Refinement for Conceptual Blending in Description Logic :
An ASP-based Approach and Case Study in EL++. 

Nuno Costa, Matthias Knorr, Joao Leite.
Extending NoHR for OWL 2 QL.

Cristina Feier, Thomas Eiter.
Reasoning with Forest Logic Programs Using Fully Enriched Automata.

Ricardo Gonçalves.
Combining rules and ontologies via parametrized logic programming.

Cristhian Ariel David Deagustini, Maria Vanina Martinez, Marcelo A. Falappa, Guillermo Simari.
On the Influence of Incoherence in Inconsistency-tolerant Semantics for Datalog⁺.

Badrinath Jayakumar, Rajshekhar Sunderraman.

Zied Bouraoui, Salem Benferhat, Sylvain Lagrue, Julien Rossit.
Merging Incommensurable Possibilistic DL-Lite Assertional Bases
From Classical to Consistent Query Answering
under Existential Rules

Andreas Pieris.
Institute of Information Systems,
Vienna University of Technology.

Abstract

We consider the well-known setting of ontology-based query answering. In real-life applications, involving large amount of data, it is possible that the data are inconsistent with the ontology. Since standard ontology languages adhere to the classical first-order logic semantics, inconsistencies are nothing else than logical contradictions. Therefore, the classical inference semantics fails terribly when faced an inconsistency, since everything is inferred from a contradiction. Querying inconsistent knowledge bases is an intriguing new problem that gave rise to a flourishing research activity in the KR community. In this talk, we focus on rule-based ontology languages, and we demonstrate the tight connection between classical and consistent query answering. More precisely, we focus on the standard inconsistency-tolerant semantics, namely, the ABox repair (AR) semantics, and we establish generic complexity results that allow us to obtain in a uniform way a relatively complete picture of the complexity of our problem. We also discuss sound approximations of the AR semantics, with the aim of achieving tractability of consistent query answering in data complexity.
Extending Answer Set Programming using Generalized Possibilistic Logic

Didier Dubois
Université Paul Sabatier, CNRS
IRIT, Toulouse, France
dubois@irit.fr

Henri Prade
Université Paul Sabatier, CNRS
IRIT, Toulouse, France
prade@irit.fr

Steven Schockaert
Cardiff University
United Kingdom
SchockaertS1@cardiff.ac.uk

Abstract

Answer set programming (ASP) is a form of logic programming in which negation-as-failure is defined in a purely declarative way, based on the notion of a stable model. This short paper briefly explains how a recent generalization of possibilistic logic (GPL) can be used to characterize the semantics of answer set programming. This characterization has several advantages over existing characterizations of the stable model semantics. First, unlike reduct-based approaches, it does not rely on a syntactic procedure: we can directly characterize answer sets based on the minimally specific models of a GPL theory. Second, GPL enables us to study extensions of ASP in an intuitive way: unlike in existing generalizations of ASP such as equilibri um logic and autoepistemic logic, all formulas in GPL have a meaning which is intuitively clear. Finally, being based on possibilistic logic, GPL offers a natural way of dealing with uncertainty in answer set programs.

1 Introduction

An answer set program is a set of rules of the form:

\[ a_1 \lor \ldots \lor a_n \leftarrow b_1 \land \ldots \land b_m \land \neg c_1 \land \ldots \land \neg c_r \] (1)

where \( a_1, \ldots, a_n, b_1, \ldots, b_m, c_1, \ldots, c_r \) are propositional literals, i.e., either atomic propositions from a given finite set \( \mathcal{A} \) or the negation of such atomic propositions. We call \( a_1 \lor \ldots \lor a_n \) the head of the rule while \( b_1 \land \ldots \land b_m \land \neg c_1 \land \ldots \land \neg c_r \) is called the body. An extended literal is a literal or an expression of the form \( \neg c_i \) (with \( c_i \) a literal). Intuitively, (1) states that if we cannot derive that any of \( c_1, \ldots, c_r \) are true and we can derive that all of \( b_1, \ldots, b_m \) are true, then we should assume that one of \( a_1, \ldots, a_n \) must be true. Formally, the semantics of an answer set program is defined in terms of the Gelfond-Lifschitz reduct [10]. In particular, given a set of literals \( M \), the reduct \( P^M \) of an answer set program \( P \) is defined as follows:

\[ P^M = \{ a_1 \lor \ldots \lor a_n \leftarrow b_1 \land \ldots \land b_m \mid M \cap \{ c_1, \ldots, c_r \} = \emptyset, (a_1 \lor \ldots \lor a_n \leftarrow b_1 \land \ldots \land b_m \land \neg c_1 \land \ldots \land \neg c_r) \in P \} \]

We say that \( M \) is a model of \( P^M \) if for each rule \( a_1 \lor \ldots \lor a_n \leftarrow b_1 \land \ldots \land b_m \leftarrow \) in \( P^M \) such that \( \{ b_1, \ldots, b_m \} \subseteq M \) it holds that \( M \cap \{ a_1, \ldots, a_n \} \neq \emptyset \). We say that \( M \) is an answer set of \( P^M \) if \( M \) is a model of \( P^M \) and there does not exist a model \( M' \) of \( P^M \) such that \( M' \subseteq M \). Intuitively, the condition \( \neg c_i \) is satisfied if \( c_i \) cannot be derived from the program. However, what literals can be derived depends on which assumptions we make about what other literals can be derived, which introduces a circular dependency. When using the Gelfond-Lifschitz reduct, this dependency is broken by making a guess \( M \) about what literals can be derived, and then verifying that \( M \) indeed coincides with the set of literals that can be derived.

A large number of equivalent characterizations of answer sets have been proposed [15]. For example, autoepistemic logic (AEL [16]), the logic of minimal belief and negation as failure (MBNF [14]) and equilibrium logic (EL [18]) can be used to define answer sets in a model-theoretic way. Moreover, MBNF and EL can be used to define answer sets of arbitrary propositional combinations of extended literals (e.g., containing disjunctions of rules, negation as failure in the head of rules, etc.).

In this paper, we show how a recent generalization of possibilistic logic (GPL [5; 7; 8]) can be used to characterize answer sets. This characterization has several advantages over existing characterizations, in particular w.r.t. how it enables us to extend ASP. For example, GPL has the advantage over MBNF and EL that its models can be naturally interpreted as the epistemic state of an agent, which allows us to give an intuitive interpretation to answer set programs in which \( \neg c_i \) means that “the agent does not know that \( c_i \) is true”. As a result, even when the syntax of ASP is extended to arbitrary propositional combinations of extended literals, the corresponding GPL formulas still have an intuitive meaning. In contrast, the intuitive meaning of EL formulas is not always clear. Moreover, since every propositional formula in GPL is encapsulated by a modal operator (see Section 2), we can distinguish between situations in which “the agent knows that either \( a \) or \( b \) holds” from “either the agent knows \( a \) or the agent knows \( b \)”, whereas EL can only model the latter.

The GPL characterization also ensures that all answer sets are minimal, i.e., that there cannot be two answer sets \( M_1 \) and \( M_2 \) such that \( M_1 \subseteq M_2 \). While this is true for any characterization of ASP when only rules of the form (1) are consid-
nered, existing characterizations do not guarantee minimality when negation-as-failure is allowed in the head of rules [13]. Finally, as the semantics of GPL is based on possibility distributions, the proposed characterization can be naturally extended to give a semantics to answer set programs in which rules can have uncertain conclusions.

In the next section, we briefly recall the syntax and semantics of GPL. Section 3 then explains how answer sets can be characterized using GPL, and how GPL can be used to define extensions of ASP. Finally, in Section 4 we consider uncertain answer set programs, and show how GPL can be used to define the possible uncertain programs of such sets.

2 Generalized possibilistic logic

Let $\Lambda_k = \{0, \frac{1}{k}, ..., 1\}$ be the considered set of certainty degrees and let $\Lambda_k^* = \Lambda_k \setminus \{0\}$. GPL formulas are then defined as follows:

- If $\lambda \in \Lambda_k^*$ and $\alpha$ is a propositional formula, then $N_\lambda(\alpha)$ is a GPL formula;
- if $\alpha$ and $\beta$ are GPL formulas, then so are $\neg \alpha$ and $\alpha \wedge \beta$.

As usual, we use $\alpha \rightarrow \beta$ and $\alpha \vee \beta$ as abbreviations for $\neg (\alpha \wedge \neg \beta)$ and $\neg (\neg \alpha \wedge \neg \beta)$. Furthermore, we write $\Pi_\lambda(\alpha)$ as an abbreviation for $\neg N_{\lambda - \lambda}(\neg \alpha)$, where $\lambda (\lambda) = 1 - \lambda + \frac{1}{2}$ for $\lambda \in \Lambda_k^*$. GPL is useful to reason about the knowledge of another agent. Intuitively $N_\lambda(\alpha)$ means that the agent knows $\alpha$ with certainty $\lambda$, while $\Pi_\lambda(\alpha)$ means that the agent considers $\alpha$ possible to the degree $\lambda$. An expression of the form $N_\lambda(\alpha)$ or $\Pi_\lambda(\alpha)$ will be called a meta-literal.

The semantics of GPL is defined in terms of possibility distributions. Let $\pi$ be a normalized possibility distribution over the set of possible worlds $\Omega$, i.e. $\pi$ is a mapping from $\Omega$ to $[0, 1]$ such that $\pi(\omega) = 1$ for at least one $\omega$ in $\Omega$. Then $\pi$ is said to satisfy the GPL formula $N_\lambda(\alpha)$, written $\pi \models N_\lambda(\alpha)$, iff

$$N(\alpha) = \min\{1 - \pi(\omega) | \omega \in \Omega, \omega \models \neg \alpha\} \geq \lambda$$

where $N$ denotes the necessity measure from possibility theory. The satisfaction relation $\models$ is extended to arbitrary GPL formulas in the usual way, i.e. $\pi \models \alpha \wedge \beta$ iff $\pi \models \alpha$ and $\pi \models \beta$, while $\pi \models \neg \alpha$ iff $\pi \not\models \alpha$. A possibility distribution $\pi$ is called a model of a set of GPL formulas $\Theta$ if $\pi$ satisfies every formula in $\Theta$. An axiomatization of GPL has been presented in [7].

Let $\pi_1$ and $\pi_2$ be two possibility distributions over $\Omega$. We say that $\pi_1$ is less specific than $\pi_2$, written $\pi_1 \preceq \pi_2$, if $\pi_1(\omega) \geq \pi_2(\omega)$ for all $\omega \in \Omega$. If $\pi_1 \preceq \pi_2$ and $\pi_1 \neq \pi_2$, we write $\pi_1 \prec \pi_2$. We say that $\pi$ is a minimally specific model of a set of GPL formulas $\Theta$ if $\pi$ is a model of $\Theta$ and there is no model $\pi'$ of $\Theta$ such that $\pi' \prec \pi$. Let $\alpha$ be a GPL formula and let $\Theta$ be a set of GPL formulas. The following three inference relations are considered in GPL:

- **standard entailment** $\Theta \models \alpha$ iff $\alpha$ is satisfied by every model of $\Theta$.
- **cautious entailment** $\Theta \models^c \alpha$ iff $\alpha$ is satisfied by all minimally specific models of $\Theta$.
- **brave entailment** $\Theta \models^b \alpha$ iff $\alpha$ is satisfied by at least one minimally specific model of $\Theta$.

The problems of checking whether $\Theta \models \alpha$, $\Theta \models^c \alpha$ and $\Theta \models^b \alpha$ hold are respectively coNP-complete, $\Pi_2^p$-complete and $\Sigma_2^p$-complete [8].

GPL generalizes standard possibilistic logic [4; 6] in that the latter only allows sets of meta-literals of the form $N_\lambda(\alpha)$, which are usually written as weighted propositional formulas of the form $(\alpha, \lambda)$. At the semantic level, a theory in possibilistic logic corresponds to a single possibility distribution, which is the unique minimally specific model of that theory, whereas theories in GPL can have several minimally specific models.

3 Characterizing and extending ASP using GPL

Given an answer set program $P$, we let $\Theta_P$ be the GPL theory which contains for each rule of the form $(1)$ in $P$ the following formula:

$$N_1(b_1) \wedge ... \wedge N_k(b_m) \wedge \Pi_1(\neg c_1) \wedge ... \wedge \Pi_1(\neg c_r) \rightarrow N_1(a_1) \vee ... \vee N_1(a_n)$$

(2)

In other words, the body of a rule of the form $(1)$ is satisfied if the agent knows each $b_i$ with maximal certainty and moreover the agent considers $\neg c_j$ fully possible for each $j$. Note that $\Pi_1(\neg c_j)$ is equivalent to $\neg N_1(c_j)$.

The transformation in (2) is by itself not enough, as ASP is based on the idea of forward chaining and in particular does not allow contrapositive reasoning (e.g. from the rule $a \leftarrow b$ and the fact $\neg a$ we should not derive $\neg b$). To see how forward chaining could be enforced using GPL, first note that there are three ways in which the formula (2) can be satisfied by a minimally specific model $\pi$ of $\Theta_P$:

1. one of the meta-literals $N_1(b_i)$ is not satisfied by $\pi$;
2. one of the meta-literals $\Pi_1(c_i)$ is not satisfied by $\pi$, i.e. $N_{\frac{1}{k}}(c_i)$ is satisfied by $\pi$;
3. one of the meta-literals $N_1(a_i)$ is satisfied by $\pi$.

The first case intuitively corresponds to an answer set which does not include $b_i$, i.e. to a situation in which the rule $(1)$ does not apply. The third case intuitively corresponds to an answer set in which $a_i$ has been included to make the rule $(1)$ satisfied, i.e. to a situation in which $a_i$ has been derived using (non-deterministic) forward chaining. The second case, however, intuitively corresponds to a contrapositive inference, i.e. $(1)$ has been satisfied by making $c_i$ true. The latter inference is not allowed in ASP and the second case should thus be excluded. To this end, we take advantage of the fact that it is only in the second case that certainty degrees other than 0 or 1 are needed. Note that here we do not use degrees for modelling uncertainty, but intuitively for differentiating between literals that are assumed to be true and literals that can effectively be derived. In particular, it turns out that answer sets correspond to the minimally specific models of $\Theta_P$ in which only the certainty degrees 0 and 1 occur. Formally, the requirement that only these certainty degrees occur is encoded.
by the GPL formula $\Phi$, defined as follows:

$$\Phi \equiv \bigwedge_{a \in At} \mathcal{N}_1(a) \lor \mathcal{N}_1(\neg a) \lor (\Pi_1(a) \land \Pi_1(\neg a)) \quad (3)$$

The formula $\Phi$ expresses that for every atom $a$, the agent is either fully certain about the truth value of $a$ (in which case $\mathcal{N}_1(a) \lor \mathcal{N}_1(\neg a)$ holds) or the agent is completely ignorant about $a$ (in which case $\Pi_1(a) \land \Pi_1(\neg a)$ holds). It turns out that the answer sets of $P$ correspond to the minimally specific models of $\Theta_P$ that satisfy $\Phi$. In particular, assuming that $k \geq 2$ (i.e., $|\Lambda_k| \geq 3$), it holds that $P$ has a consistent answer set if

$$\Theta_P \models \Phi$$

Moreover, for each consistent answer set $M$ of $P$, it holds that the following possibility distribution $\pi_M$ is a minimally specific model of $\Theta_P$ which satisfies $\Phi$:

$$\pi_M(\omega) = \begin{cases} 1 & \text{if } \omega \text{ satisfies every literal in } M \\ 0 & \text{otherwise} \end{cases}$$

Conversely, for every minimally specific model $\pi$ of $\Theta_P$ which satisfies $\Phi$, it holds that the following set of literals $M_\pi$ is an answer set of $P$:

$$M_\pi = \{ l \mid N(l) = 1 \}$$

where $N$ is the necessity measure induced by $\pi$. Note that it follows that a literal $l$ is included in at least one answer set of $P$ if

$$\Theta_P \models \Phi \land N_1(l)$$

and that $l$ is included in all answer sets of $P$ if

$$\Theta_P \models \Phi \rightarrow N_1(l)$$

This means in particular that the main reasoning tasks from ASP correspond to the standard forms of GPL inference.

This characterization of answer sets in GPL can be straightforwardly generalized to arbitrary propositional combinations of extended literals. When negation-as-failure in the head is considered, however, our characterization deviates from the existing characterizations in terms of EL and MBNF. This is illustrated in the next example.

**Example 1.** We consider a single atom $a$, in which case $\Omega = \{\emptyset, \{a\}\}$, where we identify models with the set of atoms they make true. Consider the formula $a \lor \neg a$, and the corresponding GPL encoding $\mathcal{N}_1(a) \lor \mathcal{I}_1(\neg a)$. Clearly, the latter formula has a unique minimally specific model $\pi$, defined by $\pi(\emptyset) = \pi(\{a\}) = 1$. In other words, the only answer set we find for $a \lor \neg a$ is the empty set. However, both the characterization of ASP in MBNF [14] and the characterization in EL [18] find two answer sets for this example, viz. $M_1 = \emptyset$ and $M_2 = \{a\}$.

As the intuition behind the stable model semantics is based on the idea of minimal commitment, the GPL semantics appears more natural.

The use of the modal operators $N_1$ in GPL allows us to further extend ASP. In the standard semantics of ASP, rules with disjunctions in the head intuitively correspond to a non-deterministic choice, e.g. $a \lor b \leftarrow c$ means that when the agent knows $c$ then either it knows $a$ or it knows $b$. When modelling epistemic reasoning, however, it often seems more natural to interpret $a \lor b$ as “the agent knows that either $a$ or $b$ is true (but may not know which is the case)”. This latter reading was referred to as weak disjunction in [2], where the inference problems resulting from interpreting ASP rules in this way have been investigated. Using GPL, the ASP rule $a \lor b \leftarrow c$ can be modelled as $\mathcal{N}_1(c) \rightarrow \mathcal{N}_1(a \lor b)$ when weak disjunction is considered, and as $\mathcal{N}_1(c) \rightarrow \mathcal{N}_1(a) \lor \mathcal{N}_1(b)$ otherwise.

Finally, the use of possibilistic logic enables us to consider uncertain answer set programs. This is discussed in more detail in the next section.

### 4 Modelling uncertain answer set programs

An uncertain ASP rule is an expression of the form

$$\lambda : a_1 \lor \ldots \lor a_n \leftarrow b_1 \land \ldots \land b_m \land \neg c_1 \land \ldots \land \neg c_r \quad (4)$$

where $\lambda$ is a certainty degree from $\Lambda_k$. An uncertain answer set program is a set of uncertain ASP rules. As has been observed in [1], there are two fundamentally different ways to interpret uncertain ASP rules. On the one hand, we may view $\lambda$ as reflecting the certainty that the corresponding ASP rule is valid. This interpretation leads us to view an uncertain ASP program as a possibility distribution over classical ASP programs; at the semantic level, we can then consider a possibility distribution over classical answer sets. On the other hand, we may view $\lambda$ as reflecting the certainty with which we can derive the head of the rule, given that its body is satisfied. This view leads to a semantics in which answer sets correspond to weighted epistemic states, which are modelled as possibility distributions. Note that a similar distinction is often made in first-order probabilistic logics [11] and in first-order conditional logics [9]. The former interpretation of uncertain ASP programs has been considered in [1] and [12]. The latter interpretation has been considered, among others, in [17], [2] and [3].

Modelling the former type of uncertain ASP programs in GPL would require nested modalities, encapsulating formulas of the form (2) with a modality of the form $\mathcal{N}_k$. As nested modalities are not allowed in GPL, this would require us to define a variant of GPL in which every (standard) GPL formula would be encapsulated by a modality, similar to how propositional formulas are encapsulated in standard GPL. At the semantic level, models would then correspond to possibility distributions over possibility distributions over possible worlds.

Here we focus on modelling the second type of uncertain ASP programs in GPL. Let us first consider rules without negation-as-failure:

$$l \vdash : a_1 \lor \ldots \lor a_n \leftarrow b_1 \land \ldots \land b_m$$

The corresponding GPL formula is given by

$$\bigwedge_{i=1}^l (\mathcal{N}_k(b_1) \land \ldots \land \mathcal{N}_k(b_m) \rightarrow \mathcal{N}_k(a_1) \lor \ldots \lor \mathcal{N}_k(a_n))$$
Let $P$ be an uncertain ASP program without negation-as-failure and let $Θ_P$ be the set of corresponding GPL formulas. It is not hard to see that a possibility distribution $π$ is a possibilistic answer set of $P$ in the sense of [2] iff $π$ is a minimally specific model of $Θ_P$. In absence of negation-as-failure, the semantics from [1] moreover coincides with the semantics from [17].

To deal with negation-as-failure, [2] and [17] rely on a generalization of the Gelfond-Lifschitz reduct. Consider a rule of the following form:

$$\frac{l}{k} : a_1 \lor ... \lor a_n \leftarrow b_1 \land ... \land b_m \land \neg c_1 \land ... \land \neg c_r$$

The reduct of this rule w.r.t. a possibility distribution $π$, according to the semantics from [2], is given by:

$$λ : a_1 \lor ... \lor a_n \leftarrow b_1 \land ... \land b_m$$

(5)

where $λ = \min(\frac{λ}{k}, \Pi(\neg c_1), ..., \Pi(\neg c_r))$, for $Π$ the possibility measure induced by $π$. The reduct considered in [17] boils down to choosing $λ = \frac{λ}{k}$ if $Π(\neg c_1) = ... Π(\neg c_r) = 1$ and $λ = 0$ otherwise (where rules whose certainty is 0 are simply omitted).

Using GPL, we can avoid the use of a reduct, if we assume that $k$ is even and only certainty degrees from $\{\frac{1}{2}, \frac{1}{2}, ..., 1\}$ are used in the uncertain ASP program $P$. We can always ensure that this assumption is satisfied by replacing the set of certainty degrees $Λ_k$ by the set $Λ_{2k}$, since every element from $Λ_k$ is equal to $\frac{1}{2}$ for some even value of $l$. The GPL theory $Θ_P$ corresponding to $P$ is then obtained by replacing every rule of the form (4) by the following formula:

$$\bigwedge_{i=1}^{l} (N_π(b_1) \land ... \land N_π(b_m) \land Π_π(\neg c_1) \land ... \land Π_π(\neg c_r))$$

$$\rightarrow N_π(a_1) \lor ... \lor N_π(a_n)$$

(6)

where we assume $λ = \frac{λ}{k}$. We again need to exclude models which intuitively rely on contrapositive reasoning. Similar as in Section 3, these correspond to minimally specific models $π$ which make (6) satisfied by making one of the meta-literals $Π_π(\neg c_1)$ false, i.e. by making a meta-literal $N_{π+1}(c_1)$ true. Noting that $k - i + 1$ is odd iff $i$ is even, we find that it suffices to exclude models in which certainty degrees $\frac{1}{2}$ are used with $l$ odd. Such models can be avoided by considering the following GPL formula $Φ_k$:

$$Φ_k \equiv \bigwedge_{α \in Δ} \bigwedge_{l=1}^{\frac{λ}{k}} \left( N_{π+1}(α) \rightarrow N_{π}(α) \right)$$

$$\land \left( N_{π-1}(α) \rightarrow N_{π}(\neg α) \right)$$

Note that $Φ_k$ is equivalent to the formula $Φ$ defined in (3). We propose the following definition: $φ$ is a possibilistic answer set of $P$ iff $π$ is a minimally specific model of $Θ_P$ which satisfies $Φ_k$. As in Section 3, we can then formulate the main reasoning tasks for possibilistic ASP in terms of standard entailment in GPL. For example, checking whether the certainty of $α$ is at least $\frac{1}{l}$ (with $l$ even) in every possibilistic answer set of $P$ corresponds to:

$$Θ_P \models^c Φ_k \rightarrow N_{π}(α)$$

We can show that the proposed definition of possibilistic answer set corresponds to the reduct-based definition from [2].

## 5 Conclusions

We have shown how generalized possibilistic logic (GPL) can be used to characterize answer sets without the need for a reduct operation, and how this characterization allows us to consider a range of different extensions of ASP in a natural way. In particular, the GPL characterization enables us to define answer sets for arbitrary propositional combinations of extended literals (while keeping the intuition of minimal commitment), for modelling weak disjunction, and for defining answer sets of uncertain programs.

## Acknowledgment

Steven Schockaert has been supported by a grant from the Leverhulme Trust (RPG-2014-164).

## References


Logic Programming Techniques for Reasoning with Probabilistic Ontologies

Riccardo Zese, Elena Bellodi, Evelina Lamma
Dipartimento di Ingegneria
University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[riccardo.zese,elena.bellodi,evelina.lamma]@unife.it

Fabrizio Riguzzi
Dipartimento di Matematica e Informatica
University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
fabrizio.riguzzi@unife.it

Abstract

The increasing popularity of the Semantic Web drove to a widespread adoption of Description Logics (DLs) for modeling real world domains. To help the diffusion of DLs, a large number of reasoning algorithms have been developed. Usually these algorithms are implemented in procedural languages such as Java or C++. Most of the reasoners exploit the tableau algorithm which has to manage non-determinism, a feature that is not easy to handle using such languages. Reasoning on real world domains also requires the capability of managing probabilistic and uncertain information. We thus present TRILL, for “Tableau Reasoner for description Logics in proLog”, that implements a tableau algorithm and is able to return explanations for queries and their corresponding probability, and TRILLP, for “TRILL powered by Pinpointing formulas”, which is able to compute a Boolean formula representing the set of explanations for a query. This approach can speed up the process of computing the probability. Prolog non-determinism allows us to easily handle the tableau’s non-deterministic expansion rules.

Introduction

The Semantic Web aims at making information regarding real world domains available in a form that is understandable by machines (Hitzler, Krötzsch, and Rudolph 2009). The World Wide Web Consortium is working for realizing this vision by supporting the development of the Web Ontology Language (OWL), a family of knowledge representation formalisms for defining ontologies. OWL is based on Description Logics (DLs), a set of languages that are restrictions of first order logic (FOL) with decidability and, in some cases, low complexity. For example, the OWL DL sublanguage is based on the expressive SHOIN(D) DL while OWL 2 corresponds to the SROIQ(D) DL (Hitzler, Krötzsch, and Rudolph 2009). Moreover, uncertain information is intrinsic to real world domains, thus the combination of probability and logic theories becomes of foremost importance.

In order to fully support the development of the Semantic Web, efficient DL reasoners, such as Pellet, RacerPro, FaCT++ and HermiT, are able to extract implicit information from the modeled ontologies. Despite the large number of available reasoners, only few of them are able to manage probabilistic information as well. One of the most common approaches for reasoning is the tableau algorithm that exploits some non-deterministic expansion rules. This requires the developers to implement a search strategy in an or-branching search space. Moreover, if we want to compute the probability of a query, the algorithm has to compute all the explanations for the query, thus it has to explore all the non-deterministic choices taken during the execution.

In this paper, we present the systems TRILL for “Tableau Reasoner for description Logics in proLog” and TRILLP for “TRILL powered by Pinpointing formulas”. They are tableau reasoners for the SHOIN DL and for the ALC DL respectively, both implemented in Prolog. Prolog search strategy is exploited for taking into account the non-determinism of the tableau rules. They use the Thea2 library (Vassiliadis, Wielemaker, and Mungall 2009) for parsing OWL in its various dialects. Thea2 translates OWL files into a Prolog representation in which each axiom is mapped into a fact. TRILL and TRILLP can check the consistency of a concept and the entailment of an axiom from an ontology, and can compute the probability of the entailment following DISPONTE (Riguzzi et al. 2012), a semantics for probabilistic ontologies. The availability of a Prolog implementation of a DL reasoner will also facilitate the development of probabilistic reasoners that can integrate probabilistic logic programming with probabilistic DLs. In probabilistic logic programming one of the most widespread approaches is the Distribution Semantics (Sato 1995), on which DISPONTE is based. Since our systems follow DISPONTE, they are easily extensible to take into account this integration.

Description Logics

DLs are knowledge representation formalisms that are at the basis of the Semantic Web (Baader et al. 2003; Baader, Horrocks, and Sattler 2008) and are used for modeling ontologies. They possess nice computational properties such as decidability and/or low complexity.

Usually, DLs’ syntax is based on concepts and roles which correspond respectively to sets of individuals and sets of pairs of individuals of the domain. We first briefly describe ALC and then SHOIN(D).

Let C, R and I be sets of atomic concepts, atomic roles and individuals, respectively. Concepts are defined by induc-
tion as follows. Each \( C \subseteq \mathbb{D} \) is a concept, \( \perp \) and \( \top \) are concepts. If \( C_1, C_2 \) are concepts and \( R \subseteq \mathbb{R} \), then \( (C_1 \cap C_2), (C_1 \cup C_2) \) and \( \neg C \) are concepts, as well as \( \exists R.C \) and \( \forall R.C \). A TBox \( T \) is a finite set of concept inclusion axioms \( C \subseteq D \), where \( C \) and \( D \) are concepts. We use \( \mathbb{C} \equiv \mathbb{D} \) to abbreviate the conjunction of \( C \subseteq D \) and \( D \subseteq C \). An ABox \( A \) is a finite set of concept membership axioms \( a : C \), role membership axioms \( (a, b) : R \), equality axioms \( a = b \) and inequality axioms \( a \neq b \), where \( C, R, a, b \in \mathbb{R} \) and \( a, b \in \mathbb{I} \). A knowledge base (KB) \( K = (T, A) \) consists of a TBox \( T \) and an ABox \( A \) and is usually assigned a semantics in terms of interpretations \( I = (\Delta^I, \cdot^I) \), where \( \Delta^I \) is a non-empty domain and \( \cdot^I \) is the interpretation function that assigns an element in \( \Delta^I \) to each \( a \in I \), a subset of \( \Delta^I \) to each \( C \in A \) and a subset of \( \Delta^I \times \Delta^I \) to each \( R \in \mathbb{R} \).

The mapping \( \cdot^I \) is extended to all concepts (where \( R^I(x) = \left\{ y | (x, y) \in R^I \right\} \)) as:

\[
\begin{align*}
\tau^I & = \Delta^I \\
\bot^I & = \emptyset \\
(C_1 \cap C_2)^I & = C_1^I \cap C_2^I \\
(C_1 \cup C_2)^I & = C_1^I \cup C_2^I \\
(\neg C)^I & = \Delta^I \setminus C^I \\
(R,C)^I & = \left\{ x \in \Delta^I | R^I(x) \subseteq C^I \right\} \\
(R\cdot C)^I & = \left\{ x \in \Delta^I | R^I(x) \cap C^I \neq \emptyset \right\}
\end{align*}
\]

In the following we describe SHOIN(D) showing what it adds to ALC. A role is either an atomic role \( R \in \mathbb{R} \) or the inverse \( R^\perp \) of an atomic role \( R \in \mathbb{R} \). We use \( \mathbb{R}^\perp \) to denote the set of all inverses of roles in \( \mathbb{R} \). An RBox \( R \) consists of a finite set of transitivity axioms \( \text{Trans}(R) \), where \( R \in \mathbb{R} \), and role inclusion axioms \( R \subseteq S \), where \( R, S \in \mathbb{R} \cup \mathbb{R}^\perp \).

If \( a \in I \), then \( \{ a \} \) is a concept called nominal, and if \( C, C_1 \) and \( C_2 \) are concepts and \( R \subseteq \mathbb{R} \), then \( \geq nR \) and \( \leq nR \) for an integer \( n \geq 0 \) are also concepts. A SHOIN(D) KB \( K = (T, R, A) \) consists of a TBox \( T \), an RBox \( R \) and an ABox \( A \).

The mapping \( \cdot^I \) is extended to all new concepts (where \( \# x \) denotes the cardinality of the set \( X \)) as:

\[
\begin{align*}
(R^\perp)^I & = \left\{ (y, x) | (x, y) \in R^I \right\} \\
\{ a \}^I & = \{ a \}^2 \\
(\geq nR)^I & = \left\{ x \in \Delta^I | \# R^I(x) \geq n \right\} \\
(\leq nR)^I & = \left\{ x \in \Delta^I | \# R^I(x) \leq n \right\}
\end{align*}
\]

SHOIN(D) allows for the definition of datatype roles, i.e., roles that map an individual to an element of a datatype such as integers, floats, etc. Then new concept definitions involving datatype roles are added that mirror those involving roles introduced above. We also assume that we have predicates over the datatypes.

SHOIN(D) is decidable iff there are no number restrictions on roles which are transitive or have transitive subroles.

A query \( Q \) over a KB \( K \) is usually an axiom for which we want to test the entailment from the KB, written \( K \models Q \). The entailment test may be reduced to checking the unsatisfiability of a concept in the knowledge base, i.e., the emptiness of the concept. For example, the entailment of the axiom \( C \subseteq D \) may be tested by checking the unsatisfiability of the concept \( C \cap \neg D \) while the entailment of the axiom \( a : C \) may be tested by checking the unsatisfiability of \( a : \neg C \).

**Example 1** The following KB is inspired by the ontology people+pets (Patel-Schneider, Horrocks, and Bechhofer 2003):

\[
\begin{align*}
\exists \text{hasAnimal.Pet} & \sqsubseteq \text{NatureLover} \\
\text{fluffy} : \text{Cat} \\
\text{tom} : \text{Cat} \\
\text{kevin, fluffy} : \exists \text{hasAnimal} \\
\text{kevin, tom} : \exists \text{hasAnimal}
\end{align*}
\]

It states that individuals that own an animal which is a pet are nature lovers and that kevin owns the animals fluffy and tom, which are cats. Moreover, cats are pets. The KB entails the query \( Q = \text{kevin : NatureLover} \).

**Querying KBs: The Tableau Algorithm**

In order to answer queries to DL KBs, a tableau algorithm can be used. A tableau is an ABox represented as a graph \( G \) where each node corresponds to an individual \( a \) and is labeled with the set of concepts \( L(a) \) to which \( a \) belongs. Each edge \( (a, b) \) in the graph is labeled with the set of roles \( L_t(a, b) \) to which the couple \( (a, b) \) belongs. A tableau algorithm proves an axiom by refutation, starting from a tableau that contains the negation of the axiom. For example, the axiom \( C \subseteq D \) can be proved by showing that \( C \cap \neg D \) is empty, while, if the query is a class assertion, \( C(a) \), we add \( \neg C \) to the label of \( a \). For testing the inconsistency of a concept \( C \) we have to test the emptiness of \( C \) by adding a new anonymous node \( a \) to the tableau whose label contains \( C \). Then, the tableau algorithm repeatedly applies a set of consistency preserving tableau expansion rules until a clash (i.e., a contradiction, for example, a concept \( C \) and a node \( a \) where \( C \) and \( \neg C \) are present in the label of \( a \)) is detected or a clash-free graph is found to which no more rules are applicable. If no clashes are found, the tableau represents a model for the negation of the query, which is thus not entailed.

Each expansion rule updates as well a tracing function \( \tau \), which associates labels of nodes and edges with a subset of the axioms of the KB. \( \tau \) is initialized as the empty set for all the elements of its domain except for the axioms \( \tau(C, a) \) and \( \tau(R, (a, b)) \) to which the values \( \{ a : C \} \) and \( \{(a, b) : R\} \) are assigned if \( a : C \) and \( (a, b) : R \) are in the ABox respectively.

The tableau expansion rules for SHOIN(D) are shown in Figure 1, where the rules for the ALC DL are marked by (*).

For ensuring the termination of the algorithm, a special condition known as blocking (Kalyanpur 2006) is used. In a tableau a node \( x \) can be a nominal node if its label \( L(x) \) contains a nominal or a blockable node. If there is an edge \( e = (x, y) \) then \( y \) is a successor of \( x \) and \( x \) is a predecessor of \( y \). Ancestor is the transitive closure of predecessor while descendant is the transitive closure of successor. A node \( y \) is called an R-neighbour of a node \( x \) if \( x \) is a successor of \( x \) and \( R \in L((x, y)) \), where \( R \in \mathbb{R} \).

An R-neighbour of \( x \) is safe if (i) \( x \) is blockable or if (ii) \( x \) is a nominal node and \( y \) is not blocked. Finally, a node
x is blocked if it has ancestors \( x_0, y \) and \( y_0 \) such that all the following conditions are true: (1) \( x \) is a successor of \( x_0 \) and \( y \) is a successor of \( y_0 \), (2) \( y, x \) and all nodes on the path from \( y \) to \( x \) are blockable. (3) \( \mathcal{L}(x) = \mathcal{L}(y) \) and \( \mathcal{L}(x_0) = \mathcal{L}(y_0) \). (4) \( \mathcal{L}(x_0, x) = \mathcal{L}(y_0, y) \). In this case, we say that \( y \) blocks \( x \). A node is blocked also if it is blockable and all its predecessors are blocked; if the predecessor of a safe node \( x \) is blocked, then we say that \( x \) is indirectly blocked.

Finding Explanations

The problem of finding explanations for a query has been investigated by various authors (Schlobach and Cornet 2003; Kalyanpur 2006; Halaschek-Wiener, Kalyanpur, and Parsia 2006; Kalyanpur et al. 2007). It was called axiom pinpointing in (Schlobach and Cornet 2003) and considered as a non-standard reasoning service useful for tracing derivations and debugging ontologies. In particular, minimal axiom sets or MinAs for short, also called explanations, are introduced in (Schlobach and Cornet 2003).

Definition 1 (MinA) Let \( K \) be a knowledge base and \( Q \) an axiom that follows from it, i.e., \( K \models Q \). We call a set \( M \subseteq K \) a minimal axiom set or MinA for \( Q \) in \( K \) if \( M \models Q \) and it is minimal w.r.t. set inclusion.

The problem of enumerating all MinAs is called MIN-A-ENUM in (Schlobach and Cornet 2003). ALL-MINAS(\( Q, K \)) is the set of all MinAs for query \( Q \) in the knowledge base \( K \).

The tableau algorithm returns a single MinA using the tracing function. To solve MIN-A-ENUM, reasoners written in imperative languages, like Pellet (Sirin et al. 2007), have to implement a search strategy in order to explore the entire search space of the possible explanations. In particular, Pellet, that is written entirely in Java, uses Reiter’s hitting set algorithm (Reiter 1987). The algorithm, described in detail in (Kalyanpur 2006), starts from a MinA \( S \) and initializes a labeled tree called Hitting Set Tree (HST) with \( S \) as the label of its root \( v \). Then it selects an arbitrary axiom \( E \) in \( S \), it removes it from \( K \), generating a new knowledge base \( K' = K - \{ E \} \), and tests the unsatisfiability of \( C \) w.r.t. \( K' \). If \( C \) is still unsatisfiable, we obtain a new explanation. The algorithm adds a new node \( w \) and a new edge \( (v, w) \) to the tree, then it assigns this new explanation to the label of \( w \) and the axiom \( E \) to the label of the edge. The algorithm repeats this process until the unsatisfiability test returns negative: in that case the algorithm labels the new node with \( OK \), makes it a leaf, backtracks to a previous node, selects a different axiom to be removed from the KB and repeats these operations until the HST is fully built. The algorithm also eliminates extraneous unsatisfiability tests based on previous results: once a path leading to a node labeled \( OK \) is found, any superset of that path is guaranteed to be a path leading to a node where \( C \) is satisfiable, and thus no additional unsatisfiability test is needed for that path, as indicated by an \( X \) in the node label. When the HST is fully built, all leaves of the tree are labeled with \( OK \) or \( X \). The distinct non leaf nodes of the tree collectively represent the set ALL-MINAS(\( C, K \).

In (Baader and Peñaloza 2010a; 2010b) the authors consider the problem of finding a pinpointing formula instead of ALL-MINAS(\( Q, K \)). The pinpointing formula is a monotone Boolean formula in which each Boolean variable corresponds to an axiom of the KB. This formula is built using the variables and the conjunction and disjunction connectives. It compactly encodes the set of all MinAs. Let us assume that each axiom \( E \) of a KB \( K \) is associated with a propositional variable, indicated with \( \text{var}(E) \). The set of all propositional variables is indicated with \( \text{var}(K) \). A valuation \( \nu \) of a monotone Boolean formula is the set of propositional variables that are true. For a valuation \( \nu \subseteq \text{var}(K) \), let \( K_{\nu} := \{ t \in K | \text{var}(t) \in \nu \} \).

Definition 2 (Pinpointing formula) Given a query \( Q \) and a KB \( K \), a monotone Boolean formula \( \phi \) over \( \text{var}(K) \) is called a pinpointing formula for \( Q \) if for every valuation \( \nu \subseteq \text{var}(K) \) it holds that \( K_{\nu} \models Q \) iff \( \nu \) satisfies \( \phi \).

In Lemma 2.4 of (Baader and Peñaloza 2010b) the authors proved that the set of all MinAs can be obtained by transforming the pinpointing formula into DNF and removing disjuncts involving other disjuncts. The example below illustrates axiom pinpointing and the pinpointing formula.

Example 2 (Pinpointing formula) Consider the KB of Example 1. We associate Boolean variables to axioms as follows:

\[ F_1 = \exists \text{hasAnimal}.\text{Pet} \subseteq \text{NatureLover} \]
\[ F_2 = (\text{kevin}, \text{fluffy}) : \text{hasAnimal} \]
\[ F_3 = (\text{kevin}, \text{tom}) : \text{hasAnimal} \]
\[ F_4 = \text{fluffy} : \text{Cat} \]
\[ F_5 = \text{tom} : \text{Cat} \]
\[ F_0 = \text{Cat} \subseteq \text{Pet}. \]

Let \( Q = \text{kevin} : \text{NatureLover} \) be the query, then ALL-MINAS(\( Q, K \)) = \{ \{ F_2, F_3, F_0, F_1 \}, \{ F_3, F_5, F_0, F_1 \} \}, while the pinpointing formula is \( ( ( F_2 \land F_3 ) \lor ( F_0 \land F_3 ) ) \land F_0 \land F_1 \).

In the following, we briefly define how a tableau algorithm can be modified to find the pinpointing formula. For more details and formal definitions see (Baader and Peñaloza 2010b).

Given a KB \( K \), the modified algorithm associates a label \( \text{lab}(a) \) that is a monotone Boolean formula over \( \text{var}(K) \) to every assertion \( a \). For deciding whether a rule is applicable we have to control the insertability of the new assertion. Let \( A \) be a set of labeled assertions and \( \psi \) a monotone Boolean formula, the assertion \( a \) is \( \psi \)-insertable into \( A \) if either \( a \notin A \) or \( a \in A \) but \( \psi \not\models \text{lab}(a) \). Given a set \( B \) of assertions and a set \( A \) of labeled assertions, the set of \( \psi \)-insertable elements of \( B \) into \( A \) is defined as \( \text{ins}_{\psi}(B,A) := \{ b \mid \exists b \in B \mid b \models \psi \text{-insertable into } A \} \). For deciding the applicability of a rule we need also to give the definition of substitution. A substitution is a mapping \( \rho : V \rightarrow D \), where \( V \) is a finite set of variables and \( D \) is a countably infinite set of constants that contains all the individuals in the KB and all the fresh individuals created by the application of the rules. Variables are seen as placeholder for individuals in the assertions. For example, an assertion can be \( C(x) \) or \( R(x,y) \) where \( C \) is a concept, \( R \) is a role and \( x \) and \( y \) are variables. In this case, let \( C(x) \) be an assertion with the variable \( x \) and \( \rho : x \rightarrow a \) a substitution, then \( C(x)\rho \) denotes the assertion obtained by replacing the variable with its \( \rho \)-image, i.e. \( C(a) \). A rule is of the form \( (B_0, S) \rightarrow \{ B_1, ..., B_m \} \) where \( B_i \) are finite
Deterministic rules:

\[ \text{unfold (⇤): if } A \in L(a), \text{ A atomic and } (A \sqsubseteq D) \in K, \text{ then } \]
\[ \text{if } D \notin L(a), \text{ then } \]
\[ \text{Add}(D, L(a)) \]
\[ \tau(D, a) := (\tau(A, a) \cup \{A \sqsubseteq D\}) \]

\[ \rightarrow \text{CE (⇤): if } (C \sqsubseteq D) \in K, \text{ with } C \text{ not atomic, a not blocked, then } \]
\[ \text{if } (-C \sqsubseteq D) \notin L(a), \text{ then } \]
\[ \text{Add}((-C \sqsubseteq D), a) \]
\[ \tau((-C \sqsubseteq D), a) := \{C \sqsubseteq D\} \]

\[ \rightarrow \Omega (⇤): \text{ if } (C_1 \cap C_2) \in L(a), \text{ a is not indirectly blocked, then } \]
\[ \text{if } \{C_1, C_2\} \notin L(a), \text{ then } \]
\[ \text{Add}\{C_1, C_2\}, a \]
\[ \tau(C_i, a) := \tau(C_i \cap a) \]

\[ \rightarrow \exists (⇤): \text{ if } \exists S.C \in L(a), \text{ a is not blocked, then } \]
\[ \text{if } a \text{ has no S-neighbor } b \text{ with } C \in L(b), \text{ then } \]
\[ \text{create new node } b, \text{ Add}(S, \{a, b\}), \text{ Add}(C, b) \]
\[ \tau(C, b) := \tau(\{S, C\}, a) \]
\[ \tau(S, a) := \tau(\{S, C\}, a) \]

\[ \rightarrow \forall (⇤): \text{ if } \forall (S, C) \in L(a), \text{ a is not indirectly blocked and there is an } S \text{-neighbor } b \text{ of } a, \text{ then } \]
\[ \text{if } C \notin L(b), \text{ then } \]
\[ \text{Add}(C, b) \]
\[ \tau(C, b) := \tau(\{S, C\}, a) \cup \tau(S, a) \]

\[ \rightarrow \forall (⇤): \text{ if } \forall (S, C) \in L(a), \text{ a is not indirectly blocked and there is an } S \text{-neighbor } b \text{ of } a, \text{ then } \]
\[ \text{if } C \notin L(b), \text{ then } \]
\[ \text{Add}(C, b) \]
\[ \tau(C, b) := \tau(\{S, C\}, a) \cup \tau(S, a) \]

\[ \rightarrow \exists (⇤): \text{ if } \exists S.C \in L(a), \text{ a is not blocked, then } \]
\[ \text{if } a \text{ has no S-neighbor } b \text{ with } C \in L(b), \text{ then } \]
\[ \text{create new node } b, \text{ Add}(S, \{a, b\}), \text{ Add}(C, b) \]
\[ \tau(C, b) := \tau(\{S, C\}, a) \]
\[ \tau(S, a) := \tau(\{S, C\}, a) \]

\[ \rightarrow \forall (⇤): \text{ if } \forall (S, C) \in L(a), \text{ a is not indirectly blocked and there is an } S \text{-neighbor } b \text{ of } a, \text{ then } \]
\[ \text{if } C \notin L(b), \text{ then } \]
\[ \text{Add}(C, b) \]
\[ \tau(C, b) := \tau(\{S, C\}, a) \cup \tau(S, a) \]

\[ \rightarrow \exists (⇤): \text{ if } \exists S.C \in L(a), \text{ a is not blocked, then } \]
\[ \text{if } a \text{ has no S-neighbor } b \text{ with } C \in L(b), \text{ then } \]
\[ \text{create new node } b, \text{ Add}(S, \{a, b\}), \text{ Add}(C, b) \]
\[ \tau(C, b) := \tau(\{S, C\}, a) \]
\[ \tau(S, a) := \tau(\{S, C\}, a) \]

\[ \rightarrow \forall (⇤): \text{ if } \forall (S, C) \in L(a), \text{ a is not indirectly blocked and there is an } S \text{-neighbor } b \text{ of } a, \text{ then } \]
\[ \text{if } C \notin L(b), \text{ then } \]
\[ \text{Add}(C, b) \]
\[ \tau(C, b) := \tau(\{S, C\}, a) \cup \tau(S, a) \]

Non-deterministic rules:

\[ \rightarrow \cup (⇤): \text{ if } (C_1 \sqcup C_2) \in L(a), \text{ a is not indirectly blocked, then } \]
\[ \text{if } \{C_1, C_2\} \cap L(a) = \emptyset, \text{ then } \]
\[ \text{Generate graphs } \bar{G}_i := G \text{ for each } i \in \{1, 2\} \]
\[ \text{Add}(C_i, a) \text{ in } \bar{G}_i \text{ for each } i \in \{1, 2\} \]
\[ \tau(C_i, a) := \tau(\{C_i \sqcup C_2\}, a) \]

\[ \rightarrow \leq (⇤): \text{ if } (\leq n S) \in L(a), \text{ a is not indirectly blocked, and there are } m \text{ S-neighbors } b_1, \ldots, b_m \text{ of } a \text{ with } m > n, \text{ then } \]
\[ \text{Generate a graph } G' \]
\[ \tau(\text{Merge}(b_i, b_j)) := \tau(\{\leq n S\}, a) \cup \tau(S, \{a, b_1\}) \ldots \cup \tau(S, \{a, b_m\}) \]
\[ \text{if } b_j \text{ is a nominal node, then } \text{Merge}(b_i, b_j) \text{ in } G' \]
\[ \text{else if } b_i \text{ is a nominal node or ancestor of } b_j, \text{ then } \text{Merge}(b_j, b_i) \]
\[ \text{else Merge}(b_i, b_j) \text{ in } G' \]
\[ \text{if } b_i \text{ is merged into } b_j, \text{ then for each concept } C_i \text{ in } L(b_i), \]
\[ \tau(C_i, b_j) := \tau(C_i, b_i) \cup \tau(\text{Merge}(b_i, b_j)) \]

Figure 1: TRILL tableau expansion rules; the subset of rules marked by (⇤) is employed by TRILLP.
set of assertions and $S$ is a finite set of axioms. A rule is applicable with a substitution $\rho$ on the variable occurring in $B_0$ if $S \subseteq K$, $B_0 \rho \subseteq A$, where $A$ is the set of assertions contained in the ABox and found during inference, and, for every $1 \leq i \leq m$ and every substitution $\rho'$ on the variables occurring in $B_0 \cup B_i$, we have $\text{ins}_0(B_0 \rho', A) \neq \emptyset$, where $\psi ::= \bigvee_{b \in B_0} \text{lab}(bp) \land \bigvee_{s \in S} \text{lab}(s)$. Moreover, except for the unfold rule, the node $N$ to which the rule is applicable is not (indirectly) blocked. When the tableau is fully built, the algorithm conjoins the labels of each clash for building the final pinpointing formula.

**TRILL and TRILL$^P$**

Both TRILL and TRILL$^P$ implement a tableau algorithm, the first solves MIN-A-ENUM while the second computes the pinpointing formula representing the set of MinAs. They can answer concept and role membership queries, subsumption queries and can test the unsatisfiability of a concept of the KB or the inconsistency of the entire KB. TRILL and TRILL$^P$ are implemented in Prolog, so the management of the non-determinism of the rules is delegated to the language.

We use the Thea2 library (Vassiliadis, Wiemekamer, and Mungall 2009) for converting OWL DL KBs into Prolog. Thea2 performs a direct translation of the OWL axioms into Prolog facts. For example, a simple subclass axiom between two named classes $\text{Cat} \sqsubseteq \text{Pet}$ is written using the subClassOf/2 predicate as subClassOf('Cat', 'Pet'). For more complex axioms, Thea2 exploits the list construct of Prolog, so the axiom $\text{NatureLover} \equiv \text{PetOwner} \sqcup \text{GardenOwner}$ becomes equivalentClasses([['NatureLover'], unionOf([['PetOwner'], 'GardenOwner'])])

In order to represent the tableau, TRILL and TRILL$^P$ use a pair $\text{Tableau} = (A, T)$, where $A$ is a list containing information about individuals and class assertions with the corresponding value of the tracing function. The tracing function stores a fragment of the knowledge base in TRILL and the pinpointing formula in TRILL$^P$. $T$ is a triple $(G, \text{RBN}, \text{RBR})$ in which $G$ is a directed graph that contains the structure of the tableau, $\text{RBN}$ is a red-black tree (a key-value dictionary), where a key is a couple of individuals and its value is the set of the labels of the edge between the two individuals, and $\text{RBR}$ is a red-black tree, where a key is a role and its value is the set of couples of individuals that are linked by the role. This representation allows to quickly find the information needed during the execution of the tableau algorithm. For managing the blocking system we use a predicate for each blocking state: nominal/2, blockable/2, blocked/2, indirectly_blocked/2 and safe/3. Each predicate takes as arguments the individual $\text{Ind}$ and the tableau $(A, T)$; safe/3 takes as input also the role $R$. For each individual $\text{Ind}$ in the ABox, we add the atom nominal(Ind) to $A$, then every time we have to check the blocking status of an individual we call the corresponding predicate that returns the status by checking the tableau.

Deterministic and non-deterministic tableau expansion rules are treated differently. Non-deterministic rules are implemented by a predicate rule_name(Tab, TabList) that, given the current tableau Tab, returns the list of tableaux TabList created by the application of the rule to Tab. Deterministic rules are implemented by a predicate rule_name(Tab, Tab1) that, given the current tableau Tab, returns the tableau Tab1 obtained by the application of the rule to Tab. Expansion rules are applied in order by apply_all_rules/2, first the non-deterministic ones and then the deterministic ones. The predicate apply_nondet_rules/3 takes as input the list of non-deterministic rules and the current tableau and returns a tableau obtained by the application of one of the rules. It is called as apply_nondet_rules(RuleList, Tab, Tab1) and is shown in Figure 2.

If a non-deterministic rule is applicable, the list of tableaux obtained by its application is returned by the predicate corresponding to the applied rule, a cut is performed to avoid backtracking to other rule choices and a tableau from the list is non-deterministically chosen with the member/2 predicate. If no non-deterministic rule is applicable, deterministic rules are tried sequentially with the predicate apply_det_rules/3, shown in Figure 2, that is called as apply_det_rules(RuleList,Tab,Tab1). It takes as input the list of deterministic rules and the current tableau and returns a tableau obtained with the application of one of the rules. After the application of a deterministic rule, a cut avoids backtracking to other possible choices for the deterministic rules. If no rule is applicable, the input tableau is returned and rule application stops, otherwise a new round of rule application is performed.

In Figure 1, the symbol $(*)$ denotes the rules shared by TRILL and TRILL$^P$. In these rules, the operator $\sqcup$ for $\tau$ means union between two sets in TRILL, while in TRILL$^P$ it joins two Boolean formulas with the $\lor$ Boolean operator. Moreover, when a concept is already present in a node label, TRILL checks whether to update the tracing function by verifying that the corresponding set of axioms is not a subset of $\tau$, while TRILL$^P$ performs a $\psi$-insertability test.

In case the assertion $a$ to be inserted is already associated with the corresponding individual, TRILL$^P$ tests its $\psi$-insertability by means of a satisfiability solver. In particular, it conjoins the negation of label(a) with the Boolean formula associated to the individual in the tableau, and tests the satisfiability of such formula. If the test returns true, the two Boolean formulas are combined with the $\lor$ Boolean operator.

**Computing the Probability**

The aim of our work is to implement algorithms which can compute the probability of queries to KBs following DISPONTE (Riguzzi et al. 2012). DISPONTE applies the distribution semantics (Sato 1995) of probabilistic logic programming to DLs. A program following this semantics defines a probability distribution over normal logic programs called worlds. Then the distribution is extended to a joint distribution over worlds and queries from which the probability of a query is obtained by marginalization.

In DISPONTE, a probabilistic knowledge base $K$ con-
apply_all_rules(Tab,Tab2):-
  apply_nondet_rules([...],Tab,Tab1),
  (Tab = Tab1 -> Tab2 = Tab1 ;
  apply_all_rules(Tab,Tab2)).

apply_nondet_rules([],Tab,Tab1):-
  apply_det_rules(T,Tab,Tab1).

apply_nondet_rules([H|T],Tab,Tab1):-
  apply_nondet_rules(T,Tab,Tab1),
  C=..[H,Tab,L],
  member(Tab1,L),
  call(C),!

apply_all_rules(Tab,Tab2):-
  apply_nondet_rules([...],Tab,Tab1),
  (Tab = Tab1 -> Tab2 = Tab1 ;
  apply_all_rules(Tab1,Tab2)).

apply_det_rules(T,Tab,Tab1):-
  C=..[H,Tab,L],
  member(Tab1,L),
  call(C),!

The following example illustrates inference under DISPONTE semantics.

Example 3 Consider the following KB, a probabilistic version of that proposed in Example 1.

\[ 0.5 \vdash \exists x. \text{hasAnimal}(x) \sqsubseteq \text{NatureLover}(x) \]
\[ \text{fluffy} : \text{Cat} \]
\[ \text{tom} : \text{Cat} \]

It indicates that the individuals that own an animal which is a pet are nature lovers with a 50% probability and cats are pets with a 60% probability. The KB has four possible worlds:

\{\{1\},\{2\}\}, \{\{1\}\}, \{\{2\}\}, \{\}\}

and the query axiom \( Q = \text{kevin} : \text{NatureLover} \) is true in the first of them, while in the remaining ones it is false. The probability of the query is \( P(Q) = 0.5 \cdot 0.6 = 0.3 \).

When a probabilistic KB is given as input, all the axioms are translated by means of the Thea2 library. Then, for each probabilistic axiom of the form \( \\text{Prob} \vdash \text{Axiom} \), a fact \( p(\text{Axiom},\text{Prob}) \) is asserted in the Prolog KB.

To compute the probability of queries to KBs following the DISPONTE semantics, we can first perform MIN-ENUM. Then the explanations must be made mutually exclusive, so that the probabilities of individual explanations are computed and summed. This can be done by exploiting a splitting algorithm as shown in (Poole 2000). Alternatively, we can assign independent Boolean random variables to the axioms contained in the explanations and define the DNF Boolean formula \( f_K \) which models the set of explanations \( K \). Thus \( f_K(X) = \bigvee_{\kappa \in K} \bigwedge_{(E_i,k) \in \kappa} X_i \bigwedge_{(E_i,0) \in \kappa} \neg X_i \) where \( X = \{X_i|(E_i,k) \in \kappa \} \) is the set of Boolean random variables.

TRILLP, instead, computes directly a pinpointing formula which is a monotone Boolean formula that represents the set of all MinAs.

Irrespective of which representation of the explanations we choose, a DNF or a general pinpointing formula, we can
Figure 3: BDD representing the function \( f(X) = (X_1 \land X_3) \lor (X_2 \land X_3) \).

apply knowledge compilation and transform it into a Binary Decision Diagram (BDD), from which we can compute the probability of the query with a dynamic programming algorithm that is linear in the size of the BDD.

A BDD for a function of Boolean variables is a rooted graph that has one level for each Boolean variable. A node \( n \) in a BDD has two children: one corresponding to the 1 value of the variable associated with the level of \( n \), indicated with \( \text{child}_1(n) \), and one corresponding to the 0 value of the variable, indicated with \( \text{child}_0(n) \). When drawing BDDs, the 0-branch - the one going to \( \text{child}_0(n) \) - is distinguished from the 1-branch by drawing it with a dashed line. The leaves store either 0 or 1. Figure 3 shows a BDD for the function \( f(X) = (X_1 \land X_3) \lor (X_2 \land X_3) \), where the variables \( X = \{X_1, X_2, X_3\} \) are independent Boolean random variables.

A BDD performs a Shannon expansion of the Boolean formula \( f(X) \), so that, if \( X \) is the variable associated with the root level of a BDD, the formula \( f(X) \) can be represented as \( f(X) = X \land f^X(X) \lor \bar{X} \land f^\bar{X}(X) \) where \( f^X(X) \) \( (f^\bar{X}(X)) \) is the formula obtained by \( f(X) \) by setting \( X \) to 1 (0). Now the two disjuncts are pairwise exclusive and the probability of \( f(X) \) being true can be computed as \( P(f(X)) = P(X)P(f^X(X)) + (1 - P(X))P(f^\bar{X}(X)) \) by knowing the probabilities of the Boolean variables of being true.

**TRILL-on-SWISH**

In order to popularize DISPONTE, we developed a Web application called “TRILL-on-SWISH” and available at [http://trill.lamping.unife.it](http://trill.lamping.unife.it). We exploited SWISH (Lager and Wieliemaker 2014), a recently proposed Web framework for logic programming that is based on various features and packages of SWI-Prolog. SWISH allows the user to write Prolog programs and ask queries in the browser without installing SWI-Prolog on his machine. We modified it in order to manage OWL KBs. SWISH also allows users to collaborate on code development. TRILL-on-SWISH allows users to write a KB in the RDF/XML format directly in the web page or load it from a URL, and specify queries that are answered by TRILL running on the server. Once the computation ends, the results are sent to the client browser and visualized in the Web page.

**Experiments**

In order to evaluate TRILL and TRILL\(^P\) performances, we compared them with BUNDLE, a reasoner for DISPONTE based on Pellet. We used four different knowledge bases of various complexity to which we added 50 probabilistic axioms:

- BRCA\(^1\), which models the risk factor of breast cancer;
- an extract of the DBPedia\(^2\) ontology obtained from Wikipedia;
- Biopax level 3\(^3\), which models metabolic pathways;
- Vicodi\(^4\), which contains information on European history.

For the tests, we used a version of the DBPedia and Biopax KBs without the ABox and a version of BRCA and of Vicodi with an ABox containing 1 individual and 19 individuals respectively. We added 50 probabilistic axioms to each KB. In this experimentation, the probabilistic parameter values were learned using EDGE (Riguzzi et al. 2013b), a system that computes the probability value associated with axioms starting from a set of positive and negative examples in the form of class assertion axioms that we regard as true (false), and for which we would like to get an high (low) probability respectively.

For each dataset, we randomly created 100 different queries. In particular, for the DBPedia and Biopax datasets, we created 100 subclass-of queries, while for the other KBs we created 80 subclass-of and 20 instance-of queries. For generating the subclass-of queries, we randomly selected two classes that are connected in the hierarchy of classes, so that each query had at least one explanation. For the instance-of queries, we randomly selected an individual \( a \) and a class to which \( a \) belongs by following the hierarchy of the classes, starting from the classes to which \( a \) explicitly belongs in the KB.

Table 1 shows, for each ontology, the average number of different MinAs computed and the average time in seconds that TRILL, TRILL\(^P\) and BUNDLE took for computing the probability of the queries. In particular, the BRCA and the version of DBPedia that we used contain a large number of subclass axioms between complex concepts. These preliminary tests show that both TRILL and TRILL\(^P\) performances can sometimes be better than BUNDLE, even if they lack all the optimizations that BUNDLE inherits from Pellet. This represents evidence that a Prolog implementation of a Semantic Web tableau reasoner is feasible and that may lead to practical systems. Moreover, TRILL\(^P\) provides an improvement of the execution time with respect to TRILL when more MinAs are present.

**Related Work**

Usually, DL reasoners implement a tableau algorithm using a procedural language. Since some tableau expansion rules

\(^1\)http://www2.cs.man.ac.uk/~klinovp/pronto/brc/cancer_cc.owl
\(^2\)http://dbpedia.org/
\(^3\)http://www.biopax.org/
\(^4\)http://www.vicodi.org/
are non-deterministic, the developers have to implement a search strategy from scratch. Moreover, in order to solve MIN-A-ENUM, all different ways of entailing an axiom must be found. For example, Pellet (Sirin et al. 2007) is a tableau reasoner for OWL written in Java and able to solve MIN-A-ENUM. It computes ALL-MINAS(Q, K) by finding a single MinA using the tableau algorithm and then applying the hitting set algorithm to find all the other MinAs. Recently, BUNDLE (Riguzzi et al. 2013a) was proposed for reasoning over DISPONTE KBs. BUNDLE exploits Pellet for solving MIN-A-ENUM and computes the probability of queries.

Reasoners written in Prolog can exploit its backtracking facilities for performing the search. This has been observed in various works. Beckert and Posegga (1995) proposed a tableau reasoner in Prolog for FOL based on free-variable semantic tableaux. However, the reasoner is not tailored to DLs. Meissner (2004) presented the implementation of a Prolog reasoner for the DL ACC. This work was the basis of (Herchenröder 2006), that considered ACC and improved (Meissner 2004) by implementing heuristic search techniques to reduce the running time. Faizi (2011) added to (Herchenröder 2006) the possibility of returning explanations for queries but still handled only ACC.

Husted, Motik, and Sattler (2008) presented the KAON2 algorithm that exploits basic superposition, a refutational theorem proving method for FOL with equality, and a new inference rule, called decomposition, to reduce a SHIQ KB to a disjunctive datalog program.

DLog (Lukács and Szeredi 2009) is an ABox reasoning algorithm for the SHIQ language that permits storing the content of the ABox externally in a database and answers instance check and instance retrieval queries by transforming the KB into a Prolog program. TRILL differs from these works for the considered DL and from DLog for the capability of answering general queries.

A different approach is shown in (Ricca et al. 2009), who introduced a system for reasoning on a logic-based ontology representation language, called OntoDLP, which is an extension of (disjunctive) ASP and can interoperate with OWL. This system, called OntoDLV, rewrites the OWL KB into the OntoDLP language, can retrieve information directly from external OWL ontologies and answers queries by using ASP. OntoDLV cannot find the set of explanations thus it is not applicable to DISPONTE KBs. All the presented systems are not able to compute the probability of queries.

Bruynooghe et al. (2010) presented FOPrOblLog, an extension of ProbLog where a program contains a set of probabilistic facts, i.e. facts annotated with probabilities, and a set of general clauses which can have positive and negative probabilistic facts in their body. Each fact is assumed to be probabilistically independent. FOPrOblLog follows the distribution semantics and exploits BDDs to compute the probability of queries. FOPrOblLog is a reasoner for FOL that is not tailored to DLs, so the algorithm could be suboptimal for them.

Cali et al. (2009) combine DLs and logic programs in order to integrate ontologies and rules. They use a tightly coupled approach to (probabilistic) disjunctive description logic programs. They define a description logic program as a pair \((L, P)\), where \(L\) is a DL KB and \(P\) is a disjunctive logic program which contains rules on concepts and roles of \(L\). \(P\) may contain probabilistic alternatives in the style of ICL (Poole 1997). Interpretations assign a probability to ground atoms, in the style of Nilsson probabilistic logic (Nilsson 1986). Queries can be answered by finding all answer sets. Differently from (Cali et al. 2009), in DISPONTE interpretations are not probabilistic and they are assigned a probability, instead of being a mapping from atoms to probabilities.

In (Gavanelli et al. 2015a) and (Gavanelli et al. 2015b), we addressed representation and reasoning for Datalog\(\kappa\) ontologies in an Abductive Logic Programming framework, with existential and universal variables, and Constraint Logic Programming constraints in rule heads. The underlying abductive proof procedure can be directly exploited as an ontological reasoner for query answering and consistency check.

### Conclusions

In this paper we have presented the algorithm TRILL for reasoning on SHOIN\(\kappa\) KBs and the algorithm TRILL\(P\) for reasoning on ALC KBs. The experiments performed show that Prolog is a viable language for implementing DL reasoning algorithms and that their performances are comparable with those of a state-of-art reasoner such as BUNDLE.

In the future we plan to apply various optimizations to our systems in order to better manage the expansion of the tableau. In particular, we plan to carefully choose the rule and node application order. We are also studying an extension of our systems for managing KBs integrating rules and DL axioms. Moreover, we plan to exploit TRILL for implementing algorithms for learning the parameters of probabilistic DISPONTE KBs, along the lines of (Bellodi and Riguzzi 2012; 2013; Riguzzi et al. 2014).

---

Table 1: Average number of MinAs and average time (in seconds) for computing the probability of queries with the reasoners TRILL, TRILL\(P\) and BUNDLE.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>AVG N. MINAS</th>
<th>TRILL TIME (s)</th>
<th>TRILL(^P) TIME (s)</th>
<th>BUNDLE TIME (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRCA</td>
<td>6.49</td>
<td>27.87</td>
<td>4.74</td>
<td>6.96</td>
</tr>
<tr>
<td>DBPedia</td>
<td>16.32</td>
<td>51.56</td>
<td>4.67</td>
<td>3.79</td>
</tr>
<tr>
<td>Biopax level 3</td>
<td>3.92</td>
<td>0.12</td>
<td>0.12</td>
<td>1.85</td>
</tr>
<tr>
<td>Vicodi</td>
<td>1.02</td>
<td>0.19</td>
<td>0.19</td>
<td>1.12</td>
</tr>
</tbody>
</table>
References


Patel-Schneider, P. F.; Horrocks, I.; and Bechhofer, S. 2003. Tutorial on OWL.


Schlobach, S., and Cornet, R. 2003. Non-standard reasoning services for the debugging of description logic termi-


Abstract
Answer set programming (ASP) is an appropriate formalism to represent various problems issued from artificial intelligence and arising when available information is incomplete. When dealing with information expressed in terms of ontologies in some tractable description logic language, ASP must be extended to handle existential variables. We present the syntax and semantics of an ASP language with existential variables using Skolemization. We formalize its links with standard ASP. This work has led to an implementation.

Introduction
This paper deals with the treatment of ontologies in Answer Set Programming (ASP) (Gelfond and Lifschitz 1988). We are interested in using ASP technologies for querying large scale multisource heterogeneous web information. ASP is considered to handle, by using default negation, inconsistencies emerging by the fusion of the sources expressed by scalable description logics. Moreover, ASP can enrich the language of ontologies by allowing the expression of default information (for instance, when expressing inclusion of concepts with exceptions). When dealing with ontologies in ASP, the problem stems from the presence of existential variables in description logics which are not expressible in normal logic programs. The present work proposes a definition of ASP with existential variables in order to express, in a unique formalism, ontologies enriched by default negation and rules. Processing existential variables is done in terms of Skolemization.

The study of the combination of ontologies and rules is not new (Rosati 2006; Eiter et al. 2008; de Bruijn et al. 2010; Motik and Rosati 2010; Ferraris, Lee, and Lifschitz 2011; Lee and Palla 2011; Magka, Krötzsch, and Horrocks 2013). In most of these models, the knowledge base is viewed as a hybrid knowledge base composed of two parts \((T,P)\): \(T\) is a knowledge base describing the ontological information expressed with a fragment of first-order logic, using for example description logic, and \(P\) describes the rules in terms of a logic program.

The miscellaneous attempts to integrate the two formalisms can be distributed into three classes (Eiter et al. 2008; Lee and Palla 2011).

In the first class (like in (Eiter et al. 2008)), the two formalisms are handled separately. \(T\) is seen as an external source of information which can be used by the logic program through special predicates querying the DL base. The two bases are then independent with their own semantics and the link between the two bases is performed using these special predicates. (Eiter et al. 2013) uses their extension of ASP with external atoms to simulate rules with existential variables in the head (external atoms in the body serve to introduce new null values).

The second case (like in (Rosati 2006; Motik and Rosati 2010)) corresponds to an hybrid formalism which integrates DLs and rules in a consistent semantic framework. Predicates of \(T\) can be used in the rules of the program. Nevertheless, there are some restrictions: for instance, these predicates can not be used in the negative part of the body of a rule.

The last case integrates DLs and rules in a unique formalism. For instance, (de Bruijn et al. 2010) uses quantified equilibrium logic (QEL). In this work, several hybrid knowledge bases are defined (with safe restriction, safe restriction without unique name assumption or with guarded restriction) and it is proved that each category and their models can be expressed in terms of QEL.

A large part of these works concerns the questions of complexity and decidability. In these frameworks, existential variables are allowed in the part of the ontological information but are not allowed in the head of the rules.

Next to these models, (Ferraris, Lee, and Lifschitz 2011) proposes a model allowing to cover both stable models semantics and first-order logic by means of a second-order formula issued from the initial information. Its links with the previously cited works have been established in (Lee and Palla 2011). (You, Zhang, and Zhang 2013) proposes an extension of ASP with existential variables in rule heads whose semantics corresponds to that of (Ferraris, Lee, and Lifschitz 2011).

Other works in logic programming take their origin in Datalog and extend the language for specifying ontologies. Datalog+/- is a family of such extensions with syntactical restrictions so that decidability is ensured. Several approaches with existential quantified variables based on Datalog+- have been proposed in the literature but some have no non monotonic negation (Alviano et al. 2012) and other have
only stratified negation (Cali et al. 2010). Nevertheless, one important and interesting point of these works is that they focus on queries which is an important issue when dealing with ontologies.

In (Magka, Krötzsch, and Horrocks 2013), the knowledge base is a single one allowing existential variables and default negation in the same rule. This work studies some conditions of acyclicity and stratification that must be verified by the base ensuring the existence of a unique finite stable model. The work is both theoretical and practical but it is concerned with a limited extension of ASP.

As far as we know, the only works leading to an implementation are those of(Ianni et al. 2005; Eiter et al. 2005), based on (Eiter et al. 2008), and of (Magka, Krötzsch, and Horrocks 2013) which has been applied to information about biochemistry.

The aim of our present work is to describe knowledge in a single framework which can lead to useful implementation. We focus on ASP because it is a powerful framework for knowledge representation and provides efficient solvers. The work consists in enriching the ASP framework to take into account existential variables. It can be seen as the other side of the work consisting in introducing nonmonotonicity in existential rules (Baget et al. 2014b; 2014a).

Next section gives the preliminary notions and definitions useful for the paper. Then, we define programs expressed in 3-ASP, an extension of ASP allowing existential variables, and answer sets on this kind of programs. Last, we give the links between 3-ASP and standard ASP with a method to translate a program expressed in 3-ASP into a program expressed in (standard) ASP and proofs about the transformation.

Preliminaries
In this section, we give the formal definitions of the language and some notions useful in the following of the paper.

The set £ denotes the infinite countable set of variables. A language L is defined as a triplet (CS, FS, PS) which denotes respectively the set of constant symbols, the set of function symbols and the set of predicate symbols of the language. It is assumed that the sets £, CS, FS and PS of any language are disjoint. Function ar denotes the arity function from FS to N and from PS to N which associates to each function or predicate symbol its arity.

The set T (L) denotes the set of terms of a language L = (CS, FS, PS) defined by induction as follows:

- if v £ then v £ T (L),
- if c £ CS then c £ T (L),
- if f £ FS with ar (f) = n > 0 and t1, . . . , tn £ T (L) then f (t1, . . . , tn) £ T (L).

The set GT (L) denotes the set of ground terms of a language L = (CS, FS, PS) defined by induction as follows:

- if c £ CS then c £ GT (L),
- if f £ FS with ar (f) = n > 0 and t1, . . . , tn £ GT (L) then f (t1, . . . , tn) £ GT (L).

The set A (L) denotes the set of atoms of a language L = (CS, FS, PS) defined as follows:

- if a £ PS with ar (a) = 0 then a £ A (L),
- if p £ PS with ar (p) = n > 0 and t1, . . . , tn £ T (L) then p (t1, . . . , tn) £ A (L).

The set GA (L) denotes the set of ground atoms of a language L = (CS, FS, PS) defined as follows:

- if a £ PS with ar (a) = 0 then a £ GA (L),
- if p £ PS with ar (p) = n > 0 and t1, . . . , tn £ GT (L) then p (t1, . . . , tn) £ GA (L).

A substitution over a language L is a mapping from the set of variables to the set of the terms T (L). Let t be a term (resp. a an atom) and σ a substitution, σ (t) (resp. σ (a)) is an instance of t (resp. a).

A ground substitution over a language L is a mapping from the set of variables to the set of the ground terms GT (L). Let t be a term (resp. a an atom) and σ a ground substitution, σ (t) (resp. σ (a)) is a partial ground instance of t (resp. a) w.r.t. the set of variables V.

Syntax and semantics of 3-ASP
In this section, we define a variant of ASP allowing the use of existentially quantified variables (called existential variables in the sequel). The rules proposed here extend classical safe rules (without disjunction) of the form:

H1, . . . , Hn ← B1, . . . , Bm, not N11, . . . , not Nn

where H, B1, . . . , Bm, N1, . . . , Nn are atoms. Safety imposes that all variables that appear in a rule also appear in the positive part of its body. In such a rule, all variables are interpreted as universally quantified. In the sequel, universally quantified variables will be called universal variables. These classical rules are extended in two ways. First, the head of the rule, atom H, is replaced by a conjunction of atoms and each negated atom Ni is also replaced by a conjunction of atoms. These conjunctions allow multiple atoms to refer to the same existential variable. Second, the safety condition is relaxed by allowing these new conjunctions of atoms to contain variables that do not appear in the positive part of the rule. These variables are interpreted as existential ones.

For example, in the rule (p(X,Y) ← q(X), not r(X,Z)), variable X is interpreted as universal, and Y and Z are interpreted as existential. The rule can be read as: “for all X, if q(X) is true and there does not exist Z such that r(X,Z) is true, then one can conclude that there exists Y such that p(X,Y) is true”.

**Definition 1 (3-rule and 3-program)** An 3-program P of language L = (CS, FS, PS) is a set of 3-rules r defined as follows (m, s ≥ 0, n, u1, . . . , us ≥ 1):

H1, . . . , Hn ← B1, . . . , Bm, not (N11, . . . , Nn1s), . . . , not (N1s, . . . , Nn).
with $H_1,\ldots,H_n, B_1,\ldots, B_m, N_1,\ldots, N^1_s,\ldots, N^s_u \in \mathcal{A}(L)$.

We use the following notations:

- \( \text{head}(r) = \{ H_1,\ldots, H_n \} \)
- \( \text{body}^-(r) = \{ B_1,\ldots, B_m \} \)
- \( \text{body}^+(r) = \{ N^1_1,\ldots, N^1_u,\ldots, N^s_1,\ldots, N^s_u \} \)
- \( \forall(r) \) the variables,
- \( \forall_{H}(r) \) the variables which are in \( H_1,\ldots, H_n \) but which are not in \( B_1,\ldots, B_m \) (i.e. existential variables of the head of \( r \)),
- \( \forall_{N}(r) = \{ N^1_1,\ldots, N^1_u,\ldots, N^s_1,\ldots, N^s_u \} \)
- \( \text{body}^-(r) \) must be disjoint (i.e. universal variables of the head of \( r \)), the frontier variables.
- \( \forall_{N}(r) \) the variables which are at least in \( N^1_1,\ldots, N^1_u,\ldots, N^s_1,\ldots, N^s_u \) and in \( B_1,\ldots, B_m \) (i.e. universal variables of \( N^1_1,\ldots, N^1_u,\ldots, N^s_1,\ldots, N^s_u \))

Moreover, the sets \( \forall_{N}(r) \) for every \( 1 \leq i \leq s \) must be disjoint and the sets \( \forall_{H}(r) \) and \( \forall_{N}(r) \) must also be disjoint. (If a variable appears in several of \( N^1_1,\ldots, N^1_u,\ldots, N^s_1,\ldots, N^s_u \), then it must appear in \( B_1,\ldots, B_m \) and it is a universal variable.)

For all rules \( r \) of a program \( P \), \( \forall_{N}(r) \) must be disjoint (i.e. all the names of the existential variables of the program are different).

A rule \( r \) is a definite rule if \( \text{body}^-(r) = \emptyset \) and \( \text{program} \) is a definite program if all the rules are definite.

Let us note that in such a rule \( r \), several atoms are allowed in \( \text{head}(r) \) and in each set of \( \text{body}^-(r) \). In this case, a list of atoms must be seen as the conjunction of each atom of the list.

Concerning the variables involved in the rule, they can be quantified universally or existentially. The quantifiers are not explicitly expressed in the rule but they depend on the position in the rule: the variables appearing in \( \text{body}^-(r) \) are universally quantified while the ones not appearing in \( \text{body}^+(r) \) are existentially quantified. Let us note that the existential variables, in the head or in each negative part of the body, are locally quantified.

**Example 1** Let \( P_2 \) be an \( \exists \)-program of language \( L^r = \{(\{a\},\emptyset,\{p,\text{phdS},d,l,gC\})\} \) with \( \alpha(p) = \alpha(d) = \alpha(l) = 1 \) and \( \alpha(\text{phdS}) = \alpha(gC) = 2 \). \( p \) stands for person, \( \text{phdS} \) for \( \text{phdStudent} \), \( d \) for director, \( l \) for lecturer and \( gC \) for givesCourses.

\[
P_2 = \{ \\
\begin{align*}
r_0 : & p(a), \\
r_1 : & l(a), \\
r_2 : & \text{phdS}(X,D), d(D) \leftarrow p(X), \text{not}(l(X), gC(X,Y)) \\
\end{align*}
\]

The rule \( r_2 \) means that for a person \( X \) there exists a director \( D \) and a phD student of \( D \), unless \( X \) is a lecturer and it exists a course given by \( X \).

We have \( \forall_{H}(r) = \{ X \} \), \( \forall_{H}(r) = \{ D \} \), \( \forall_{N}(r) = \{ l(X), gC(X,Y) \} \) and \( \forall_{N}(r) = \{ X, D \} \).

For each program \( P \), we consider that its language \( L_P = (CS,FS,PS) \) consists of exactly the constant symbols, function symbols and predicate symbols appearing in \( P \).

**Proposition 1** Any (first-order classical) ASP program is an \( \exists \)-program.

**Proof 1** This is a direct consequence of Definition 1.

The semantics of \( \exists \)-programs uses Skolemization of existental variables appearing in the heads of the rules. We now define this Skolemization.

**Definition 2 (Skolem symbols)** Let \( r \) be an \( \exists \)-rule, \( n \) the cardinality of \( \forall_{H}(r) \) and \( Y \in \forall_{H}(r) \) an existental variable of \( r \) then \( sk^1_Y \) is a Skolem function symbol of arity \( n \) if \( n = 0 \) then \( sk \) is a Skolem constant symbol.

**Example 2 (Example 1 continued)** Symbol \( sk^1_D \) is a Skolem function symbol of arity 1 for the existential variable \( D \) of the head of the rule \( r_2 \).

**Definition 3 (Skolem Program)** Let \( P \) be an \( \exists \)-program of language \( L_P \).

Let \( s \) be an ordered sequence of the variables \( \forall_{H}(r) \) of an \( \exists \)-rule \( r \) of \( P \). \( sk(r) \) denotes a Skolem rule obtained from \( r \) as follows: every existental variable \( v \in \forall_{H}(r) \) is substituted by the term \( sk^v \) the Skolem function (constant) symbol associated to \( v \) and \( n = \alpha(sk^v) \) the size of \( s \) (zero if \( \forall_{H}(r) = \emptyset \)). The Skolem program \( \text{sk}(P) \) of an \( \exists \)-program \( P \) is defined by \( \text{sk}(P) = \{ sk(r) \mid r \in P \} \).

**Example 3 (Example 1 continued)** The Skolem rule of \( r_2 \) is the rule:

\[
\text{sk}(r_2) = (\text{phdS}(X, sk_D^1(X)), d(sk_D^1(X)) \leftarrow p(X), \text{not}(l(X), gC(X,Y)))).
\]

Hence \( \text{sk}(P_2) = \{ r_0, r_1, \text{sk}(r_2) \} \) and \( L_{\text{sk}(P_2)} = \{(\{a\}, \{sk^1_D\}, \{p, \text{phdS}, d, l, gC\})\} \).

Skolem rules are still not safe: existential variables remain in the negative bodies. The grounding of such a rule is a partial grounding restricted to the universal variables of the rule, the existental ones remaining not ground. Indeed, a non-ground rule \((p(X) \leftarrow q(X), \text{not}(r(X,Z)))\) could be fired for some constant \( a \) if \( q(a) \) is true and, for all \( z \), \( r(a,z) \) is not true. Suppose two constants \( a \) and \( b \). Then \((p(a) \leftarrow q(a), \text{not}(r(a,a)))\) and \((p(a) \leftarrow q(a), \text{not}(r(a,b)))\) are not equivalent to the non-ground rule for \( X = a \) because the first instance could be fired if \( r(a,b) \) is true (but not \( r(a,a) \)) and the second could be fired if \( r(a,a) \) is true (but not \( r(a,b) \)). Yet neither \( r(a,b) \) nor \( r(a,a) \) should be true for the initial rule to be fired. We thus define a partial grounding, only concerning universal variables. For instance, a partial ground instance of the above non-ground rule would be:

\[
(p(a) \leftarrow q(a), \text{not}(r(a,Z))).
\]

**Definition 4 (Partial Ground Program)** Set \( PG(r) \) for a rule \( r \) of an \( \exists \)-program \( P \) of language \( L_P \) denotes the set
of all partial ground instances of \( r \) over the language \( \mathcal{L}_P \) for \( \forall \mathcal{N}(r) \). The partial ground program \( \text{PG}(P) \) of an \( \exists \)-program \( P \) is defined by \( \text{PG}(P) = \bigcup_{P \in \mathcal{P}} \text{PG}(r) \).

**Example 4 (Example 1 continued)** The language of the Skolem program \( sk(P_U) \) contains only one constant, \( a \), and only one function symbol, \( sk_1 \). The set of ground terms is infinite and the partial grounding leads then to the following infinite program:

\[
\text{PG}(sk(P_U)) = \{
\begin{align*}
p(a), \quad l(a), \\
p(a), \text{not } (l(a), gC(a, Y)). \\
pds(sk_1^1(a), sk_1^1(sk_1^1(a))), (sk_1^1(sk_1^1(a))) \leftarrow \text{p}(sk_1^1(a)) \text{, not } (l(sk_1^1(a), gC(sk_1^1(a), Y)). \\
dots
\end{align*}
\]

**Proposition 2** The partial ground program of an \( \exists \)-program with no multiple head, no multiple default negation and no existential variable is a ground (classical) ASP program.

**Proof 2** This is a direct consequence of Definitions 1 and 4.

**Definition 5 (Reduct)** Let \( P \) be an \( \exists \)-program of language \( \mathcal{L}_P \) and \( X \subseteq \text{GA}(\mathcal{L}_{sk(P)}) \). The reduct of the partial ground program \( \text{PG}(sk(P)) \) w.r.t. \( X \) is the definite partial ground program

\[
\text{PG}(sk(P))|_X = \{
\begin{align*}
\text{head}(r) & \leftarrow \text{body}^+(r), r \in \text{PG}(sk(P)), \\
\text{for all } N & \in \text{body}^-(r) \text{ and } \\
\text{for all ground substitution } \sigma & \text{ over } \mathcal{L}_{sk(P)}, \sigma(N) \not\subseteq X
\end{align*}
\]

**Example 5 (Example 1 continued)** Let

\[
X_1 = \{p(a), l(a), pds(a, sk_1^1(a)), (sk_1^1(a))\}.
\]

Then, for the rule

\[
pds(a, sk_1^1(a)), (sk_1^1(a)) \leftarrow \text{p}(a), \text{not } (l(a), gC(a, Y)).
\]

there is no ground instance of \( l(a), gC(a, Y) \) that is included in \( X_1 \) (since \( X_1 \) does not contain any atom with \( gC \)) and the positive part of the rule is kept. Other rules are kept for the same reason. The obtained program is then:

\[
\text{PG}(sk(P_U))|_{X_1} = \{
\begin{align*}
p(a), \\
l(a), \\
pds(a, sk_1^1(a)), (sk_1^1(a)) \leftarrow \text{p}(a), \\
pds(sk_1^1(a), sk_1^1(sk_1^1(a))), (sk_1^1(sk_1^1(a))) \leftarrow \text{p}(sk_1^1(a)), \\
\text{...}
\end{align*}
\]

Now, let \( X_2 = X_1 \cup \{gC(a, m)\} \) and \( P_{U/2} = P_U \cup \{gC(a, m)\} \).

Here, \( l(a), gC(a, m) \) is a ground instance of the negative body of the rule

\[
pds(a, sk_1^1(a)), (sk_1^1(a)) \leftarrow \text{p}(a), \text{not } (l(a), gC(a, Y)).
\]

that is included in \( X_2 \). Thus, the rule is excluded from the reduct. Other rules are kept. The obtained program is then:

\[
\text{PG}(sk(P_U) \cup \{gC(a, m)\})|_{X_2} \subseteq \{gC(a, m), p(a), l(a), pds(sk_1^1(a), sk_1^1(sk_1^1(a))), (sk_1^1(sk_1^1(a))) \leftarrow \text{p}(sk_1^1(a)), \text{...}\}
\]

Note that the reduct of a program that is Skolemized and partially grounded is a definite ground program: it no longer contains variables. The consequence operator can then be defined as usual, the only difference is that rules can have a conjunction of atoms at head.

**Definition 6** (\( T_P \) consequence operator and \( Cn \) its closure)

Let \( P \) be a definite partial ground program of an \( \exists \)-program of language \( \mathcal{L}_P \). The operator \( T_P : 2^{\text{GA}(\mathcal{L}_P)} \rightarrow 2^{\text{GA}(\mathcal{L}_P)} \) is defined by

\[
T_P(X) = \{a|r \in P, a \in \text{head}(r), \text{body}^+(r) \subseteq X\}.
\]

\[
\text{Cn}(P) = \bigcup_{n=0}^{\infty} T^n_P(\emptyset) \text{ is the least fix-point of the consequence operator } T_P.
\]

**Example 6 (Example 1 continued)**

\[
\text{Cn}(\text{PG}(sk(P_U))|_X) = X \text{ but } \text{Cn}(\text{PG}(sk(P_U) \cup \{gC(a, m)\})|_X) = \{p(a), l(a), gC(a, m)\}.
\]

**Definition 7** (\( \exists \)-answer set)

Let \( P \) be an \( \exists \)-program of language \( \mathcal{L}_P \) and \( X \subseteq \text{GA}(\mathcal{L}_{sk(P)}) \). \( X \) is an \( \exists \)-answer set of \( P \) if and only if \( X = \text{Cn}(\text{PG}(sk(P_U))|_X) \).

**Example 7 (Example 1 continued)** \( X \) is an \( \exists \)-answer set of \( P_U \) and \( \{p(a), l(a), gC(a, m)\} \) is an \( \exists \)-answer set of \( P_U \cup \{gC(a, m)\} \).

**Proposition 3** Let \( P \) be a (classical) ASP program of language \( \mathcal{L}_P \) and \( X \subseteq \text{GA}(\mathcal{L}_P) \). \( X \) is an answer set of \( P \) if and only if \( X \) is an \( \exists \)-answer set of \( P \) considered as an \( \exists \)-program.

**Proof 3** Since \( P \) is a classical ASP program, \( sk(P) = P \) and its (classical) ground ASP program corresponds exactly to \( \text{PG}(P) = \text{PG}(sk(P)) \). Hence \( X \in \text{GA}(\mathcal{L}_P) = \text{GA}(\mathcal{L}_{sk(P)}) \) is an answer set of ground \( P \), by Definition 7, if and only if it is an \( \exists \)-answer set of \( P \) considered as an \( \exists \)-program.

*From \( \exists \)-ASP to ASP*

In this section, we give the translation of an \( \exists \)-ASP program into a standard ASP program and we show that the \( \exists \)-answer sets of the initial program correspond to the answer sets of the new program.

The first step of the translation is the normalization whose goal is twofold: to remove the conjunctions of atoms from negative parts of the rules and to remove existential variables from these negative parts. The obtained program is equivalent in terms of answer sets.
Definition 8 Let \( P \) be an \( \exists \)-program of language \( \mathcal{L}_P \). Let \( r \) be an \( \exists \)-rule of \( P \) \((m,s \geq 0, n,u_1,\ldots,u_s \geq 1)\):

\[
H_1,\ldots,H_n \leftarrow B_1,\ldots,B_m, \not \exists (N_1^1,\ldots,N_s^1),\ldots,\not \exists (N_1^u,\ldots,N_s^u) .
\]

with \( H_1,\ldots,H_n, B_1,\ldots,B_m, N_1^1,\ldots,N_s^1,\ldots,N_1^u,\ldots,N_s^u \in \mathcal{A}(\mathcal{L}_P) \). Let \( \mathcal{N} \) be a set of new predicate symbols (i.e. \( \mathcal{N} \cap \mathcal{P} \mathcal{S} = \emptyset \)).

The normalization of such an \( \exists \)-rule is the set of \( \exists \)-rules

\[
\mathcal{N}(r) = \{ H_1,\ldots,H_n \leftarrow B_1,\ldots,B_m, \not \exists (N_1^1,\ldots,N_s^1),\ldots,\not \exists (N_1^u,\ldots,N_s^u) .
\]

with \( N_i \), the new atom \( p(X_1,\ldots,X_v) \), \( p \in \mathcal{N} \) new predicate symbol for every \( N_i \) and \( \forall \exists(r)(N_1^i,\ldots,N_s^i) = \{X_1,\ldots,X_v\} \).

The normalization of \( P \) is defined as \( \mathcal{N}(P) = \bigcup_{r \in P} \mathcal{N}(r) \).

Set \( \mathcal{G} \mathcal{A}(\mathcal{L}_{\mathcal{S}k}(P)) \) is the set of Skolem ground atoms for the new predicate symbols defined as follows:

- if \( a \in \mathcal{N} \) with \( \mathcal{a}(a) = 0 \) then \( a \in \mathcal{G} \mathcal{A}(\mathcal{L}_{\mathcal{S}k}(P)) \).
- if \( p \in \mathcal{N} \) with \( \mathcal{a}(p) > 0 \) and \( t_1,\ldots,t_a \in \mathcal{G} \mathcal{T}(\mathcal{L}_{\mathcal{S}k}(P)) \) then \( p(t_1,\ldots,t_a) \in \mathcal{G} \mathcal{A}(\mathcal{L}_{\mathcal{S}k}(P)) \).

Example 8 (Example 1 continued) Let \( p^N \) be a new predicate symbol. The negative part of the rule \( r \) of \( X = \{X_1,\ldots,X_v\} \) and \( \exists \)-answer set of \( PG(\mathcal{L}_{\mathcal{S}k}(P)) \).

The following proposition shows that normalization preserves answer sets of an \( \exists \)-program: it only adds some atoms formed with the new predicate symbols from \( \mathcal{N} \).

Proposition 4 Let \( P \) be an \( \exists \)-program of language \( \mathcal{L}_P \) and \( X \subseteq \mathcal{G} \mathcal{A}(\mathcal{L}_{\mathcal{S}k}(P)) \). If \( X \) is an \( \exists \)-answer set of \( P \) then there exists \( \exists \)-answer set of \( N(P) \). If \( X \) is an \( \exists \)-answer set of \( N(P) \) then \( \mathcal{G} \mathcal{A}(\mathcal{L}_{\mathcal{S}k}(P)) \) is an \( \exists \)-answer set of \( P \).

The lemma used in the following proof shows that if the normalization is applied on only one rule \( r \) and only one part of the negative body of this rule, then the answer sets of the original program are preserved up to the added atom. If \( r \) has the following form:

\[
H_1,\ldots,H_n \leftarrow B_1,\ldots,B_m, \not \exists (N_1^1,\ldots,N_s^1),\ldots,\not \exists (N_1^u,\ldots,N_s^u) .
\]

then the "partial normalization" of \( r \) for \( N_1^1,\ldots,N_s^1 \) leads to the rules

\[
r^1 = H_1,\ldots,H_n \leftarrow B_1,\ldots,B_m, \\
\not \exists (N_1^1,\ldots,N_s^1),\ldots,\not \exists (N_1^{u-1},\ldots,N_s^{u-1}) , \not N_a .
\]

and \( r^2 = N_a \leftarrow N_1^1,\ldots,N_s^1 \). A program \( P \) with the rule \( r \) and the program \( PG(\mathcal{L}_{\mathcal{S}k}(P)) \) where the rule is replaced by the rules \( r^1 \) and \( r^2 \) have the same answer sets except for \( N_a \).

The proof can be constructed by induction by applying the lemma to each part of the negative body of \( r \) and, then, to each rule of the program.

Proof 4 The proof is by induction on the following lemma:

Let \( P \) be an \( \exists \)-program of language \( \mathcal{L}_P \), \( r = (H \leftarrow C, not \exists (N_1,\ldots,N_u) \in PG(\mathcal{L}_{\mathcal{S}k}(P)) \}\{r\}, r^1 = (H \leftarrow C, not \exists (N_1,\ldots,N_u) \in PG(\mathcal{L}_{\mathcal{S}k}(P)) \}\{r\}, r^2 = PG(\mathcal{L}_{\mathcal{S}k}(P)) \}\{N_a \leftarrow N_1,\ldots,N_u \in PG(\mathcal{L}_{\mathcal{S}k}(P)) \}\}

\( \mathcal{N} \subseteq \mathcal{G} \mathcal{A}(\mathcal{L}_{\mathcal{S}k}(P)) \).

If there exists a substitution \( \theta \) such that \( \{\theta(N_1),\ldots,\theta(N_u)\} \subseteq \mathcal{N} \) if and only if \( \mathcal{C}_n((P^r \cup \{r\}^X) = \mathcal{X} \cup \{N\} \).

If for all substitutions \( \theta \), \( \{\theta(N_1),\ldots,\theta(N_u)\} \subseteq \mathcal{N} \) if and only if \( \mathcal{C}_n((P^r \cup \{r\}^X) = \mathcal{X} \cup \{N\} \).

Proof 4 Let \( P \) be an \( \exists \)-program of language \( \mathcal{L}_P \).

¬p \( \mathcal{P} \)

[43x270]not p

It is replaced by

[42x425]not

The following proposition shows that Skolemization and grounding preserve answer sets of a normalized \( \exists \)-program.

Proposition 5 Let \( P \) be a normalized \( \exists \)-program of language \( \mathcal{L}_P \) and \( X \subseteq \mathcal{G} \mathcal{A}(\mathcal{L}_{\mathcal{S}k}(P)) \). X is an \( \exists \)-answer set of \( P \) if and only if \( X \) is an \( \exists \)-answer set of \( PG(\mathcal{L}_{\mathcal{S}k}(P)) \).
Proof 5 Since for all \( r \in PG(sk(P)) \), \( \forall_{N\exists}(r) = \emptyset \) (since \( r \) is normalized), \( \forall_{N\exists}(r) = V(r) \) and \( V_{N\exists}(r) = \emptyset \) (since \( r \) is Skolemized) then \( PG(sk(P)) = sk(PG(sk(P))) = PG(sk(P))(sk(P)) \).

By Definition 7, \( X \) is an \( \exists \)-answer set of \( P \) iff \( X = Cn(PG(sk(P))^{X}) \) iff \( X = Cn(PG(sk(Pg(P)(sk(P))))^{X}) \) iff \( X \) is an \( \exists \)-answer set of \( PG(sk(P)) \).

Once an \( \exists \)-program is normalized and Skolemized, the only non-standard parts that remain are the conjunctions of atoms in rule heads. The last step of the translation is the expansion where we remove the sets of atoms in each head while keeping the link between the existential variables. It simply consists to duplicate a rule as many time as the rule contains atoms in its head, each new rule having only one of these atoms in its head. Preceding Skolemization allows to preserve the links between the existential variables of the head. The obtained program is equivalent in terms of answer sets.

Definition 9 Let \( P \) be a ground Skolemized normalized program and \( r \in P \) \( m, s \geq 0, n > 0 \):

\[
H_{1}, \ldots, H_{n} \leftarrow B_{1}, \ldots, B_{m}, \text{not } N_{1}, \ldots, \text{not } N_{n},
\]

with \( H_{1}, \ldots, H_{n}, B_{1}, \ldots, B_{m}, N_{1}, \ldots, N_{n} \in GA(L(P)) \).

The expansion of such a rule is the set defined by:

\[
\text{Exp}(r) = \{ H_{1} \leftarrow B_{1}, \ldots, B_{m}, \text{not } N_{1}, \ldots, \text{not } N_{n}, \ldots \}
\]

The expansion of \( P \) is defined as \( \text{Exp}(P) = \bigcup r \in P \text{ Exp}(r) \).

Example 10 (Example 1 continued) The following rule of the program from Example 9: \( phdS(a, sk_{D}^{1}(a)) \leftarrow p(a), \text{not } p^{N}(a) \). is split into the two rules:

\( phdS(a, sk_{D}^{1}(a)) \leftarrow p(a), \text{not } p^{N}(a) \) and \( d(sk_{D}^{1}(a)) \leftarrow p(a), \text{not } p^{N}(a) \).

The same treatment is applied to the other rules with both predicates \( phdS \) and \( d \) in the head.

The following program is obtained:

\[
\text{Exp}(PG(sk(N(P_{d}))))) = \{ p(a), \text{not } p^{N}(a), \text{not } p^{N}(a), \text{not } p^{N}(a), \}
\]

Proposition 6 Let \( P \) be a ground Skolemized normalized \( \exists \)-program of language \( L(P) \) and \( X \subseteq GA(L(P)) \). \( X \) is an \( \exists \)-answer set of \( P \) if and only if \( X \) is an \( \exists \)-answer set of \( \text{Exp}(P) \).

Proof 6 The only difference is on the computation of the fix-point of (classical) \( T_{P} \) operator and new \( T_{P} \) operator defined in Definition 6 and clearly enough fix-point are identical since \( P \) is ground.

Proposition 7 Let \( P \) be an \( \exists \)-program. \( \text{Exp}(PG(sk(N(P)))) \) is an (ground classical) ASP program.

Proof 7 This proposition is a direct consequence of Definitons 3, 4, 8, 9 and Proposition 2.

The last proposition establishes equivalence, up to new atoms introduced by normalization, between \( \exists \)-answer sets of an \( \exists \)-program and classical answer sets of the program after normalization, Skolemization and expansion.

Proposition 8 Let \( P \) be an \( \exists \)-program of language \( L(P) \) and \( X \subseteq GA(L(sk(P))) \). If \( X \) is an \( \exists \)-answer set of \( P \) then there exists \( Y \subseteq \text{GAN}(L(sk(P))) \) such that \( X \cup Y \) is a (classical) answer set of \( \text{Exp}(PG(sk(N(P)))) \).

By propositions 3 and 7, \( X \subseteq GA(L(P)) \). If \( Y \) is an \( \exists \)-answer set of \( \text{Exp}(PG(sk(N(P)))) \) then \( X \cup Y \) is an \( \exists \)-answer set of \( \text{Exp}(PG(sk(N(P)))) \).

Proposition 8 Let \( P \) be an \( \exists \)-program and \( X \subseteq GA(L(sk(P))) \).

- If \( X \) is an \( \exists \)-answer set of \( P \) then, by proposition 4, there exists \( Y \subseteq \text{GAN}(L(sk(P))) \) such that \( X \cup Y \) is an \( \exists \)-answer set of \( P \).

- By proposition 6, \( X \cup Y \) is an \( \exists \)-answer set of \( \text{Exp}(PG(sk(N(P)))) \).

By propositions 3 and 7, \( X \subseteq GA(L(P)) \). If \( Y \) is an \( \exists \)-answer set of \( \text{Exp}(PG(sk(N(P)))) \) then \( X \cup Y \) is an \( \exists \)-answer set of \( \text{Exp}(PG(sk(N(P)))) \).

- If \( X \) is a (classical) answer set of \( \text{Exp}(PG(sk(N(P)))) \) then, by propositions 3 and 7, \( X \) is an \( \exists \)-answer set of \( \text{Exp}(PG(sk(N(P)))) \).

By proposition 6, \( X \) is an \( \exists \)-answer set of \( \text{Exp}(PG(sk(N(P)))) \). By proposition 5, \( X \) is an \( \exists \)-answer set of \( N(P) \). By proposition 4, \( X \subseteq GA(L(sk(P))) \) is an \( \exists \)-answer set of \( P \).

Conclusion

This paper is a first step of formalisation of ASP allowing the use of existential variables. It is well suited to integrate ontologies and rules in a unique formalism.

From a practical point of view, the proposed translation from \( \exists \)-ASP to ASP allows us to use any solver. But let us note that we have implemented this translation as a front-end of the solver ASPeriX which uses on-the-fly grounding (Lefèvre et al. 2015). This should help, in the future, for dealing with variables in a more efficient way.

An in-depth comparison with other formalisms remains to be done. One of the closest work is (Baget et al. 2014b) dealing with existential rules extended with non monotonic negation. In this work, existential variables are only allowed in the rule heads, not in the negative bodies. ASPeriX semantics (defined via a notion of computation inspired from (Liu et al. 2010)) is adapted for defining different chases
(forward chaining algorithms) for non monotonic existential rules. Our present work should be linked to one of these chases, the Skolem-chase.

Another ongoing work is to deal efficiently with queries in this framework. This is not obvious due to the nonmonotonic aspect of ASP and the potential inconsistency of an ASP program. It seems that very little work has been done on these aspects.

Acknowledgements
This work received support from ANR (French National Research Agency), ASPIQ project reference ANR-12-BS02-0003.

References


Imperfect Querying through Womb Grammars plus Ontologies

Veronica Dahl
Universitat Ulm and Simon Fraser University
Germany and Canada
veronica@cs.sfu.ca

Sergio Tessaris
University of Bozen–Bolzano
Italy
stessaris@inf.unibz.it

Thom Fruehwirth
Universitat Ulm
Germany
thom.fruehwirth@uni-ulm.de

Abstract

Womb grammars, or WGs, are a failure-driven constraint-based parsing mechanism specifically developed for cross-language grammar engineering, whose main parsing operation consists of looking for failed constraints between pairs of daughters of a phrasal category. For instance, rather than rejecting those noun phrases where an adjective daughter precedes the noun daughter (a natural mistake for say, an Italian querying in English), a WG checks whether that English ordering requirement fails, and produces a failure indicator if so. Thus, rather than acting solely as filters impeding incorrect sentences from being parsed, the constraints described for a WG can be relaxed to admit mistakes that are personalized to a certain type of user.

Syntactic constraints have been the most studied for WGs, since their first aim was to “repair” a known language’s grammar until it reflected that of another language, by modifying constraints that failed with respect to input in the other language. However any other kind of information can also be consulted.

In this article we extend WG parsing to incorporate semantic information in view of imperfect querying, and we show how the approach lends itself in particular to ontology-driven enhancements. We assume familiarity with Prolog and in particular, CHR.

Introduction

Constraint Satisfaction has yielded powerful results in many AI areas, including human language processing. Systems that handle multi sets of constraints (as CHR (Frühwirth 1998) and CHRG (Christiansen 2005)) have proved especially suitable for efficiently automating bottom-up sentence analysis through interpreting grammar specifications as directly executable.

The constraint-based approach to parsing typically expresses a language’s grammar as a set of linguistic constraints whose satisfaction within a given sentence sanctions it as correct (or not) in the language described by the grammar, and associates it with some desired representation (e.g. syntactic, semantic, pragmatic).

Among the linguistic theories that lend themselves the most to constraint-based implementation are those that split the information previously packed into one rewriting rule into several constraints or properties. These constraint based or property-based theories, such as Property Grammars (PG) (Blache 2005) evolved from IDLP, which unfolds a rewrite rule into the two constraints of immediate dominance (expressing which categories are allowable daughters of a phrasal category) and linear precedence (expressing which of the daughters must precede which others). They deal mostly with syntax, whereas for query answering we obviously need to address meaning representation too.

For example in the PG framework, English noun phrases can be described through a few constraints such as precedence (a determiner must precede a noun, an adjective must precede a noun), uniqueness (there must be at most one determiner), exclusion (an adjective phrase must not coexist with a superlative), obligation (a noun phrase must contain the head noun), and so on. Instead of resulting in either a parse tree or in failure as traditional parsing schemes do, such frameworks characterize a sentence through the list of the constraints a phrase satisfies and the list of constraints it violates, so that even incorrect or incomplete phrases will be parsed. Moreover, it is possible to relax some of the constraints by declaring relaxation conditions in modular fashion. A recent adaptation of this framework into grammar transformation — Womb Grammars (Dahl and Miralles 2012) — automates as well the induction of a language’s syntax from that of another.

In such theories, the modularity obtained by splitting grammatical information apart into constraints leads naturally to more robust parsers, since it allows us to clearly identify from the parser’s output which constraints are satisfied and which fail, which allows us to accept even incomplete or incorrect sentences, instead of simply failing to parse them. We can also produce some indication of the sentence’s degree of acceptability by analyzing the failed properties.

While the property-based family of grammars deals mostly with syntax, one of its properties —dependency— has been designed to carry some semantic information. However, this information is restricted to stating how a category’s features (such as gender and number) affect the features of another category, so it is patently insufficient to build meaning representations.

In this article we extend WGs to incorporate semantic in-
formation in view of imperfect querying, and we show how this approach lends itself in particular to ontology-driven enhancements, by dynamically exploiting the failure-driven parsing methodology of WGs together with ontological information that allows us to perfect the input.

**Motivation**

In the PG formalism per se (Blache 2005), from which WG evolved, no parse tree is considered necessary (although at least one implementation offers them in view of user-friendliness (Dahl and Blache 2004)). Instead, sentences are characterized as a list of satisfied constraints (or properties) and a list of unsatisfied ones.

WG, in contrast, associate a parse tree to every successfully parsed sentence (notice however that parsing success no longer implies correctness).

The parse tree is incrementally built bottom-up, from the partial parse trees of its sub phrases. These remain available even when it’s impossible to parse all the way up to a sentence node. In that case, the partial parse trees are output, together with a list of failed syntactic properties.

The satisfied properties are left implicit as a complement of the unsatisfied ones, and the syntactic tree (or trees) obtained can then serve for modularly building semantics that might in turn, concomitantly with the failed properties and ontological information, aid in perfecting the input to the point where the analysis can further proceed. This observation motivates the present work.

For example, an Italian person querying in English might mean to ask “How many dessert dishes are there in the menu?”, but actually enter “How many dish desserts are there in the menu?”. Since “dish” can legitimately act as an adjective, as in “a dish washer”, in the absence of semantic constraints the sentence entered could be taken as a request to count the number of desserts of type “dish”. Therefore, just relaxing the linear precedence English constraint between noun and adjective in order to cater to Italian users would not help detect the incorrect ordering- it would only result in a wrong parse. However by consulting ontologies, we can detect that “dessert” is a more likely qualifier of “dish” than the other way around, use the failed constraint to correct the number agreement feature in both words, and identify “dessert” as the adjective for the noun “dishes”.

Note that for a grammar whose constraints are fully described, the satisfied syntactic properties of a given sentence’s parse will be a complement of those failed, and will thus be deducible from them whenever needed. Therefore, by checking only for failure we lose no generality, while obtaining considerable gains in efficiency.

**Background**

**Property Grammars**

The PG formalism presently comprises the following seven categories (we adopt the handy notation of (Duchier, Dao, and Parmentier 2013) for readability, and the same example):

**Constituency** $A : S$, children must have categories in the set $S$

**Obligation** $A : \triangle B$, at least one $B$ child

**Uniqueness** $A : B!$, at most one $B$ child

**Precedence** $A : B \prec C$, $B$ children precede $C$ children

**Requirement** $A : B \Rightarrow C$, if $B$ is a child, then also $C$ is a child

**Exclusion** $A : B \not\Leftrightarrow C$, $B$ and $C$ children are mutually exclusive

**Dependency** $A : B \sim C$, the features of $C1$ and $C2$ are the same

**Example 1** For example, if we denote determiners by $D$, nouns by $N$, personal nouns by $PN$, verbs by $V$, noun phrases by $NP$, verb phrases by $VP$ and sentences by $Se$, the context free rules $NP \rightarrow D N$ and $NP \rightarrow N$, which determine what a noun phrase is, can be translated into the following equivalent constraints: $NP : \{D, N\}, NP : D!, NP : \triangle N, NP : N!, NP : D \prec N, D : \{\}, N : \{\}$.

Incorrect sentences can be “accepted” through declaring some constraints as relaxable. For instance, while from the context-free grammar rules shown we wouldn’t be able to parse “the the book” (a common mistake from cutting and pasting in word processors), in the constraint-based formulation we can if we relax the uniqueness of determiner constraint.

Relaxation can be made conditional (e. g. a head noun’s requirement for a determiner can be made relaxable in case the head noun is generic and in plural form, as in “Lions sleep tonight”). The failure of relaxable constraints is signaled in the output, but does not block the entire sentence’s analysis. Implementations not including constraint relaxation capabilities implicitly consider all properties as relaxable.

Incomplete sentences can be parsed to whatever degree is possible, e.g. input such as “The lion shrewdly” might yield a correct analysis of the noun phrase and the adverb but identify no verb phrase- and hence no sentence-, or alternatively, the input might be parsed into an incomplete sentence, if a sentence’s requirement for a verb phrase is relaxed.

**Womb Grammars**

Womb grammars include the set of properties shown above. They can not only parse sentences from a given grammar, as PG can, but can also induce a target language’s grammar from another language’s known grammar. The latter functionality shall not concern us here. Interested readers can refer to (Dahl and Miralles 2012; Dahl, Miralles, and Becerra 2012; Becerra, Dahl, and Miralles 2013; Becerra, Dahl, and Jiménez-López 2014).

Previous parsing mechanisms for property-based grammars – as well as for many other constraint-based research areas – focus on constraint satisfaction. In the remainder of this paper, we shall show how our parsing mechanism for WGs, through focusing on constraint failure instead, allows us a useful and elegant extension into semantic properties as well as a clean while dynamic interaction between syntax and semantics, with particularly fruitful ramifications in interaction with ontologies.
Our implementation is done in terms of CHR grammar, or CHRG (Christiansen 2005). CHRGs are a grammatical interface to CHR, providing it what DCGs provide to Prolog—namely, they invisibly handle input and output strings for the user. In addition, they include constructs to access those strings dynamically, and the possibility of reasoning in non-classical ways, with abduction or with resource-based assumptions.

For the purposes of this paper, we only use two types of CHRG rules, which parallel the CHR rules of propagation and simplification, and are respectively defined as follows:

A propagation grammar rule is of the form
\[ \alpha \rightarrow \beta \leftarrow \gamma \rightleftharpoons G | \delta. \]

The part of the rule preceding the arrow \( \leftleftharpoons \) is called the head, \( G \) the guard, and \( \delta \) the body; \( \alpha, \beta, \gamma \) are sequences of grammar symbols and constraints so that \( \beta \) contains at least one grammar symbol, and \( \delta \) contains exactly one grammar symbol which is a nonterminal (and perhaps constraints); \( \alpha (\gamma) \) is called left (right) context and \( \beta \) the core of the head; \( G \) is a conjunction of built-in constraints as in CHR and no variable in \( G \) can occur in \( \delta \). If left or right context is empty, the corresponding marker is left out and if \( G \) is empty (interpreted as true), the vertical bar is left out. The convention from DCG is adopted that Prolog calls (i.e., non-grammatical stuff) in head and body of a rule are enclosed by curly brackets. Gaps and parallel match are not allowed in rule bodies.

A simplification grammar rule is similar to a propagation grammar rule except that the arrow is replaced by \( \leftleftharpoons \).

Whereas propagation rules add \( \delta \) to the constraint store (where \( \delta' \) denotes \( \delta \) affected by any substitutions needed for the rule’s application), simplification rules replace \( \beta' \) by \( \delta' \), so that \( \beta' \) is removed from the constraint store.

**Failure-Driven Parsing**

Typically, constraint based programming strives to solve constraints, i.e., to satisfy them. In our problem domain however, the aim is to reach an internal representation of a query even if imperfect.

For perfect queries, if we can assume that the satisfied constraints will be the complement of those that fail (a reasonable assumption, which we make), we can get away with checking that no constraint fails.

For imperfect queries, clearly we cannot allow all constraints to fail at once without significant trouble -or even impossibility- in arriving at any useful meaning representation of the query. However we can capitalize on knowing who will query the system, and admit one type of failed constraint accordingly. For the example in our abstract, relaxing precedence between noun and adjectival will cater to romance language speakers.

In rigour, not all constraints are checked only for failure. The constituency constraint is checked for satisfaction. The reason for this is that our parsing is driven by projection (e.g. making a noun phrase out of a noun, a verb phrase out of a verb, etc.) plus category expansion, which will expand a phrase to include any adjacent constituents that are legal for the type of phrase and that satisfy all non-relaxable constraints between them. Since the expansion rule is guided by constituency, senseless expansions do not occur. e.g. NP can be expanded to include an adjacent D (to the left or right!) but not a V. A consequence of this approach is that the constituency constraint is not relaxable.

In order to check for failed constraints efficiently, lexical categories are parsed into a representation that includes their lexical type, word boundaries within the sentence, their list of syntactic attributes (gender, number) and the portion of parse tree that they will contribute to the entire parse. Concretely, our parser expands instantiated categories, which are CHRG grammar symbols of the form

\[ \text{iCat}(\text{Category}, \text{Attributes}, \text{Tree}) \]

These are compiled into CHR constraints with the word boundaries having been made explicit:

\[ \text{iCat}(\text{Start}, \text{End}, \text{Category}, \text{Attributes}, \text{Tree}) \]

Within a CHRG rule, we can spy whenever needed on the dynamic values of the Start and End points, simply by adding them explicitly after the grammar symbol, in the notation \( :(\text{Start},\text{End}) \). For instance:

\[ \text{iCat}(\text{Category}, \text{Attributes}, \text{Tree}) :(\text{Start},\text{End}) \]

**Example 2 (Instantiated categories)** Take the noun phrase “an apple”, implicitly located (as all phrases input to the analyser) as from point 0. Parsing it results in the following instantiated categories (shown in the implicit CHR notation):

\[ \text{iCat}(0, 1, \text{det}, \text{[sing,neutral]}, \text{det(an)}) \]
\[ \text{iCat}(1, 2, \text{n}, \text{[sing,neutral]}, \text{n(apple)}) \]
\[ \text{iCat}(0, 2, \text{np}, \text{[sing,neutral]}, \text{np(iCat(0, 1, \text{det}, \text{[sing,neutral]}, \text{det(an)}), \text{iCat}(1, 2, \text{n}, \text{[sing,neutral]}, \text{n(apple))}))}) \]

Notice that the NP inherits the attributes of the underlying N. This is useful for checking dependency constraints, which require attributes to match. Later we shall see how to incorporate semantics as well.

**Checking for Constraint Violation**

Because properties are defined on pairs of daughters of a given phrasal category, they can be checked in very modular fashion. Their relationships with other parts of the sentence, including with further ancestors than the phrase itself, are verified in the same way, phrase by phrase. So each phrase in a sentence to be analyzed is sanctioned bottom-up from its direct daughters, by checking their properties. This allows us to construct semantic representations in equally modular fashion, as we will see in the next section. Properties other than constituency, as we saw, are checked for failure, and their failure is signalled by adding a constraint which carries the information that they have failed.

We next show, through the examples of uniqueness and obligation, that different constraint violation checking may need to take place at different stages of analysis.
Example 3 (Uniqueness) Violations of uniqueness constraints are checked by a CHRG rule that finds two adjacent words of same category C within the bounds of a phrase of category Cat being parsed, where C has been constrained to appear only once within that phrase. Should such adjacent words be found, the information that uniqueness of a category C under Cat has been violated in the range these categories cover is added as a (fact represented as a CHR) constraint, and the parse tree for the phrase can either delete one occurrence if the repeated category is the same one (as in “the the book”), or include both if different, so that further considerations can be made before choosing one over the other. In slightly simplified form, our CHRG rule that checks for the violation of uniqueness Cat : C! looks as follows:

\[
\text{icat}(C, \text{Attr1}, \text{Tree1}): (N1, N2), \ldots, \text{icat}(C, \text{Attr2}, \text{Tree2}): (N3, N4), \{\text{icat}(N5, N6, \text{Cat}, \_\_ \text{Tree})\}, \\
\{\text{tpl}(\text{uniqueness} (\text{Cat}, C))\}
\]

\% The C's are within the bounds of Cat:
\% N5 \prec N1, N4 \prec N6,
\% And they are its direct daughters:
\% Tree=..[Cat|T],
\% member(icat(N1,N2,C, Attr1,Tree1),T),
\% member(icat(N3,N4,C, Attr2,Tree2),T)
\% failed(uniqueness(Cat,C)).

The first line in the above code finds a category C between word boundaries N1 and N2, with attributes Attr1 and parse tree Tree1. The three dots indicate a skipped substring after N2, before another instance of the same category C is found between the word boundaries N3 and N4. The Prolog calls (between curly brackets) and the guard find a category Cat that dominates both instances of C, and a uniqueness property that is required between a phrase Cat and its immediate daughter C (i.e., a requirement that C appear no more than once as immediate daughter of a phrase of category Cat). Once all this is checked, a grammar symbol (failed/1) is thrown into the constraint store, that states that uniqueness of C within Cat is falsified between word boundaries N1 and N4 (since grammar rules are compiled into CHR rules, what will appear in the constraint store is the equivalent CHR constraint (failed/3), namely failed(N1,N4,uniqueness(Cat,C)).

This rule can fire even if a phrasal category hasn’t been fully expanded, because adding more components to an icat is not going to modify uniqueness having been violated.

Example 4 (Obligation) A constraint such as obligation, in contrast, is designed to only fire after the phrasal category has been fully expanded, since adding one more component into the phrase can result in its becoming satisfied (in the case in which the component added is the one which is obligatory). We now show the CHRG rule that checks for failed obligation:

\[
\text{icat}(\text{Cat}, \text{Attr}, \text{Tree}), \\
\% Found Cat
\]

\[
\{\text{tpl}(\text{obligation}(\text{Cat}, C))\}
\% Cat should have C
\% there isn't a child C
\% failed(obligation(Cat, C)).
\% Obligation is violated.
\]

The obligation rule checks a given category for its direct daughters. Since icats are retracted when they fail, this check only fires after the icat has been fully expanded.

Incorporating Semantics

Since our pieces of parse tree may end up disconnected (e.g. if noise words that cannot be parsed intervene, impeding us from reaching a “sentence” node), it would be useful to have partial semantic representations which can be further completed if possible, and when not possible, can at least contribute partial meanings.

For this reason we have chosen a compositional semantics, in which each constituent is given a representation built from applying an expression to another. We use the well-known lambda calculus for representing and combining meaning.

For instance, we may wish to associate the Montague based meaning representation no(X,bird(X),sings(X)) to the sentence “No bird sings”, given the following word representations (shown in functional notation, and using String’ to denote the semantic representation of String):

\[
\text{no'} = \lambda P1. \lambda P2. \lambda X. (P1(X), @((P2,X)))
\]

\[
\text{bird'} = \lambda X. \text{bird}(X)
\]

\[
\text{sings'} = \lambda X. \text{sings}(X)
\]

The sentence’s meaning can be build compositionally, by applying the meaning of the determiner “no” over the meaning of the noun “bird”, which results in the following lambda-expression for “no bird”:

\[
(\text{no bird}') = \lambda X. \text{no}(X, \text{bird}(X), @((P2,X)))
\]

This lambda-expression is then applied to that of the verb phrase, which yields:

\[
(\text{no bird sings}') = \lambda X. (\text{bird}(X), \text{sings}(X))
\]

Overall System Architecture Our system comprises three main components:

- WG
- ontological
- semantic

These components cooperate with each other as follows: Faced to an input sentence, the WG component operates bottom-up from the words’ syntactic representations, building a parse tree plus a list of failed properties for each phrase it can recognize as such.

For instance, for the sentence “Lions sleep” (more on this example later), it will create the following noun phrase’s parse tree:
as well as the list of failed properties
Failed= [exigency(noun,determiner),0,1]

(stating that the property of exigency between a head noun and its required determiner fails in the noun phrase recognized between points 0 and 1 of the input sentence).

The failed exigency property will trigger a semantic completion rule which calls the ontological component in order to verify whether "lions" is a generic term, and given that it is, will complete the analysis by making the implicit meaning "every" explicit in the parse tree:

noun_phrase(det(every),noun(lions))

Once a phrase has been completed, the semantic component is called. This component combines the meaning representations of the noun phrase’s components:

\[
every' = \lambda P_1 \lambda P_2. every(X, @(P_1, X), @(P_2, X))
\]

\[
lion' = \lambda X. lion(X)
\]

by applying that of "every" over that of "lion", which results in

\[
\lambda P_2. every(X, lion(X), @(P_2, X))
\]

Similarly, the WG component will also arrive at a syntactic tree plus a list of failed properties representation for the verb phrase, namely:

verb_phrase(verb(sleeps))
Failed=()

Once these two phrases (noun phrase and verb phrase) have been parsed, the semantic component is called to apply the representation of the noun phrase over that of the verb phrase. Note that since the verb phrase contains only an intransitive verb, the verb phrase’s semantic representation coincides with that of the verb itself:

\[
\lambda X. sleeps(X)
\]

The application of

\[
\lambda P_2. every(X, lion(X), @(P_2, X))
\]

over

\[
\lambda X. sleeps(X)
\]

yields the desired representation

\[
every(X, lion(X), sleeps(X))
\]

Also the WG parser can call the ontological component by expressing the call in the guard of any of its rules. Likewise, the semantic component can, other than controlling the order of application of semantic representations over one another, also call explicitly for ontological information that might allow it to make better decisions at any point.

Graphically, we can depict this architecture as follows:

The degree of interaction inherent in our architecture serves many purposes, allowing us e.g. to discard an extra determiner if two wrongly made it to the input sentence, to reorder two constituents that appear in non-allowable orderings, to reconstruct implicit meanings, to connect partial subtrees together using ontological consultation in order to arrive at a parse for the complete sentence (this is particularly useful in the case of noisy input intervening within the input sentence).

In short, each phrase that the WG parser outputs will enter a stage of semantic composition, and each phrase thus completed will be combined, if possible, with other thus completed phrases, into a higher level phrase that will in turn check semantic composition rules for combining them, with ontologies being available for consultation at any stage of the syntactic parsing or the semantic analysis process.

Implementation Considerations

In terms of implementation, extending WGs to incorporating semantics as described here requires to now have 6-ary instantiated categories whose last argument is the partial semantics associated with that category:

\[
iCat(Start, End, Category, Attributes, Tree, LambdaExpression)
\]

Since we are using Prolog, we must transform the above functional representations into relational ones.

Since CHRG does not support true lambda expressions, we represent \(\lambda X. P\) by the first-order term \(X \backslash P\).

The calls to our (relational equivalent of) "@" use the following Prolog implementation of beta-reduction:

\[
\text{at}(X \backslash P, X, P).
\]

To add the semantic component on each category we can propagate our 5-ary categories into the new 6-ary ones, e.g.:

\[
iCat(det, \{sing, neutral\}, det(every))
\]
This can in fact be automated by a more general rule which consults modular definitions of semantics for each word, so it becomes easy to experiment with different compositional representations.

Next we need to combine the meanings at appropriate points in our parsing process. We postulate that the appropriate point is every time a phrase is completed (i.e., cannot be further expanded). For our example “no bird sings”, once “no bird” has been analyzed into a noun phrase, this noun phrase cannot be further expanded, since “sings” is not allowable as a noun phrase’s direct daughter. At this point the parser looks at the parse tree, at any failed properties associated with it, and consults any ontological information needed to take into account the failed properties, in order to construct the meaning representation of the noun phrase.

A distinction between lambda-calculus variables and query representation variables needs to be made, since a query’s representation will in general include Prolog rather than lambda variables – i.e., variables that are to remain in the final result, being necessary for evaluating the answer to a query. E.g., the variable X in the formula 
\[ \text{np}(X, \text{bird}(X), \text{sings}(X)) \]

is not a variable under lambda-abstraction but rather, a part of the sentence’s meaning representation, which acts as a placeholder for the answers.

Of course, we could have chosen any other kind of meaning representation while using semantic compositionality together with our techniques for managing failed properties in combination with ontological information. The exemplification in terms of lambda-calculus presented above is merely a proof of concept.

Dynamic Interactions with Failed Properties

Types of Failed Properties

Failed properties are placed in the constraint store during the parse of a sentence. There are two kinds of failed properties:

Strongly failed: those that caused an until then potential analysis to fail, and block the said analysis. An example would be the attempt to make “adam eats” a verb phrase; this attempt is blocked because the constraint that a verb must precede its np in a verb phrase does not hold, so the hypothesis that “adam eats” could be a verb phrase is discarded.

Interestingly Failed: those that hold of some constituent that nevertheless does become a part of the result. An example would be the failed but relaxable requirement for a determiner in the subject noun phrase of “Medicines are toxic”. Ontological interactions with a specific semantic domain’s ontologies can help determine which is best. For instance, while for “Lions are quadrupeds” the correct determiner is indeed “every”, for “Medicines are toxic”, “most” might be more appropriate.

Meaning Extraction through Constraint Relaxation and Failed Constraints

Property relaxation can work together with failed constraints for various purposes. For instance, a noun’s syntactic requirement for a determiner must be relaxed on the syntactico-semantic condition that the noun is in plural form and represents a generic concept. This will be accepted and produce an analysis of the noun into a noun phrase. In particular, the constraint equivalent to the grammar symbol:

\[ \text{failed}(\text{obligation}(\text{np}, n, \text{det})). \]

amely:

\[ \text{failed}(\text{N1}, \text{N2}, \text{obligation}(\text{np}, n, \text{det})). \]

will appear in the constraint store.

This (interestingly) failed constraint can be used by the semantic part of our analyzer to correctly extract the meaning of the noun phrase, through reconstructing the missing determiner’s meaning (e.g. as “all” or as “most”).

The following CHR rule achieves this, through consulting semantic, syntactic and ontological information. Notice that it is applicable only if the noun can be determined to be a generic one. The call to generic(PluralNoun) in the rule’s guard serves this purpose, by consulting appropriate domain ontologies.

\[ \text{failed}(\text{N1}, \text{N2}, \text{obligation}(\text{np}, n, \text{det})), \text{icat}(\text{N1}, \text{N2}, \text{np}, \text{plural},_), \text{np}(\text{n}(\text{PluralNoun})), _\text{sem}), \rightarrow \text{generic}(	ext{PluralNoun}) | \text{icat}(\text{N1}, \text{N1}, \text{det}, \text{plural},\_\text{neutral}, \text{det}(\text{every}), \text{np}(\text{n}(\text{PluralNoun})), _\text{sem}), \rightarrow \text{AP1}\text{AP2.ever}(X, @((\text{P1}, X), @(\text{P2}, X))). \]

The semantic argument of icat above is shown in functional representation for readability (as said, we actually use relational equivalents in our code).

Notice that the non-overtness of the determiner “every” is indicated by its word boundaries being the same: it stretches between point N1 and N1 itself. This easy way of recognizing non-overtness can be exploited further if necessary during other aspects of a sentence’s analysis – e.g. relativization, where the antecedent is non-overt.

Since all noun phrase daughters are now explicit for this noun phrase, we can now apply the meaning of “every” over the meaning of “lions” (which will have been propagated from the noun “lions”, i.e. nounSem=lions,AX.lion(X)) and we obtain (modulo notation):

\[ \text{A}\text{P2.ever}(X, \text{lion}(X), @((\text{P2}, X))) \]

Of course, we could have chosen to materialize the implicit determiner as any other one, e.g. as “most” instead of “every”. Ontological interactions with a specific semantic domain’s ontologies can help determine which is best. For instance, while for “Lions are quadrupeds” the correct determiner is indeed “every”, for “Medicines are toxic”, “most” might be more appropriate.

Lexical Meaning Extraction through Ontologies and Failed Properties

Unknown words can be parsed through Womb Grammar by anonymizing their category and features. These will become efficiently instantiated through constraint satisfaction, taking into account all the syntactic and semantic properties that must be satisfied by the unknown word’s interaction with its context.
The clues they can provide regarding syntactic category can serve to guide a subsequent semantic analysis, or to bypass the need for a complete semantic analysis by the concomitant use of ontologies relevant to domain-specific uses of our parser.

For instance, “the resistente virus” includes an ill-typed word which a human would immediately suspect stands for “resistant”. A computer can guess at least its syntactic category through explicitly checking which syntactic constraints fail because of “resistente” not having parsed: given that adjectives must precede nouns, that a noun phrase can have only one head noun, and that determiners are also unique within a noun phrase, the funny word can only be an adjective. The meaning, and perhaps the precise form of the corrected word, can sometimes be determined in consultation with a domain-relevant ontology, in this case of medicine or biology, by marking the word as unknown and letting the semantic construction module consult such ontologies in interaction with the meanings of the known words of the phrase they belong to. In our example, for instance, the similarity with the adjective “resistant” which ontological consultation would semantically link to “virus” may result in proposing it as a possible correction.

Similarly, extraneous words that repeat might allow a domain-dependent ontology to help determine their meaning. For instance, from “his humongous fever” and “the humongous white cell count” by consulting the ontology besides the constraints, we can not only determine that “humongous” is an adjective, but also that it probably refers to some quality similar to “high”. It would be most interesting to carefully study under which conditions such ontological inferences would be warranted.

In general, we are not necessarily interested in capturing the exact meaning of each unrecognized word; but rather in inferring its relation with known words. The problem can be cast into the (automatic) extraction of a portion of the hypernym relation involving the extraneous word using the actual document or additional sources as corpora (see (Clark et al. 2012)). Starting from existing lexical ontologies (e.g. EuroWordNet (Vossen 2004)) different techniques can be used to expand the ontological knowledge about an unknown word (see e.g. (Gupta and Oates 2007; Snow, Jurafsky, and Ng 2006)). Approaches currently described in the literature can be enhanced by means of the clues that Womb Grammar provides upon failure. E.g., one of the most successful techniques involve the search on corpora (or the web) for specific textual patterns indicating relationships between keywords (“a borogove is”) and the fact that the grammar expects a word of a specific category can be used to narrow down the textual patterns to be used to scan the corpus.

**Exploiting failed constraints to complete ontologies** It is worth noting that ontological information can not only be consulted, but in some cases also potentially augmented by a WG analysis of trustworthy text. E.g. in the absence of lexical information about the word “Absettarov”, the parse of a query such as “What illness does the Absettarov virus cause?” should not only classify Absettarov as a proper name (on the grounds of its being capitalized, and of the grammar accepting proper names as adjectives) but should also include Absettarov as a virus (with some indication that this is only postulated, and perhaps some degree of certainty) in the domain-dependent taxonomy the parser is using.

**Flexible word order through Constraint relaxation and failure analysis** Notice that totally permissive word order is very easy to achieve within our framework. This is interesting because discontinuous constituents such as noun phrases are common in some languages and even even in Latin or Greek very common in verse (and not unusual even in certain prose genres, e.g. Plato’s late work, such as the Laws). A contrived example for Latin would be “Puella bona puerum parvum amat” (Good girl loves small boy), where all 5! word permutations are possible, and we certainly do not want to write a separate rule for each of the possible orderings.

In our WG framework, when all permutations are possible, all we have to do is not to include any precedence rule for them! The case in which the contents of constituents may be scrambled up with elements from other constituents can be dealt with by specifying ordering only between those pairs of constituents which must precede each other.

However in our present work we want to describe proper orderings in the host language, say English, while being sensitive to alternative orderings that will eventually be produced by native speakers of another language, say Italian. We deal with this case by stating the English ordering and declaring it as relaxable. This will result in accepting for instance our Italian speaker’s query “What illnesses deadly does the Absettarov virus cause?”. Imperturbable, our analyzer will produce the correct parse tree for the incorrect sentence, while leaving a marker of failed linear precedence between noun and adjective in the form of a constraint. Thus the semantic representation, which is dependent on the parse tree, can be found anyway.

**Other uses of semantico-syntactic constraints**

While interestingly failed constraints have an obvious use for modularly handling exceptions as just exemplified, we can apply the same methodology we described for them above, for refining analyses of structures even if they do not exhibit any failed constraints. For instance, the correct sentences: “Adam gave an apple to Eve” and “Adam gave Eve an apple” differ only in the ordering of the verb’s complements, and in the fact that in the second sentence, such ordering makes the preposition implicit. We can reconstruct it through a constraint that identifies “Eve” as the preposition phrase given that it is animate, while the other candidate (“an apple”) is inanimate.

**Possible Applications and Extensions** We have proposed an analysis of individual sentences which takes semantic and syntactic constraints into account and obtains a parse tree or trees, a list of failed properties and a semantic representation as a result. The semantic representation is full blown unless it is not possible (e.g. due to the
An interesting specialization of our work might be to exploit the fact that ontologies can be consulted freely, whether from the semantic or the syntactic component, and instead of aiming at a full-blown representation, aim at complementing syntax in more minimalistic ways, e.g. by obtaining just a semantically annotated parse tree which might be enough for some applications, or even be as much as we can expect given the domain of application chosen.

One important possible application is that of semantically annotating knowledge bases, e.g. those in the semantic web, with ontological information or little more. Typically such applications do not resort to NLU techniques per se but to hand-produced or only semi-automated approaches.

The construction of knowledge from text is crucial to web mining, and WG parsing might allow us to complement the conventional approaches by adding enough semantic information to better guide the web search. For instance, sub-queries that can be gleaned from a query can be analysed and either evaluated (e.g. "last year" could concretely evaluate to 2014) or independently submitted to a standard Web search engine, such as Google, and the results can then be combined to produce an answer in many cases more accurate than was possible with previous methods. This may allow us for instance to correctly answer queries containing conjunctions and disjunctions that natural language based systems surprisingly fail to understand, as pointed out in (Gottlob 2009).

**Discussion**

The idea of extending Womb Grammars with ontologies was first proposed in (Adebara, Dahl, and Tessaris 2015), where it was designed specifically for completing mixed language grammars, and dealt mainly with syntax. The extension we have presented here is tailored to imperfect querying and in particular incorporates a semantic component, which the previous work does not. We have shown how this approach is especially suitable for ontology-driven enhancements. To the best of our knowledge, this is the first time that the parsing power of constraint failure is exploited to the extent that we do in our work.

The resulting search space reduction is significant because deep parsing with the types of constraint grammars we address is theoretically exponential in the number of categories of the grammar and the size of the sentence to parse (van Rullen 2005).

Other than achieving a considerable search space reduction by only having to evaluate constraints for failure, the use of constraint failure in conjunction with semantics and ontologies allows us to parse imperfect queries in repairing ways, so that they can be effectively answered.

As well, this approach promotes easy interaction between different levels of analysis, opening possibilities for other levels of analysis than just syntax and semantics (e.g. pragmatics) to also interact.

The practice of building a parse tree as a side effect of parsing means that interestingly, also ill-formed sentences can generate a parse tree, which can make the source of failure visually clear.

Clearly, while any of a grammar’s constraints’ failure can be tolerated by our approach (save obligation of a head of phrase, because our parser expands phrases starting from the head), it would not be wise to relax all constraints at the same time: totally erroneous input would be far too intractable to parse. Our compromise solution is to admit those imperfections typical for a given user (e.g. replicating in English the word order due in Italian), which offers a useful enough degree of tolerance without undue extra stress on the parser.

**Related Work**

Other than the background related work already mentioned in this paper, the previous work that most resembles ours is CDG (Foth, Daum, and Menzel 2005), which also replaces well-formedness rules by declarative constraints that integrate different sources of linguistic knowledge, and includes a related mechanism to that of relaxable constraints (defeasible constraints) in order to accept incomplete or incorrect input. Defeasible constraints are more informative than constraint relaxation because they are weighted: they handle constraints of specified scores.

The observed constraint violations in (Foth, Daum, and Menzel 2005) serve to diagnose the input but serve no active role in coming up with appropriate semantics. In our approach, we exploit the combination of these different sources in order to actively determine both syntactic and semantic aspects of the analysis.

Another difference is that in CDG, structure buildup is incrementally achieved by pruning out and adding substructures as a consequence of failed properties. In contrast, we start with minimalistic trees by selecting appropriate subsets of words (just a phrase node and its immediate daughters) as basis for constraint application, and express violated properties explicitly rather than deleting their manifestation in the structure.

Both the CDG approach and our own differ considerably from constraint-based unification grammars such as HPSG, because neither of them include explicit generative rules such as

\[ s \rightarrow np, vp. \]

As discussed, phenomena like free word order is therefore easier to implement, because linear precedence is clearly separated from all other constraints.

With this work we hope to stimulate further research into the uses of ontologies in constraint-based parsing.

**Acknowledgments.** This paper was supported by NSERC Discovery grant 31611024, and developed during a visit to University of Ulm.

**References**

Adebara, I.; Dahl, V.; and Tessaris, S. 2015. Completing mixed language grammars through womb grammars plus ontologies. In Henning Christiansen, M. D. J. L., and


Upward Refinement for Conceptual Blending in Description Logic
— An ASP-based Approach and Case Study in $\mathcal{EL}^{++}$

Roberto Confalonieri, Marco Schorlemmer, Enric Plaza
Artificial Intelligence Research Institute, IIIA-CSIC, Spain
{confalonieri,marco,enric}@iiia.csic.es

Manfred Eppe
International Computer Science Institute, Berkeley, USA
eppe@icsi.berkeley.edu

Oliver Kutz, Rafael Peñaloza
Free University of Bozen-Bolzano, Italy
{oliver.kutz,rafael.penaloza}@unibz.it

Abstract
Conceptual blending is understood to be a process that serves a variety of cognitive purposes, including creativity, and has been highly influential in cognitive linguistics. In this line of thinking, human creativity is modeled as a blending process that takes different mental spaces as input and combines them into a new mental space, called a blend. According to this form of combinatorial creativity, a blend is constructed by taking the existing commonalities among the input mental spaces—known as the generic space—into account, and by projecting their structure in a selective way. Since input spaces for interesting blends are often initially incompatible, a generalisation step is needed before they can be blended. In this paper, we apply this idea to blend input spaces specified in the description logic $\mathcal{EL}^{++}$ and propose an upward refinement operator for generalising $\mathcal{EL}^{++}$ concepts. We show how the generalisation operator is translated to Answer Set Programming (ASP) in order to implement a search process that finds possible generalisations of input concepts. We exemplify our approach in the domain of computer icons.

Introduction
The upward refinement—or generalisation—of concepts plays a crucial role in creative processes for analogical reasoning and concept invention, in particular conceptual blending (Fauconnier and Turner 2002). This paper addresses the formalisation of such a generalisation process and tackles the following question: How can we define and implement a generalisation operator for description logics (DLs) and what are its desired or necessary properties in order to use it in the blending process?

We focus on the particular case of the description logic $\mathcal{EL}^{++}$ (Baader, Brandt, and Lutz 2005; Baader, Brandt, and Lutz 2008). This is an excellent starting point for our investigation for several reasons. First, $\mathcal{EL}^{++}$ is the underpinning logic of the OWL 2 EL Profile, a recommendation of the W3C. Second, $\mathcal{EL}^{++}$ offers a good tradeoff between expressivity and efficiency of reasoning. Indeed, the $\mathcal{EL}^{++}$ syntax and axioms are considered to be sufficiently expressive to model large real-world ontologies. Finally, subsumption in an $\mathcal{EL}^{++}$ TBox is computable in polynomial time (Baader, Brandt, and Lutz 2005).

The generalisation of $\mathcal{EL}^{++}$ concepts has been studied both in the DLs and in the Inductive Logic Programming (ILP) literature, although from different perspectives. Whilst approaches in DL focus on formalising the computation of a least general generalisation (LGG) (also known as least common subsumer) among different concepts as a non-standard reasoning task (Baader 2005; Baader, Sertkaya, and Turhan 2007; Turhan and Zarrieß 2013), approaches in ILP are concerned on learning DL descriptions from examples (Lehmann and Hitzler 2010). In both cases, however, finding a LGG is a challenging task. Its computability depends on the type of DL adopted and on the assumptions made over the structure of concept definitions.

Our work relates to these approaches, but our main motivation for generalising DL concepts is intrinsically different. Although we do need to be aware of what concepts share in order to blend them, it is not necessary (though desirable) to find a generic space that is also a LGG. A sufficiently specific common subsumer will suffice. With this objective in mind, we propose an upward refinement operator for generalising $\mathcal{EL}^{++}$ concepts which is inductively defined over the structure of concept descriptions. We discuss some of the properties typically used to characterise refinement operators; namely, finiteness, properness and completeness (van der Laag and Nienhuys-Cheng 1998). Briefly, a refinement operator is said to be finite when it generates a finite set of refinements; proper, when its refinements are not equivalent to the original concept, and complete, when it produces all possible refinements of a given concept.

Particularly, the generalisations produced by our opera-
A concept description interpretation

<table>
<thead>
<tr>
<th>A</th>
<th>( A^x \subseteq \Delta^x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bot )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( C \land D )</td>
<td>( C^x \cap D^x )</td>
</tr>
<tr>
<td>( \forall r.C )</td>
<td>( { x \in \Delta^x \mid \exists y \in \Delta^x, (x, y) \in \wedge y \in C^x } )</td>
</tr>
</tbody>
</table>

Axiom satisfaction

<table>
<thead>
<tr>
<th>C ( \subseteq ) D</th>
<th>( C^x \subseteq D^x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 \circ \cdots \circ r_n \subseteq r )</td>
<td>( r_1^x : \cdots : r_n^x \subseteq r^x )</td>
</tr>
<tr>
<td>range(r) ( \subseteq ) C</td>
<td>( r^x \subseteq C^x \times \Delta^x )</td>
</tr>
<tr>
<td>domain(r) ( \subseteq ) C</td>
<td>( r^x \subseteq \Delta^x \times C^x )</td>
</tr>
</tbody>
</table>

Table 1: Syntax and semantics of some \( \mathcal{EL}^{++} \) constructors and axioms. (Note: \( \cdot \) is the usual composition operator in relation algebra.)

A knowledge base usually consists of a finite set \( T \) of terminological axioms, called TBox, which contains intensional knowledge defining the main notions relevant to the domain of discourse, and a finite set \( A \) of assertional axioms, called ABox, which contains extensional knowledge about individual objects of the domain. In this paper, we shall focus only on terminological axioms of the form \( C \subseteq D \), i.e. general concept inclusions (GCIs), and \( r_1 \circ \cdots \circ r_n \subseteq r \), i.e. role inclusions (RIs), as well as axioms specifying domain and range restrictions for roles. Table 1 shows the form of these axioms, together with the condition for these to be satisfied by an interpretation \( I \). By \( L(T) \) we refer to the set of all \( \mathcal{EL}^{++} \) concept descriptions we can form with the concept and role names occurring in \( T \).

RIs allow one to specify role hierarchies (\( r \subseteq s \)) and role transitivity (\( r \circ r \subseteq r \)). The bottom concept \( \bot \), in combination with GCIs, allows one to express disjointness of concept descriptions, e.g. \( C \cap D \subseteq \bot \) tells that \( C \) and \( D \) are disjoint. An interpretation \( I \) is a model of a TBox \( T \) iff it satisfies all axioms in \( T \). The basic reasoning task in \( \mathcal{EL}^{++} \) is subsumption. Given a TBox \( T \) and two concept descriptions \( C \) and \( D \), we say that \( C \) is (strictly) subsumed by \( D \) w.r.t. \( T \), denoted as \( C \subseteq T D \) (\( C \subseteq T D \) ), iff \( C^x \subseteq D^x \) (\( C^x \subseteq D^x \) and \( C^x \neq D^x \)) for every model \( I \) of \( T \). Analogously, given two roles \( r, s \in N_r \), we say that \( r \) is (strictly) subsumed by \( s \) w.r.t. \( T \), denoted as \( r \subseteq T s \) (\( r \subseteq T s \) ), iff \( r^x \subseteq s^x \) (\( r^x \subseteq s^x \) and \( r^x \neq s^x \)) for every model \( I \) of \( T \). Finally, an equivalence \( C \equiv D \) is just an abbreviation for \( C \subseteq D \) and \( D \subseteq C \).

### \( \mathcal{EL}^{++} \) Running Example

To exemplify our approach, we take the domain of computer icons into account. We consider computer icons as combinations of signs (e.g. Document, MagnifyingGlass, HardDisk, Pen, etc.) (Confalonieri et al. 2015). Signs are related by qualitative spatial relations such as above, behind, etc.

In the ontology below, concept names are capitalised (e.g. Sign) and role names are written in lower-case (e.g. hasSign). We assume that a TBox \( T \) consists of two parts: one part that contains the background knowledge about the icon domain \( T_{bk} \), and another part that contains the domain knowledge about icon definitions \( T_{dk} \).

Generally, an icon is related to different signs by means of the hasSign role.

- \( \text{hasSign} : \text{Icon} \rightarrow \text{Thing} \)
- \( \text{hasSign} : \text{Sign} \rightarrow \text{Thing} \)
- \( \text{hasSign} : \text{Document} \rightarrow \text{Sign} \)
- \( \text{hasSign} : \text{HardDisk} \rightarrow \text{Sign} \)
- \( \text{hasSign} : \text{MagnifyingGlass} \rightarrow \text{Sign} \)
- \( \text{hasSign} : \text{Pen} \rightarrow \text{Sign} \)

Sign concepts are disjoint (\( \text{hasSign} \neq \text{hasSign} \)) and they are related by spatial relationships isAbove, isBehind, isLeft and isRight that are modelled as roles. These roles are subsumed by a more generic role that is isInSpatialRelation.
whose domain and range is the Sign concept. Spatial relationships are transitive as expressed by axioms:

\[
\begin{align*}
\text{o3a16}: & \quad \text{domain}(\text{isInSpatialRelation}) \sqsubseteq \text{Sign} \\
\text{o3a17}: & \quad \text{range}(\text{isInSpatialRelation}) \sqsubseteq \text{Sign} \\
\text{o3a18}: & \quad \text{isAbove} \sqsubseteq \text{isInSpatialRelation} \\
\text{o3a19}: & \quad \text{isLeft} \sqsubseteq \text{isInSpatialRelation} \\
\text{o3a20}: & \quad \text{isRight} \sqsubseteq \text{isInSpatialRelation} \\
\text{o3a21}: & \quad \text{isAbove} \circ \text{isAbove} \sqsubseteq \text{isAbove} \\
\text{o3a22}: & \quad \text{isInSpatialRelation} \circ \text{isInSpatialRelation} \sqsubseteq \text{isInSpatialRelation}
\end{align*}
\]

Given the axioms above, we can describe some icons in the domain knowledge of a TBox.

**Example 1.** SearchHardDisk is an icon that consists of two signs MagnifyingGlass and HardDisk, where the MagnifyingGlass sign is above the HardDisk sign. Another icon is the EditDocument icon, where the Pen sign is above the Document sign. Both icons are shown in Figure 1:

\[
\begin{align*}
\text{o4a1}: & \quad \text{SearchHardDisk} \equiv \text{Icon} \sqcap \exists \text{hasSign}. \text{HardDisk} \sqcap \\
& \quad \exists \text{hasSign}. (\text{MagnifyingGlass} \sqcap \exists \text{isAbove}. \text{HardDisk}) \\
\text{o4a2}: & \quad \text{EditDocument} \equiv \text{Icon} \sqcap \exists \text{hasSign}. \text{Document} \sqcap \\
& \quad \exists \text{hasSign}. (\text{Pen} \sqcap \exists \text{isAbove}. \text{Document})
\end{align*}
\]

In the paper, we will show how we can create new \(\mathcal{EL}^{++}\) concepts by blending existing concepts in the domain knowledge of a TBox. Specifically, we will see how to generate the following concept representing a new blended icon:

\[
\text{Icon} \sqcap \exists \text{hasSign}. \text{Document} \sqcap \exists \text{hasSign}. (\text{MagnifyingGlass} \sqcap \exists \text{isAbove}. \text{Document})
\]

Intuitively, given two input concepts, a new concept is created by taking the commonalities and some specific parts of the input concepts into account (Figure 1). For instance, both SearchHardDisk and EditDocument are icons that consist of two signs with one sign above the other one (the generic space). Then, if we keep the MagnifyingGlass sign from SearchHardDisk and the Document sign from EditDocument, and we ‘relax’ the HardDisk and Pen signs, we can blend the generalised input concepts of SearchHardDisk and EditDocument into a new concept representing a preview-document icon.\(^1\)

The process of conceptual blending is characterised in terms of amalgamation (Ontañón and Plaza, 2010), an approach that has its root in case-based reasoning and focuses on the problem of combining solutions coming from multiple cases. According to this approach, input concepts are generalised until a generic space is found, and pairs of generalised versions of the input concepts are ‘combined’ to create blends.

\(^1\)Of course, there are some combinations of generalised input concepts that are not interesting. For instance, a concept such as \(\text{Icon} \sqcap \exists \text{hasSign}. \text{MagnifyingGlass} \sqcap \exists \text{hasSign}. (\text{Pen} \sqcap \exists \text{isAbove}. \text{MagnifyingGlass})\) should be discarded. This issue relates to blend evaluation and it is not addressed in this paper. We refer the interested reader to (Confalonieri et al. 2015) where a discussion about the use of argumentation to evaluate conceptual blends is presented.

Figure 1: Blending the SearchHardDisk and EditDocument icon concepts into a new concept representing a preview-document icon.

**Computational concept blending by amalgamation**

Formally, the notion of amalgam can be defined in any representation language \(\mathcal{L}\) for which a subsumption relation between formulas (or descriptions) of \(\mathcal{L}\) can be defined, and therefore also in \(\mathcal{L}(T)\) with the subsumption relation \(\sqsubseteq_T\) for a given \(\mathcal{EL}^{++}\) TBox \(T\).

Given two descriptions \(C_1, C_2 \in \mathcal{L}(T)\), a most general specialisation (MGS) is a description \(C_{\text{mgs}}\) such that \(C_{\text{mgs}} \sqsubseteq_T C_1\) and \(C_{\text{mgs}} \sqsubseteq_T C_2\) and for any other description \(D\) satisfying these properties, \(D \sqsubseteq_T C_{\text{mgs}}\). A least general generalisation (LGG) is a description \(C_{\text{lgg}}\) such that \(C_1 \sqsubseteq_T C_{\text{lgg}}\) and \(C_2 \sqsubseteq_T C_{\text{lgg}}\) and for any other description \(D\) satisfying these properties, \(C_{\text{lgg}} \sqsubseteq_T D\).

Intuitively, a MGS is a description that has all the information in both the original descriptions \(C_1\) and \(C_2\), while a LGG contains that which is common to them.

An amalgam of two descriptions is a new description that contains **parts from these original descriptions**. For instance, an amalgam of ‘a red French sedan’ and ‘a blue German minivan’ is ‘a red German sedan;’ clearly there are always multiple possibilities for amalgams, like ‘a blue French minivan’.

For the purposes of this paper we can define an amalgam of two descriptions as follows:

**Definition 1 (Amalgam).** Let \(T\) be a TBox in \(\mathcal{EL}^{++}\). A description \(C_{\text{am}} \in \mathcal{L}(T)\) is an amalgam of two descriptions \(C_1\) and \(C_2\) (with LGG \(C_{\text{lgg}}\)) if there exist two descriptions \(C'_1\) and \(C'_2\) such that:

1. \(C_1 \sqsubseteq_T C'_1 \sqsubseteq_T C_{\text{lgg}}\),
2. \(C_2 \sqsubseteq_T C'_2 \sqsubseteq_T C_{\text{lgg}}\), and
3. \(C_{\text{am}}\) is a MGS of \(C'_1\) and \(C'_2\).

In the next section, we define an upward refinement operator that allows us to find generalisations of \(\mathcal{EL}^{++}\) concept descriptions needed for computing the amalgams as described above, although we may generalise concepts \(C_1\) and \(C_2\) beyond the LGG \(C_{\text{lgg}}\). We do this to guarantee termination, as we shall explain below.
Refinement Operators

Refinement operators are a well known notion in Inductive Logic Programming where they are used to structure a search process for learning concepts from examples. In this setting, two types of refinement operators exist: specialisation (or downward) refinement operators and generalisation (or upward) refinement operators. While the former constructs specialisations of hypotheses, the latter constructs generalisations.\(^2\)

Generally speaking, refinement operators are defined over quasi-ordered sets. A quasi-ordered set is a pair \((S, \preceq)\) where \(S\) is a set and \(\preceq\) is a binary relation among elements of \(S\) that is reflexive (\(a \preceq a\)) and transitive (if \(a \preceq b\) and \(b \preceq c\) then \(a \preceq c\)). If \(a \preceq b\), we say that \(b\) is more general than \(a\), and if also \(b \preceq a\) we say that \(a\) and \(b\) are equivalent.

A generalisation refinement operator is defined as follows.

**Definition 2.** A generalisation refinement operator \(\gamma\) over a quasi-ordered set \((S, \preceq)\) is a function such that \(\forall a \in S : \gamma(a) \subseteq \{ b \in S \mid a \preceq b \}\).

A refinement operator \(\gamma\) can be classified according to some desirable properties (van der Laag and Nienhuys-Cheng 1998). We say that \(\gamma\) is:

- **locally finite** if the number of generalisations generated for any given element by the operator is finite, that is, \(\forall a \in S : \gamma(a)\) is finite;
- **proper** if an element is not equivalent to any of its generalisations, i.e., \(\forall a, b \in S\), if \(b \in \gamma(a)\), then \(a\) and \(b\) are not equivalent;
- **complete** if there are no generalisations which are not generated by the operator, i.e., \(\forall a, b \in S\) it holds that if \(a \preceq b\), then \(b \in \gamma(a)\) (where \(\gamma(a)\) denotes the set of all elements which can be reached from \(a\) by means of \(\gamma\) in a finite number of steps).

When a refinement operator is locally finite, proper, and complete it is said to be **ideal**.

An ideal specialisation refinement operator for \(EL\) has been explored in (Lehmann and Haase 2010). In what follows, we will define a generalisation refinement operator for \(EL^{++}\) and discuss its properties.

A Generalisation Refinement Operator for \(EL^{++}\)

In any description logic the set of (complex) concept descriptions are ordered under the subsumption relation forming a quasi-ordered set. For \(EL^{++}\) in particular they form a bounded meet-semilattice with conjunction as meet operation and \(\top\) as greatest element as well as \(\bot\) as least element.

In order to define a generalisation refinement operator for \(EL^{++}\) we will need some auxiliary definitions.

**Definition 3.** Let \(T\) be a TBox in \(EL^{++}\). The set of subconcepts of \(T\) is given as

\[
\text{sub}(T) = \bigcup_{C \subseteq D \in T} \text{sub}(C) \cup \text{sub}(D)
\]

where \(\text{sub}\) is inductively defined over the structure of concept descriptions as follows:

\[
\begin{align*}
\text{sub}(A) &= \{A\} \\
\text{sub} (\bot) &= \{\bot\} \\
\text{sub}(\top) &= \{\top\} \\
\text{sub}(C \cap D) &= \{C \cap D\} \cup \text{sub}(C) \cup \text{sub}(D) \\
\text{sub}(\exists r.C) &= \{\exists r.C\} \cup \text{sub}(C)
\end{align*}
\]

Based on \(\text{sub}(T)\), we define the upward cover set of atomic concepts and roles. \(\text{sub}(T)\) guarantees the following upward cover set to be finite.

**Definition 4.** Let \(T\) be a TBox in \(EL^{++}\) (with concept names from \(N_C\)). The set of upward covers\(^3\) of an atomic concept \(A \in N_C \cup \{\top, \bot\}\) and of a role \(r \in N_r\) with respect to \(T\) is given as:

\[
\begin{align*}
\text{UpCov}(A) &= \{C \in \text{sub}(T) \mid A \sqsubseteq_T C \text{ and there is no } C' \in \text{sub}(T) \text{ such that } A \sqsubseteq_T C' \sqsubseteq_T C\} \\
\text{UpCov}(r) &= \{s \in N_r \mid r \sqsubseteq_T s \text{ and there is no } s' \in N_r \text{ such that } r \sqsubseteq_T s' \sqsubseteq_T s\}
\end{align*}
\]

We can now define our generalisation refinement operator for \(EL^{++}\) as follows:

**Definition 5.** Let \(T\) be a TBox in \(EL^{++}\). We define the generalisation operator \(\gamma\) inductively over the structure of concept descriptions as shown in Figure 2.

Given a refinement operator \(\gamma\), \(EL^{++}\) concepts are related by refinement paths as follows:

**Definition 6.** A finite sequence \(C_1, \ldots, C_n\) of concepts is a concept refinement path \(C_1 \sqsubseteq_{\gamma} C_n\) from \(C_1\) to \(C_n\) of a generalisation operator \(\gamma\) if \(C_2 \in \gamma(C_1), \ldots, C_n \in \gamma(C_{n-1})\). A D can be reached from C by \(\gamma\) if there exists a refinement path from C to D. \(\gamma(C)\) denotes the set of all concepts that can be reached from C by means of \(\gamma\). Sometimes we write \(C \sim_{\gamma} D\) instead of \(D \in \gamma(C)\).

That \(\gamma\) is indeed a generalisation refinement operator as defined in Definition 2 can be proven by applying structural induction on \(EL^{++}\) concepts to show that \(C \sim_{\gamma} D\) implies \(C \sqsubseteq D\) in a similar way as in the proof of Proposition 11 from (Lehmann and Hitzler 2010).

As far as the properties of \(\gamma\) are concerned, we can observe the following. Our definition of UpCov for concepts and roles only considers the set of subconcepts present in the TBox \(T\). On the one hand, this guarantees that \(\gamma\) is finite, since at each generalisation step, the set of possible generalisations is finite. On the other hand, however, this implies

---

\(^2\)A deeper analysis of refinement operators can be found in (van der Laag and Nienhuys-Cheng 1998).

\(^3\)Notice that the set of upward covers we define only takes into account subconcepts already present in the TBox. Therefore, strictly speaking, its elements may not be covers with respect to subsumption in \(EL^{++}\).
that γ is not complete, since it cannot find all possible upward covers of a concept w.r.t. subsumption in $\mathcal{EL}^{++}$.  

Regarding the properness property, γ is not proper since there exist cases in which the generalisations produced by γ are equivalent to the concept being generalised. One way to guarantee the properness of γ is by rewriting the concept that we want to generalise into equivalent normal forms before and after each generalisation steps. We will investigate how normalisation can be done as future research.

For the blending, we are interested in finding a common generalisation G (generic space) between two concepts.

**Example 2.** Let us consider the $\mathcal{EL}^{++}$ concepts EditDocument and SearchHardDisk defined in Example 1. It can be checked that:

\[
\text{EditDocument} \rightarrow \text{Icon} \sqcap \exists \text{hasSign}. \text{Sign} \sqcap \exists \text{hasSign}. \text{Sign} \sqcap \exists \text{isAbove}. \text{Sign}.
\]

\[
\text{SearchHardDisk} \rightarrow \text{Icon} \sqcap \exists \text{hasSign}. \text{Sign} \sqcap \exists \text{hasSign}. \text{Sign} \sqcap \exists \text{isAbove}. \text{Sign}.
\]

Therefore, G = Icon \sqcap \exists \text{hasSign}. \text{Sign} \sqcap \exists \text{hasSign}. \text{Sign} \sqcap \exists \text{isAbove}. \text{Sign} is a generic space for EditDocument and SearchHardDisk.

It is worth to discuss that since γ is not complete, we cannot guarantee that the generic space between two $\mathcal{EL}^{++}$ concepts is a least general generalisation. Nevertheless, since the concepts and generalisations that γ finds form a bounded semilattice, we can ensure that we can always find a generic space between two concepts. We believe that not finding a least general generalisation is not a weakness of our approach since we are interested in finding a generic space that allows us to create new $\mathcal{EL}^{++}$ concepts from existing ones by conceptual blending.

**Implementing Upward Refinement in ASP**

We consider two TBoxes $\mathcal{T}_1$ and $\mathcal{T}_2$ and we assume that $\mathcal{T}_1$ and $\mathcal{T}_2$ have the same background knowledge about icon domain but different domain knowledge which, in our case, contain icon definitions.

In order to find a generic space between two icons, we generalise their definitions using the upward refinement operator in Figure 2 implemented in ASP.

The current status of the implementation considers two types of generalisation: γ(A) that generalises an atomic concept by its upward cover and γ(C, r, C) that generalises a concept that fills the range of a role. We reserve the complete implementation of γ as future work. In this setting, the domain knowledge of a TBox $\mathcal{T}$ is generalised in a step-wise transition process. In the following, $t$ denotes a step-counter that represents the number of modifications made to the domain knowledge part of $\mathcal{T}$. Table 2 shows the main EDB and IDB predicates used in the ASP implementation.

**Modeling $\mathcal{EL}^{++}$ concepts in ASP**

For each concept name $A \in N_C$ in a TBox $\mathcal{T}$, we state the facts:

\[
\text{hasConcept}(T, A, t),
\text{isAtomicConcept}(T, A, t),
\text{isBackgroundConcept}(T, A),
\text{isComplexConcept}(T, A, t).
\]

For each role $r \in N_r$ in a TBox $\mathcal{T}$, with domain($r$) $\subseteq C$ and range($r$) $\subseteq D$, we state the facts:

\[
\text{hasRole}(T, r, t),
\text{hasRoleRange}(T, r, D, t),
\text{hasRoleDomain}(T, r, C, t),
\text{isBackgroundRole}(T, r, t).
\]

For each inclusion axiom $A \sqsubseteq B \in T$ and $A, B$ are atomic concepts, we state the fact:

\[
\text{hasParentConcept}(T, A, B, t).
\]

Similarly, for each role inclusion axiom $r \sqsubseteq s \in T$, we state the fact:

\[
\text{hasParentRole}(T, r, s, t).
\]

For each inclusion axiom $A \sqsubseteq C \in T$, in which $A$ is an atomic concept and $C$ is a complex concept, we call $C$ the concept definition of $A$ and denote it as ADef within the following ASP facts:

\[
\text{hasConcept}(T, A, t),
\text{isComplexConcept}(T, A, t),
\text{hasParentConcept}(T, A, A, t),
\text{isBackgroundConcept}(T, A, t).
\]

For each $C_i$ in the complex concept $C = C_1 \sqcap \ldots \sqcap C_n$, we make a case distinction and state the following facts:
EDB predicates | Description
--- | ---
`spec(T)` | A reference to a TBox $T$
`hasConcept(T,C,t)` | A concept $C$ belongs to a TBox $T$ at step $t$
`hasParentConcept(T,A,B,t)` | A concept $B$ subsumes $A$ in a TBox $T$ at step $t$
`isComplexConcept(T,C,t)` | A concept $C$ is a complex concept in a TBox $T$ at step $t$
`complexCInvolvesCon(T,C,A,1)` | A concept $C$ contains a concept $A$ in a TBox $T$ at step $t$
`complexCInvolvesRole(T,C,r,A,t)` | A concept $C$ contains a role $r$ whose range is filled by $A$ in a TBox $T$ at step $t$

IDB predicates | Description
--- | ---
`notEqual(T1,T2,t)` | The TBoxes $T_1$, $T_2$ are not equivalent at step $t$
`conceptsNotEquivalent(T1,T2,t)` | The concepts in the TBoxes $T_1$, $T_2$ are not equivalent at step $t$
`complexCConceptNotEq(T1,T2,C,A,t)` | A concept $A$ in $C$ is not equivalent in the TBoxes $T_1$, $T_2$ at step $t$
`complexCRoleConceptNotEq(T1,T2,C,r,A,t)` | A concept $A$ filling the role $r$ of $C$ is not equivalent in the TBoxes $T_1$, $T_2$ at step $t$
`poss(a,T)` | An upward refinement step $a$ is executable in a TBox $T$ at step $t$
`exec(a,T)` | An upward refinement step $a$ is executed in a TBox $T$ at step $t$

Table 2: Overview of the main EDB and IDB predicates used to formalise the upward refinement process in ASP.

1. if $C_i = B$

   \[ \text{complexCInvolvesCon}(T, \text{Adef}, B, t) \quad (8) \]

2. if $C_i = \exists r. A$

   \[ \text{complexCInvolvesRole}(T, \text{Adef}, r, A, t) \quad (9) \]

3. if $C_i = \exists r. D$

   \[ \text{hasConcept}(T, D\text{def}, t) \quad (10a) \]
   \[ \text{isComplexConcept}(T, D\text{def}, t) \quad (10b) \]
   \[ \text{complexCInvolvesRole}(T, \text{Adef}, r, D\text{def}, t) \quad (10c) \]
   \[ \text{isBackgroundConcept}(T, D\text{def}) \quad (10d) \]

The concepts belonging to the domain knowledge part $\mathcal{T}_d$ of a TBox $T$ are modeled in a similar way but without the `isBackgroundConcept`/3 facts. Besides, we model the concept $\top$ as the fact `hasConcept($T$, Thing, t)`, and for each concept name $A \in N_C$, which is not already subsumed by other concept names, we add a fact `hasParentConcept($T$, A, Thing, t)`. We check for (in)equality of TBoxes by a rule `notEqual($T_1$, $T_2$, t) ← conceptsNotEquivalent($T_1$, $T_2$, t)`. The rule is triggered by additional rules if, for $T_1$ and $T_2$, at step $t$, atomic concepts, roles, and complex concepts are not equal.

**Formalising upward refinement in ASP**

In what follows, we refer to the upward operator types we implemented as $\gamma_A$ and $\gamma_C$. $\gamma_A$ stands for the generalisation of an atomic concept (first row in Fig. 2) and $\gamma_C$ for the generalisation of a concept filling a range of a role. Each upward refinement operator type is an action; to this end, we model each operator type via a precondition rule, an inertia rule, and an effect rule. Preconditions are modelled with a predicate `poss/3` that states when it is possible to execute an operator type. Inertia is modelled with a predicate `noninertial/3` that states when an element of a concept in $\mathcal{T}$ remains unchanged after the execution of a refinement operator type. Effect rules model how a refinement operator type changes a concept in the domain knowledge. We represent the execution of an upward refinement operator type with atoms `exec($\gamma_2$, $\mathcal{T}$, $t$)`, to denote that a generalisation operator type $\gamma_i \in \{\gamma_A, \gamma_C\}$ is applied to $\mathcal{T}$ at step $t$.

**Upward refinement of atomic concepts.** A fact `exec($\text{genConcept}(C, A, B)$, $T$, $t$)` denotes the generalisation of a concept $A$ to a concept $B$ in a complex concept $C$ of a TBox $T$ at step $t$ using $\gamma_A$. The precondition rule for generalising $A$ is:

\[ \text{poss}($\text{genConcept}(C, A, B)$, $T_i$, $t$) ← \quad (11) \]
\[ \neg \text{isBackgroundConcept}(T_i, C), \quad \text{complexCInvolvesCon}(T_i, C, A, t), \quad \text{hasParentConcept}(T_i, A, B, t), \quad \text{complexCConceptNotEq}(T_i, T_2, C, A, t), \quad \neg \text{exec}($\text{genConcept}(C, A, B)$, $T_2$, $t$), \quad \text{spec}(T_2) \]

There are several preconditions for generalising an atomic concept in a complex concept $C$. First, $C$ must not be a background concept since we do not want to modify the background knowledge of a TBox $T$. Second, the concept $C$ involves a concept $A$ that has a parent concept $B$ in the subsumption hierarchy defined by the axioms of the TBox. Third, the definition of $C$ in the TBoxes is not equivalent (`complexCConceptNotEq/4`). The atom `complexCConceptNotEq/4` is true either when one of the TBoxes does not contain $C$ or when the definitions of (the structure of) $C$ are different. Another condition is that $C$ has not been generalised in the other TBox.

We also need a simple inertia rule for generalising a concept in a complex concept. This is as follows:

\[ \text{noninertial}(T, C, A, t) ← \text{exec}($\text{genConcept}(C, A, _)$, $T$, $t$) \quad (12) \]

**noninertial** atoms will cause a concept $A$ to remain in the complex concept $C$ in a TBox $T$, as defined via rule (15a).

**Upward refinement of range concepts.** A fact `exec($\text{genConceptInRole}(C, r, A, B)$, $T$, $t$)` denotes the generalisation of a concept $A$ to a concept $B$ when $A$ fills the range of a role $r$ in a complex concept $C$ of a TBox $T$ at step $t$ using $\gamma_C$. The precondition rule for generalising
a concept $A$ is:

$$\text{poss} \left( \text{genConceptInRole} (C, r, A, B), T_1, t \right) \leftarrow \quad (13)$$

not isBackgroundConcept $(T_1, C)$,

complexCInvolvesRole $(T_1, C, r, A, t)$

hasParentConcept $(T_1, A, B, t)$,

complexCRoleConceptNotEq $(T_1, r, B, spec(T_2))$,

not $\text{exec} \left( \text{genConceptInRole} (C, r, A, B), T_2, t \right)$, $\text{spec}(T_2)$

The preconditions for generalising a concept filling the role of a complex concept $C$ are similar to the case of generalising an atomic concept. First, $C$ must not be a background concept. Second, $C$ involves a role in which the concept to be generalised has a parent concept in the subsumption hierarchy of the TBox. Then, the definitions of the concept to be generalised must not be equivalent in the TBoxes. ($\text{complexCRoleConceptNotEq}$). Another condition is needed in the case of this generalisation, that is, the concept that we want to use to generalise the range of a role, must be in the range of $r$. This is checked by means of the atom $\text{isNotInRoleRange}(C, r, A, t)$. Finally, the concept to be generalised must have not been generalised in the other TBox.

The inertia rule for generalising a concept that fills the range of a role in a TBox is analogous to the inertial rule for generalising a concept:

$$\text{noninertial}(T, C, A, t) \leftarrow \quad (14)$$

$$\text{exec} \left( \text{genConceptInRole} (C, A, t), T, t \right)$$

noninertial atoms will cause a concept $A$ to remain in the range of a role as defined via rule (15b).

**Inertia.** The following rules state which concepts remain in a TBox $T$ when they are inertial.

$$\text{complexCInvolvesCon}(T, C, A, t + 1) \leftarrow \quad (15a)$$

$$\text{complexCInvolvesCon}(T, C, A, t), \quad \text{not} \quad \text{noninertial}(T, C, A, t)$$

$$\text{complexCInvolvesRole}(T, C, r, A, t + 1) \leftarrow \quad (15b)$$

$$\text{complexCInvolvesRole}(T, C, r, A, t), \quad \text{not} \quad \text{noninertial}(T, C, A, t)$$

Besides, other inertia rules are needed for expressing that all the concepts and roles of the background knowledge and their subsumption relations remain in a TBox $T$. We omit them.

**Effects.** The following rules state which concepts change in a TBox $T$ when they are generalised.

$$\text{complexCInvolvesCon}(T, C, B, t + 1) \leftarrow \quad (16a)$$

$$\text{complexCInvolvesCon}(T, C, A, t), \quad \text{exec} \left( \text{genConcept}(C, A, B), T, t \right)$$

$$\text{complexCInvolvesRole}(T, C, r, B, t + 1) \leftarrow \quad (16b)$$

$$\text{complexCInvolvesRole}(T, C, r, A, t), \quad \text{exec} \left( \text{genConceptInRole} (C, r, A, B), T, t \right)$$

**Upward refinement search**

We use ASP for finding a generic space and the generalised versions of the concepts in the domain knowledge of $T$, which can lead to a blend. This is done by successively generalising the concepts in the domain knowledge by means of the upward operator steps we described in the previous subsection. Again, this is a first implementation and does not capture the recursive definition of the upward refinement operator that we leave as future work.

A sequence of generalisation operator types defines a refinement path.

**Definition 7** (Refinement path). Let $T$ be a TBox, let $\{\gamma_1, \ldots, \gamma_n\}$ be the set of generalisation steps for $T$, $t_1 < \cdots < t_n$ be refinement steps and $\gamma_i \in \{\gamma_{A}, \gamma_{C}\}$. The set of atoms $P = \{\text{exec}(\gamma_1, T, t_1), \ldots, \text{exec}(\gamma_n, T, t_n)\}$ is a refinement path of $T$.

Refinement paths are generated with the following choice rule, that allows one or zero refinement operators per $T$ at each step $t$:

$$0 \{\text{exec}(a, T, t) : \text{poss}(a, T, t)\} 1 \leftarrow \quad (17)$$

not genericReached $(t)$, spec $(T)$

The only generalisations that are executed are those whose preconditions are satisfied. Refinement paths lead from the input TBoxes to a generic space, which is a generalised TBox that contains the commonalities of the concepts in the domain knowledge. A generic space is reached, if the generalised TBoxes are equals. The $\text{notEqual}$ predicate is used to determine if a generic space is reached.

$$\text{notGenericReached}(t) \leftarrow \text{spec}(T_1), \text{spec}(T_2), \quad (18a)$$

$$\text{notGenericReached}(t), \text{spec}(T_1), \text{notEqual}(T_1, T_2), \quad \text{notEqual}(T_1, T_2, T_1 \neq T_2) \leftarrow \quad (18b)$$

The constraint (18b) ensures that the generic space is reached in all stable models. The ASP program generates one stable model for each generalisation path, and each refinement step.

**Example 3.** Let us consider the SearchHardDisk and EditDocument concepts in Example 1 representing icons in the domain knowledge of two TBoxes SearchHD and EditDoc. Their refinement paths are:

$$P_{\text{SearchHD}} = \{\text{exec}(\text{genConceptInRole}(\text{SearchHDDef}, \text{hasSign}, \text{HardDisk}, \text{Sign}, \text{SearchHD}, 0)), \text{exec}(\text{genConceptInRole}(\text{SearchHDDefDef}, \text{isAbove}, \text{HardDisk}, \text{Sign}, \text{SearchHD}, 1))\}$$

$$\text{exec}(\text{genConcept}(\text{SearchHDDefDef}), \text{MagnifyingGlass}, \text{Sign}, \text{SearchHD}, 2), P_{\text{EditDoc}} = \{\text{exec}(\text{genConcept}(\text{EditDocDefDef}, \text{Pen}, \text{Sign}, \text{EditDoc}, 0)), \text{exec}(\text{genConceptInRole}(\text{EditDocDefDef}), \text{hasSign}, \text{Document}, \text{Sign}, \text{EditDoc}, 1), \text{exec}(\text{genConceptInRole}(\text{EditDocDefDef}), \text{isAbove}, \text{Document}, \text{Sign}, \text{EditDoc}, 2))\}$$
After applying the respective generalisation operators a generic space is reached. It is easy to check that this corresponds to the generic space in Example 2.

### Blending $\mathcal{EL}^{++}$ concepts

In Definition 1, we defined the blends of two $\mathcal{EL}^{++}$ concepts in terms of amalgams. Once a generic space between two concepts has been determined, blends are created by computing the MGS of pairs of generalised concepts. In $\mathcal{EL}^{++}$ the MGS of two $\mathcal{EL}^{++}$ concepts is just their conjunction. Then we will have to normalise this conjunction to obtain the most concise description of the newly created concept. This is just a rough description of conceptual blending in $\mathcal{EL}^{++}$, and normalisation for newly created blends still needs to be studied in detail (this also relates to the operator properness). Blending will also need to consider an interleaved evaluation and generation process in order to find the best pairs of generalised concepts for creating interesting blends.

**Example 4.** Let us consider $C_1 = \text{SearchHardDisk}$, $C_2 = \text{EditDocument}$, $G$ in Example 2 and the refinement paths $P_{\text{SearchHardDisk}}$, $P_{\text{EditDoc}}$ in Example 3. The generalisations of $C_1$ and $C_2$ by applying the generalisation steps 0-1 in $P_{\text{SearchHardDisk}}$ and step 0 in $P_{\text{EditDoc}}$, respectively are:

$$C_1' = \text{Icon} \sqcap \exists \text{hasSign}.\text{Sign} \sqcap \exists \text{hasSign}.\text{Document}$$

$$(\text{MagnifyingGlass} \sqcap \exists \text{isAbove}.\text{Sign})$$

$$C_2' = \text{Icon} \sqcap \exists \text{hasSign}.\text{Document} \sqcap \exists \text{hasSign}.\text{Document}$$

$$(\text{Sign} \sqcap \exists \text{isAbove}.\text{Document})$$

Then, the conjunction $C_1' \sqcap C_2' = (\text{Icon} \sqcap \exists \text{hasSign}.\text{Sign} \sqcap \exists \text{hasSign}.\text{Document} \sqcap \exists \text{hasSign}.\text{Document} \sqcap \exists \text{hasSign}.\text{Document})$ $$(\text{Sign} \sqcap \exists \text{isAbove}.\text{Document}).$$

$C_1' \sqcap C_2'$ is simplified to obtain the blend concept $\text{Icon} \sqcap \exists \text{hasSign}.\text{Document} \sqcap \exists \text{hasSign}.\text{Document} \sqcap \exists \text{hasSign}.\text{Document}$ by normalisation.

Conceptual blending in $\mathcal{EL}^{++}$ as described in this paper is a special case of blending as modelled in (Bou et al. 2014) and implemented for CASL theories in (Eppe et al. 2015).

### Related Work

There exist several works that relate to ours from different perspectives, that are, approaches to conceptual blending of ontologies, approaches for finding the LGG in the $\mathcal{EL}$ family, and approaches that uses ASP for reasoning over DL ontologies.

Conceptual blending of ontologies in DLs has been explored in (Hois et al. 2010; Kutz et al. 2014) where blends are computed as colimits of algebraic specifications. As such, the blending process is not characterised in terms of amalgamation, the input concepts are not generalised, and the generic space is assumed to be given.

Several approaches for generalising ontology concepts in the $\mathcal{EL}$ family exist in the DLs and ILP literature.

On the one hand, in DL approaches the LGG is defined in terms of a non-standard reasoning task over a TBox (Baader 2003; 2005; Baader, Sertkaya, and Turhan 2007; Zarrieß and Turhan 2013; Turhan and Zarrieß 2013). Generally speaking, since the LGG w.r.t. general TBoxes in the $\mathcal{EL}$ family does usually not exist, these approaches propose several ‘workarounds’ for computing it. For instance, (Baader 2003; 2005) devises the exact conditions for the existence of the LGG for cyclic $\mathcal{EL}$-TBoxes based on graph-theoretic generalisations. (Baader, Sertkaya, and Turhan 2007) propose an algorithm for computing good LGGs w.r.t. a background terminology. (Zarrieß and Turhan 2013; Turhan and Zarrieß 2013) specify the conditions for the existence of the LGG for general $\mathcal{EL}$- and $\mathcal{EL}^{+}$-TBoxes based on canonical models. As already commented in the introduction, our work relates to these approaches, but it is different in spirit, since we do not need to find the LGG between (two) $\mathcal{EL}^{++}$ concepts for the kind of application we are developing.

ILP approaches, on the other hand, study the LGG in terms of generalisation and specialisation refinement operators. Specialisation refinement operators have been defined for learning DL ontologies (Lehmann and Hitzler 2008; Lehmann and Haase 2010) and for measuring the similarity of $\mathcal{EL}$ concepts (Sánchez-Ruiz et al. 2011; 2013).

Finally, some of the approaches that combine ASP for reasoning over DL ontologies are (Swift 2004; Eiter et al. 2008; Ricca et al. 2009).

### Conclusion and Future Works

In this paper, we defined an upward refinement operator for generalising $\mathcal{EL}^{++}$ concepts for conceptual blending. The operator works by recursively traversing their descriptions. We discussed the properties of the refinement operator. The operator is finite, can be proper (by allowing a normalisation before each refinement step), but it is not complete. We claimed, however, that completeness is not an essential property for our needs, since being able to find a generic space between two $\mathcal{EL}^{++}$ concepts, although not a LGG, is already a sufficient condition for conceptual blending.

We presented a first implementation of the refinement operator in ASP. We showed how to model the description of $\mathcal{EL}^{++}$ concepts in ASP and to formalise two refinement operator types for generalising the domain knowledge of a TBox. The stable models of the ASP program contain the generalisation steps needed to be applied in order to generalise two $\mathcal{EL}^{++}$ concepts until a generic space is reached.

We discussed how the blend of two $\mathcal{EL}^{++}$ concepts, defined in terms of their most general specification, can be obtained by their conjunction. We exemplified our approach in the domain of computer icon design.

As future works, we plan to continue with the implementation of the $\mathcal{EL}^{++}$ generalisation operator, to investigate the normalisation rules needed for the operator to be proper and for normalising the blends, as well as to implement a conceptual blending algorithm. We also aim at studying the relationship between the category of CASL theories (Mosses 2004) and signature morphisms and the category of $\mathcal{EL}^{++}$ concept descriptions and subsumption relation.

We consider the work of this paper to be a fundamental step towards the challenging task of defining and implementing an upward refinement operator for more expressive DLs in the context of conceptual blending.
Acknowledgements
This work is partially supported by the COINVENT project (FET-Open grant number: 611553).

References
Turhan, A., and Zarrieß, B. 2013. Computing the lcs w.r.t. general $\mathcal{EL}^\mathcal{+}$-tboxes. In Proceedings of the 26th International Workshop on Description Logics, 477–488.
Extending NoHR for OWL 2 QL

Nuno Costa, Matthias Knorr and João Leite
NOVA LINCS
Departamento de Informática
Universidade NOVA de Lisboa
2829-516 Caparica, Portugal

Abstract
The Protégé plug-in NoHR allows the user to combine an OWL 2 EL ontology with a set of non-monotonic (logic programming) rules – suitable, e.g., to express defaults and exceptions – and query the combined knowledge base (KB). The formal approach realized in NoHR is polynomial (w.r.t. data complexity) and it has been shown that even very large health care ontologies, such as SNOMED CT, can be handled. As each of the tractable OWL profiles is motivated by different application cases, extending the tool to the other profiles is of particular interest, also because these preserve the polynomial data complexity of the combined formalism. Yet, a straightforward adaptation of the existing approach to OWL 2 QL turns out to not be viable. In this paper, we provide the non-trivial solution for the extension of NoHR to OWL 2 QL by directly translating the ontology into rules without any prior pre-processing or classification. We have implemented our approach and our evaluation shows encouraging results.

1 Introduction
NoHR is a plug-in for the ontology editor Protégé that allows its users to query combinations of \$\mathcal{EL}^+\$ ontologies and non-monotonic rules in a top-down manner.

Its motivation stems from the fact that many current ontologies, such as the very large health care ontologies widely used in the area of medicine, e.g., SNOMED CT, are expressed in OWL 2 EL, one of the OWL 2 profiles [Motik et al., 2013], and its underlying description logic (DL) \$\mathcal{EL}^{++}\$ [Baader et al., 2005]. Yet, due to their monotonic semantics, i.e., previously drawn conclusions persist when new additional information is adopted, DL-based ontology languages [Baader et al., 2010] are not suitable to model defaults and exceptions with a closed-world view, a frequently requested feature, e.g., when matching patient records to clinical trial criteria [Patel et al., 2007].

Among the plethora of approaches for extending DLs with non-monotonic features and deal with this problem (c.f. related work in [Eiter et al., 2008; Motik and Rosati, 2010]), NoHR builds on Hybrid MKNF [Motik and Rosati, 2010], which is based on the logic of minimal knowledge and negation as failure (MKNF) [Lifschitz, 1991], under the well-founded semantics [Knorr et al., 2011], a formalism that combines DLs and non-monotonic rules as known from Logic Programming (see also [Alberti et al., 2012] for further motivation in its favor).

This choice is motivated, on the one hand, by the fact that non-monotonic logic programming rules are one of the most well-studied formalisms that admit expressing defaults, exceptions, and also integrity constraints in a declarative way, and are part of RIF [Boley and Kifer, 2013], the other expressive language for the Semantic Web whose standardization is driven by the W3C. On the other hand, Hybrid MKNF provides a very general and flexible framework for combining DL ontologies and non-monotonic rules (see [Motik and Rosati, 2010]). In addition, [Knorr et al., 2011], which is a variant of [Motik and Rosati, 2010] based on the well-founded semantics [Gelder et al., 1991] for logic programs, has a (lower) polynomial data complexity and is amenable for applying top-down query procedures, such as SLG(\$\mathcal{O}\$) [Alférès et al., 2013], to answer queries based only on the information relevant for the query, and without computing the entire model.

NoHR is thus applicable to combinations of non-monotonic rules and OWL 2 EL ontologies. However, other applications (see, e.g., [Calvanese et al., 2011; Savo et al., 2010]) require ontologies using DL constructors which are not covered by OWL 2 EL, such as concept and role negation or role inverses, as admitting these would raise its polynomial complexity [Baader et al., 2005].

OWL 2 QL and the \$DL$-Lite family \cite{Calvanese:2007,Artale:2009} to which the DL underneath OWL 2 QL belongs, \$DL$-Lite\(_B\), is suitable in these cases and has recently drawn a lot of attention in research and in applications.
Even though a simple language at first glance, it is expressible enough to capture basic ontology languages, conceptual data models, e.g., Entity-Relationship, and object-oriented formalisms, e.g., basic UML class diagrams. Reasoning focuses on answering queries by rewriting the initial query, with the help of the ontology, into a set of queries that can be answered using an industry-strength SQL engine over the data. This provides the very low data complexity of LOGSPACE for query answering, but also links directly to applications in ontology-based data access (OBDA) [Calvaneso et al., 2011; Kontchakov et al., 2011]. Altogether, OWL 2 QL is naturally tailored towards huge datasets.

In order to provide also such applications based on OWL 2 QL with the additional expressive power obtained from combining DL ontologies with non-monotonic rules, in this paper, we extend NoHR to OWL 2 QL. Whereas, at first sight, this could seem like a routine exercise, the fact that, to the best of our knowledge, no dedicated open-source OWL 2 QL classifier with OWL API is available, and applying the EL reasoner ELK [Kazakov et al., 2013], currently used in NoHR, to classify a DL-LiteR ontology is obviously not possible, we have to follow a different path here, namely translate the ontology directly into rules. This introduces some non-trivial problems, in particular, the need to capture unsatisfiable concepts and roles and reflexive roles, for which in [Calvaneso et al., 2007] a closure of so-called negative axioms is computed, potentially introducing a huge number of additional axioms. We solve this problem by introducing an extension of the graph, used, e.g., in [Lembo et al., 2013] for classification in OWL QL, to negative axioms. The resulting translation is implemented as a module of the NoHR translator, and its performance evaluated. Our main contributions are:

- A procedure for translating DL-LiteR ontologies into rules which allows answering queries over hybrid KBs combining such ontologies and non-monotonic rules;
- An substantial extension of the Protégé plug-in NoHR to include OWL 2 QL ontologies, beyond DL-LiteR via normalizations, including optimizations on the number of created rules and the use of tabling in the top-down query engine XSB Prolog;\(^5\)
- An evaluation of our extension that shows that NoHR for OWL 2 QL maintains all positive evaluation results of the OWL 2 EL version [Ivanov et al., 2013], and is even faster during pre-processing, as no classification is necessary, in exchange for an on average slightly longer response time during querying.

The remainder of the paper is structured as follows. In Sect. 2, we briefly recall DL-LiteR and MKNF knowledge bases as a tight combination of the former DL and non-monotonic rules. Then, we present the translation of DL-LiteR ontologies which allows us to query such MKNF knowledge bases in Sect. 3. In Sect. 4, we discuss the changes made in the implementation for OWL 2 QL including optimizations, and evaluate it in Sect. 5, before we conclude in Sect. 6.

\(^5\)http://xsb.sourceforge.net

2 Preliminaries

2.1 DL-LiteR

The description logic underlying OWL QL is DL-LiteR, one language of the DL-Lite family [Calvaneso et al., 2007; Artale et al., 2009], which we recall following the presentation in [Knorr and Alfner, 2011].

The syntax of DL-LiteR is based on three disjoint sets of individual names \(N_i\), concept names \(N_c\), and role names \(N_r\). Complex concepts and roles can be formed according to the following grammar

\[
B \rightarrow A | \exists Q \ C \rightarrow B | \neg B \ Q \rightarrow P | P^\alpha \ R \rightarrow Q | \neg Q
\]

where \(A \in N_c\) is a concept name, \(P \in N_r\) a role name, and \(P^\alpha\) its inverse. We also call \(B\) a basic concept, \(Q\) a basic relation, \(C\) a general concept and \(R\) a general role.

A DL-LiteR knowledge base \(\mathcal{O} = (\mathcal{T}, \mathcal{A})\) consists of a TBox \(\mathcal{T}\) and an ABox \(\mathcal{A}\). The TBox contains general inclusion axioms (GCI) of the form \(B \subseteq C\) and role inclusion axioms (RI) of the form \(R \subseteq Q\), with \(B, C, Q,\) and \(R\) defined as above. We term positive inclusion axioms all GCI and RIs in \(\mathcal{O}\) such that \(B\) is a basic concept and \(R\) is a basic relation, respectively, and all other GCI and RIs negative inclusion axioms. We also assume that \(Q^-\) denotes the role \(P\) if \(Q = P^\alpha\), and \(P^-\) if \(Q = P\). The ABox contains assertions of the form \(A(a)\) and \(P(a, b)\) where \(A \in N_c\), \(P \in N_r\), and \(a, b \in N_i\). Assertions \(C(a)\) for general concepts \(C\) can be included by \(A \sqsubseteq C\) and \(A(a)\) for a new concept name \(A\).

The semantics of DL-LiteR is based on interpretations \(\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})\) consisting of a nonempty interpretation domain \(\Delta^\mathcal{I}\) and an interpretation function \(\cdot^\mathcal{I}\) that assigns to each individual \(a\) a distinct element \(a^\mathcal{I}\) of \(\Delta^\mathcal{I}\), to each concept name \(N_i\) a subset \(N_i^\mathcal{I}\), and to each role name \(P\) a binary relation \(P^\mathcal{I}\) over \(\Delta^\mathcal{I}\). This can be extended as usual:

\[
\begin{align*}
(P^-)^\mathcal{I} &= \{(i_2, i_1) | (i_1, i_2) \in P^\mathcal{I}\} \\
(-B)^\mathcal{I} &= \Delta^\mathcal{I} \setminus B^\mathcal{I} \\
(Q)^\mathcal{I} &= \{(i, i') | i, i' \in Q^\mathcal{I}\} \\
(-Q)^\mathcal{I} &= \Delta^\mathcal{I} \times \Delta^\mathcal{I} \setminus Q^\mathcal{I}
\end{align*}
\]

An interpretation \(\mathcal{I}\) is a model of GCI \(B \subseteq C\) and of RI \(Q \subseteq R\) if \(B^\mathcal{I} \subseteq C^\mathcal{I}\) and \(Q^\mathcal{I} \subseteq R^\mathcal{I}\) respectively. \(\mathcal{I}\) is also a model of an assertion \(A(a)\) \((P(a, b))\) if \(a^\mathcal{I} \in A^\mathcal{I}\) \((P(a, b)^\mathcal{I})\). Given an axiom/assertion \(\alpha\) we denote by \(\mathcal{I} \models \alpha\) that \(\mathcal{I}\) is a model of \(\alpha\). A model of a DL-LiteR KB \(\mathcal{O} = (\mathcal{T}, \mathcal{A})\) is an interpretation \(\mathcal{I}\) such that \(\mathcal{I} \models \alpha\) holds for all \(\alpha \in \mathcal{T} \cup \mathcal{A}\), and \(\mathcal{O}\) is satisfiable if it has at least one model, and unsatisfiable otherwise. Also, \(\mathcal{O}\) entails axiom \(\alpha\), written \(\mathcal{O} \models \alpha\), if every model of \(\mathcal{O}\) satisfies \(\alpha\).

2.2 MKNF Knowledge Bases

MKNF knowledge bases (KBs) build on the logic of minimal knowledge and negation as failure (MKNF) [Lifschitz, 1991]. Two main different semantics have been defined [Motik and Rosati, 2010; Knorr et al., 2011], and we focus on the well-founded version [Knorr et al., 2011], due to its lower computational complexity and amenability to top-down querying without computing the entire model. Here, we only point out

\(^6\)Hence, the unique name assumption is applied and, as shown in [Artale et al., 2009], dropping it would increase significantly the computational complexity of DL-LiteR.
important notions following [Ivanov et al., 2013], and refer to
[Knorr et al., 2011] and [Alferes et al., 2013] for the details.

We start by recalling MKNF knowledge bases as presented in
[Alferes et al., 2013] to combine an ontology and a set of non-monotonic rules (similar to a normal logic program).

**Definition 1.** Let O be an ontology. A function-free first-order atom \( P(t_1, \ldots, t_n) \) s.t. \( P \) occurs in \( O \) is called DL-atom; otherwise non-DL-atom. A rule \( r \) is of the form

\[
H \leftarrow A_1, \ldots, A_n, \text{not}B_1, \ldots, \text{not}B_m \tag{1}
\]

where the head of \( r \), \( H \), and all \( A_i \) with \( 1 \leq i \leq n \) and \( B_j \) with \( 1 \leq j \leq m \) in the body of \( r \) are atoms. A program \( \mathcal{P} \) is a finite set of rules, and an MKNF knowledge base \( \mathcal{K} \) is a pair \((O, \mathcal{P})\). A rule \( r \) is \( DL\)-safe if all its variables occur in at least one non-DL-atom \( A_i \) with \( 1 \leq i \leq n \), and \( \mathcal{K} \) is \( DL\)-safe if all its rules are \( DL\)-safe.

\( DL\)-safety ensures decidability of reasoning with MKNF knowledge bases and can be achieved by introducing a new predicate \( o \), adding \( o(i) \) to \( \mathcal{P} \) for all constants \( i \) appearing in \( \mathcal{K} \) and, for each rule \( r \in \mathcal{P} \), adding \( o(X) \) for each variable \( X \) appearing in \( r \) to the body of \( r \). Therefore, we only consider \( DL\)-safe MKNF knowledge bases.

**Example 1.** Consider an MKNF knowledge base \( \mathcal{K} \) as given below for recommending CDs adapted from [Knorr et al., 2011] (with some modifications). We denote DL-atoms and constants with upper-case names and non-DL-atoms and variables with lower-case names.

\[
\text{HasArtist} \subseteq \text{Artist} \quad \text{Piece} \subseteq \text{HasArtist}
\]

\[
\text{HasComposed} \subseteq \text{Piece} \quad \text{Artist} \ni \text{not} \text{Piece}
\]

\[
\text{recommend}(x) \leftarrow \text{Piece}(x), \text{not} \text{owns}(x), \text{not} \text{lowEval}(x), \text{interesting}(x)
\]

\[
\text{interesting}(x) \leftarrow \text{Piece}(x), \text{not} \text{owns}(x), \text{Piece}(y), \text{owns}(y), \text{Artist}(z), \text{HasArtist}(y, z), \text{HasArtist}(x, z), \text{owns}(\text{Summertime}), \text{Piece}(\text{Summertime}) \leftarrow \text{HasArtist}(\text{Summertime}, \text{Gershwin})
\]

\[
\text{HasComposed}(\text{Gershwin}, \text{RhapsodyInBlue}) \leftarrow
\]

This example shows that we can seamlessly express defaults and exceptions, such as recommending pieces as long as they are not owned or having a low evaluation, and at the same time taxonomical/ontological knowledge including information over unknown individuals, such as every piece having at least one artist without having to specify whom, but also features of DL-Lite\(_R\), such as domain and range restrictions (of roles).

The semantics of MKNF knowledge bases \( \mathcal{K} \) is usually given by a translation \( \pi \) into an MKNF formula \( \pi(\mathcal{K}) \), i.e., a translation over first-order logic extended with two modal operators \( \mathcal{K} \) and \( \text{not} \). Namely, every rule of the form \( (1) \) is translated into \( \mathcal{K}H \leftarrow \mathcal{KA}_1, \ldots, \mathcal{KA}_n, \text{not}B_1, \ldots, \text{not}B_m, \pi(\mathcal{P}) \) is the conjunction of the translations of its rules, and \( \pi(\mathcal{K}) = \pi(O) \land \pi(\mathcal{P}) \) where \( \pi(\mathcal{O}) \) is the first-order translation of \( O \). Reasoning with such MKNF formulas is then commonly achieved using a partition of modal atoms, i.e., all expressions of the form \( \mathcal{K}A_i \) for each \( K \) or \( \text{not}_A \) occurring in \( \pi(\mathcal{K}) \). For [Knorr et al., 2011], such a partition assigns true, false, or undefined to (modal) atoms, and can be effectively computed in polynomial time. If \( \mathcal{K} \) is MKNF-consistent, then this partition does correspond to the unique model of \( \mathcal{K} \) [Knorr et al., 2011], and, like in [Alferes et al., 2013], we call the partition the well-founded MKNF model \( \text{Maf}(\mathcal{K}) \). Here, \( \mathcal{K} \) may indeed not be MKNF-consistent if the ontology alone is unsatisfiable, or by the combination of appropriate axioms in \( O \) and rules in \( \mathcal{P} \), e.g., \( A \ni \text{not}B \), and \( \text{A}(a) \leftarrow \text{B}(a) \). Strictly speaking, unlike [Ivanov et al., 2013], we do not have to make assumptions on the satisfiability of \( O \) as we are not going to use a classifier when processing DL-Lite\(_R\) ontologies. Still, for the technical results established in Sec. 3, we will rely on satisfiability, since we are able to entail everything from an unsatisfiable \( O \), whereas the translation into rules defined in Sec. 3 would not permit that. This is why in the following, we assume that \( O \) occurring in \( \mathcal{K} \) is satisfiable, which does not truly constitute a restriction as we can always turn the ABox into rules without any effect on \( \text{Maf}(\mathcal{K}) \). An alternative approach would be to use one of the paraconsistent semantics for MKNF knowledge bases [Kaminski et al., 2015], but this is outside the scope of this paper, and an issue for future work as currently no paraconsistent correspondence to the querying procedure SLG(\( O \)) used here exists.

### 2.3 Querying in MKNF Knowledge Bases

In [Alferes et al., 2013], a procedure, called SLG(\( O \)), is defined for querying MKNF knowledge bases under the well-founded MKNF semantics. This procedure extends SLG resolution with tabling [Chen and Warren, 1996] with an oracle to \( O \) that handles ground queries to the DL-part of \( \mathcal{K} \) by returning (possibly empty) sets of atoms that, together with \( O \) and information already proven true, allows us to derive the queried atom. We refer to [Alferes et al., 2013] for the full account of SLG(\( O \)), and only recall a few crucial notions necessary in the following.

SLG(\( O \)) is based on creating top-down derivation trees with the aim of answering (DL-safe) conjunctive queries \( Q = q(X) \leftarrow A_1, \ldots, A_n, \text{not}B_1, \ldots, \text{not}B_m \) where each variable in \( Q \) occurs in at least one non-DL atom in \( Q \), and where \( X \) is the (possibly empty) set of requested variables appearing in the body.

In general, the computation of \( \text{Maf}(\mathcal{K}) \) uses two different versions of \( \mathcal{K} \) in parallel to guarantee that a coherence is ensured, i.e., if \( \text{not}P(\alpha) \) is derivable, then \( \text{not}P(\alpha) \) has to be true as well (cf. also [Knorr et al., 2011]), and b) MKNF-consistency of \( \mathcal{K} \) can be verified. For a top-down approach this is impractical, so, instead, a doubled MKNF knowledge base \( \mathcal{K}^d = (O, O^d, P^d) \) is defined in which a copy of \( O \) with new doubled predicates is added, and two rules occur in \( P^d \) for each rule in \( P \), intertwining original and doubled predicates (see Def. 3.1 in [Alferes et al., 2013]). It is shown that an atom \( A \) is true in \( \text{Maf}(\mathcal{K}) \) iff \( A \) is true in \( \text{Maf}(\mathcal{K}^d) \) and \( A \) is false in \( \text{Maf}(\mathcal{K}) \) iff \( A^d \) is false in \( \text{Maf}(\mathcal{K}^d) \). Note that \( \mathcal{K}^d \) is necessary in general, but we can use \( \mathcal{K} \) here if it contains no negative inclusion axioms.
In [Alferes et al., 2013], the notion of oracle is defined to handle ground queries to the ontology, but before we recall that notion, we use an example to illustrate the idea.

**Example 2.** Recall $K$ in Ex. I. Here, we omit $K^d$ and restrict ourselves to $K$, which suffices our purposes. Consider query $q = \text{recommend}(\text{Summertime})$. By instantiating the body of the matching rule head in $K$ with $x = \text{Summertime}$, we obtain two new queries. The first one, $\text{Piece}(\text{Summertime})$, can be answered by means of the rule with matching head. The second, $\text{notowns}(\text{Summertime})$, is handled by querying for owns($\text{Summertime}$), for which a corresponding rule exists, so $\text{notowns}(\text{Summertime})$ fails, hence $q$ is false.

Consider $q_1 = \text{recommend}(\text{RhapsodyInBlue})$. Using the same rule with matching rule head we obtain two new instantiated queries from the rule body. Now, $\text{Piece}(\text{RhapsodyInBlue})$ cannot be derived from the rules, but we can query the ontology and the oracle will return, e.g., a query $\text{HasComposed}(x, \text{RhapsodyInBlue})$ that if proven true can be added to $\mathcal{O}$, which would allow us to derive the queried goal. This query succeeds because of $\text{HasComposed}(\text{Gershwin, RhapsodyInBlue}) \iff$, and so does $\text{Piece}(\text{RhapsodyInBlue})$. Then, we cannot prove owns(\text{RhapsodyInBlue}) nor lowEval(\text{RhapsodyInBlue}), so both fail, succeeding their (default) negated queries. For the remaining new query interesting(\text{RhapsodyInBlue}), the second rule head matches, creating further subgoals. The first two were just answered, as the next two with $y = \text{Summertime}$ for $q$. The remaining also follow from the interplay of $\mathcal{O}$ and $\mathcal{P}$ in $\mathcal{K}$, so $q_1$ succeeds.

We recall the notions of a complete and a (correct) partial oracle from [Alferes et al., 2013].

**Definition 2.** Let $\mathcal{K}^d = (\mathcal{O}, \mathcal{O}^d, \mathcal{P}^d)$ be a doubled MKNF KB, $\mathcal{I}$ a set of ground atoms (already proven to be true), $S$ a ground query, and $\mathcal{L}$ a set of ground atoms such that each $L \in \mathcal{L}$ is unifiable with at least one rule head in $\mathcal{P}^d$. The complete oracle for $\mathcal{O}$, denoted $\text{compTo}(\mathcal{I}, \mathcal{S}, \mathcal{L})$ iff $\mathcal{O} \cup \mathcal{I} \cup \mathcal{L} \models S$ or $\mathcal{O}^d \cup \mathcal{I} \cup \mathcal{L} \models S$. A partial oracle for $\mathcal{O}$, denoted $\text{pTo}(\mathcal{I}, \mathcal{S}, \mathcal{L})$ such that if $\mathcal{P}(\mathcal{I}, \mathcal{S}, \mathcal{L})$, then $\mathcal{O} \cup \mathcal{I} \cup \mathcal{L} \models S$ or $\mathcal{O}^d \cup \mathcal{I} \cup \mathcal{L} \models S$ for consistent $\mathcal{O} \cup \mathcal{I} \cup \mathcal{L}$ and $\mathcal{O}^d \cup \mathcal{I} \cup \mathcal{L}$, respectively.

A partial oracle $\text{pTo}$ is correct w.r.t. $\text{compTo}$ iff, for all MKNF-consistent $\mathcal{K}^d$, replacing $\text{compTo}$ in $\text{SLG}(\mathcal{O})$ with $\text{pTo}$ succeeds for exactly the same set of queries.

Partial oracles may avoid returning unnecessary answers $\mathcal{L}$, such as non-minimal answers or those that try to derive an MKNF-inconsistency even though $\mathcal{K}^d$ is MKNF-consistent. Also, correctness of partial oracles is only defined w.r.t MKNF-consistent $\mathcal{K}$. The rationale is that, when querying top-down, we want to avoid checking whether the entire KB $\mathcal{K}^d$ is MKNF-consistent. This leads to para-consistent derivations if $\mathcal{K}^d$ is not MKNF-consistent, e.g., some atom $P$ is true, yet $\mathcal{P}^d$ is false, while other independent atoms are evaluated as if $\mathcal{K}^d$ was MKNF-consistent (see [Alferes et al., 2013]).

## Translating the Ontology into Rules

As argued for the case of $\mathcal{EL}^+_d$ [Ivanov et al., 2013], axioms with $\exists$ on the right-hand side, e.g., $\text{Piece} \subseteq \exists \text{HasArtist}$, cannot be translated straightforwardly into rules, nor do they directly contribute to the result when querying for ground instances, e.g., of $\text{HasArtist}(x, y)$. Still, such axioms may contribute to derivations within $\mathcal{O}$, which is why, in [Ivanov et al., 2013], a classification using the dedicated and highly efficient $\mathcal{EL}$ reasoner ELK [Kazakov et al., 2013] is first applied to derive implicit consequences. These, together with all axioms in $\mathcal{O}$, are then translated into rules, now discarding certain axioms with $\exists$ on the right-hand side.

Here, since to the best of our knowledge no dedicated, open-source OWL 2 QL classifier with OWL API is available, we opt to follow a different path, namely translate the ontology directly into rules. This also simplifies and shortens the preprocessing phase and avoids a priori-classification, but requires some non-trivial considerations to ensure that no derivations are lost in the process, which we will explain next.

Essentially, axioms, such as $\text{Piece} \subseteq \exists \text{HasArtist}$, cannot be translated into a rule $\text{HasArtist}(x, y) \iff \text{Piece}(x)$ using a universal variable $y$, as this would allow us to derive $\text{HasArtist}(x, y)$ for any $\text{Piece}(x)$ and $y$, which is clearly not what the axiom expresses. Using a new constant $c$ instead of $y$ would not be correct either, as querying for $\text{HasArtist}(x, y)$ would return $\text{HasArtist}(x, c)$ for any $\text{Piece}(x)$ for the same $c$. Therefore, we proceed differently by introducing new auxiliary predicates that intuitively represent the domain and range of roles. For our example, this will yield the rule $\text{DHasArtist}(x, y) \iff \text{Piece}(x)$ where $\text{DHasArtist}$ stands for the domain of $\text{HasArtist}$ (and $\text{RHasArtist}$ its range). Using such auxiliary predicates also means that we have to make sure that, e.g., $\text{HasArtist}(\text{Summertime, Gershwin})$ allows us to derive $\text{DHasArtist}(\text{Summertime})$, which can be achieved via an additional rule $\text{DHasArtist}(x) \iff \text{HasArtist}(x, y)$. Moreover, for $\text{HasComposed} \subseteq \exists \text{HasArtist}$, it does not suffice to translate the axiom to $\text{HasArtist}(x, y) \iff \text{HasComposed}(y, x)$, but also link the new auxiliary predicates for both roles, by adding, $\text{DHasArtist}(x) \iff \text{RHasComposed}(x)$ and $\text{RHasArtist}(x) \iff \text{DHasComposed}(x)$.

We now formalize this translation, and we start by introducing notation on how to translate general concepts and roles. For that purpose, we formally introduce for each role $P \in \mathcal{NR}$ auxiliary predicates $DP$ and $RP$ with the intuition of representing the domain and range of $P$. Also, similar to previous work in [Alferes et al., 2013; Ivanov et al., 2013], we use special atoms $NH(i)$ in $\text{SLG}(\mathcal{O})$ that represent a query $\neg H(i)$ to the oracle. These are, of course, only relevant if $\mathcal{O}$ contains negative inclusion axioms.

**Definition 3.** Let $C$ be a concept, $R$ a role, $x$ and $y$ variables, and $v$ a new (anonymous) variable (disjoint from $x$ and $y$).
We define $tr(C, x)$ and $tr(R, x, y)$ as follows:

$$
tr(C, x) =
\begin{cases}
  A(x) & \text{if } C = A \\
  DP(x) & \text{if } C = \exists P \\
  RP(x) & \text{if } C = \exists P^- \\
  NA(x) & \text{if } C = \neg A \\
  (tr(\neg Q, x, v)) & \text{if } C = \neg \exists Q
\end{cases}
$$

$$
tr(R, x, y) =
\begin{cases}
  P(x, y) & \text{if } R = P \\
  P(y, x) & \text{if } R = P^- \\
  NP(x, y) & \text{if } C = \neg P \\
  NP(y, x) & \text{if } C = \neg P^-
\end{cases}
$$

We obtain $tr^d(C, x)$ and $tr^d(Q, x, y)$ from $tr(C, x)$ and $tr(Q, x, y)$ by substituting all predicates $P$ in $tr(C, x)$ and $tr(Q, x, y)$ with $P^d$, respectively.

$tr(C, x)$ and $tr(R, x, y)$ handle both positive and negative inclusions and no additional case distinction is necessary.

Before we present the actual translation, we need to introduce one central notion, namely a graph to represent the axioms in a given TBox $\mathcal{T}$ as well as the implicitly derivable axioms, which will be necessary for defining the translation itself, but also turn out useful when establishing the correctness of the translation. Graphs have been used for classification in OWL QL (of positive inclusion axioms) [Lembo et al., 2013], and we extend the notion here to also take negative inclusion axioms into account. We thus introduce the digraph (directed graph) of $\mathcal{T}$ as follows.

**Definition 4.** Let $\mathcal{T}$ be a DL-Lite$_R$ TBox. The digraph of $\mathcal{T}$, $\mathcal{G}_\mathcal{T} = (\mathcal{V}, \mathcal{E})$, is constructively defined as follows.

1. If $A \in N_C$, then $A$ and $\neg A$ are in $\mathcal{V}$;
2. If $R \in N_R$, then $P$, $\exists P$, $\exists P^-$, $\neg P$, $\neg \exists P$, and $\neg P^-$ are in $\mathcal{V}$;
3. If $B_1 \sqsubseteq B_2 \in \mathcal{T}$, then the edges $(B_1, B_2)$ and $(\neg B_2, \neg B_1)$ are in $\mathcal{E}$;
4. If $Q_1 \sqsubseteq Q_2 \in \mathcal{T}$, then the edges $(Q_1, Q_2)$, $(Q_1^-, Q_2^-)$, $(\exists Q_1, \exists Q_2)$, $(\exists Q_1^-, \exists Q_2^-)$, $(\neg Q_2, \neg Q_1)$, $(\neg Q_2^-, \neg Q_1^-, Q_1)$, $(\neg \exists Q_2, \neg \exists Q_1)$, $(\neg \exists Q_2^-, \neg \exists Q_1^-)$ are in $\mathcal{E}$;
5. If $B_1 \sqsubseteq B_2 \in \mathcal{T}$, then the edges $(B_1, \neg B_2)$ and $(\neg B_2, \neg B_1)$ are in $\mathcal{E}$;
6. If $Q_1 \sqsubseteq \neg Q_2 \in \mathcal{T}$, then the edges $(Q_1, \neg Q_2)$, $(Q^-_1, \neg Q^-_2)$, $(\exists Q_1, \neg \exists Q_2)$, $(\exists Q^-_1, \neg \exists Q^-_2)$ are in $\mathcal{E}$.

Basically, each possible general concept and general role over $N_C$ and $N_R$ is a node in $\mathcal{G}_\mathcal{T}$, and the directed edges represent logical implications that follow from the axioms. Namely, for items 3. and 5., the subset inclusion itself and its contrapositive are in $\mathcal{E}$, and this is similar for items 4. and 6., only that the additional combinations due to inverses, $\exists$, and $\neg$ have to be taken into account. In this sense, the graph can be understood as capturing all subset inclusions (explicit and implicit) in $\mathcal{O}$, i.e., whenever there is a path from concept $C_1$ to concept $C_2$ and from role $R_1$ to role $R_2$, then $C_1 \sqsubseteq C_2$ and $R_1 \sqsubseteq R_2$ hold respectively. An Example of such a digraph is given in Fig. 1 for the TBox $\mathcal{T}$ from Example 1.

One observation to be made in Fig. 1, is that $\exists HasComposed \sqsubseteq \neg \exists HasComposed$, i.e., $HasComposed$ is irreflexive. Even though this does not entail any assertion, knowing that $\forall x. \neg HasComposed(x, x)$ does hold should be captured in the translation. We introduce $\Psi(\mathcal{T})$, the set of irreflexive roles in $\mathcal{T}$, to be able to ensure exactly that.

**Definition 5.** Let $\mathcal{T}$ be a DL-Lite$_R$ TBox and $\mathcal{G}_\mathcal{T}$ its digraph. We define $\Psi(\mathcal{T})$ as the smallest set of all $P \in N_R$ that satisfy at least one of the following conditions:

1. For some $B_1 \sqsubseteq B_2 \in \mathcal{T}$, there exist paths from $\exists P$ to $B_1$ and from $\exists P^-$ to $B_2$;
2. For some $B_1 \sqsubseteq \neg B_2 \in \mathcal{T}$, there exist paths from $\exists P^-$ to $B_1$ and from $\exists P$ to $B_2$;
3. For some $Q_1 \sqsubseteq \neg Q_2 \in \mathcal{T}$, there exist paths from $P$ to $Q_1$ and from $P^-$ to $Q_2$;
4. For some $Q_1 \sqsubseteq \neg Q_2 \in \mathcal{T}$, there exist paths from $P^-$ to $Q_1$ and from $P$ to $Q_2$.

This notion builds on $\mathcal{G}_\mathcal{T}$, which is also required for detecting a further set of derivations. Imagine we would (wrongfully) add $Art\text{ist} \sqsubseteq \exists HasComposed^\neg$ to $\mathcal{O}$ in Example 1. Then there would be a path from $Art\text{ist}$ to both $Piece$ and $\neg Piece$, i.e., the concept $Art\text{ist}$ would be unsatisfiable. Note that independently of whether the hybrid KB is MKKNF-inconsistent or not, we need to make sure that all unsatisfiable concepts and roles are determined, so we introduce $\Omega(\mathcal{T})$, quite similar in spirit to $\Psi(\mathcal{T})$.

**Definition 6.** Let $\mathcal{T}$ be a DL-Lite$_R$ TBox and $\mathcal{G}_\mathcal{T}$ its digraph. We define $\Omega(\mathcal{T})$ as the smallest set of all $A \in N_C$ such that, for some $B_1 \sqsubseteq \neg B_2 \in \mathcal{T}$, there exist paths from $A$ to both $B_1$ and $B_2$, and all $P \in N_R$ that satisfy at least one of the following conditions:

1. For some $B_1 \sqsubseteq \neg B_2 \in \mathcal{T}$, there exist paths from $\exists P$ to both $B_1$ and $B_2$;
2. For some $B_1 \sqsubseteq \neg B_2 \in \mathcal{T}$, there exist paths from $\exists P^-$ to both $B_1$ and $B_2$;
3. For some $Q_1 \sqsubseteq \neg Q_2 \in \mathcal{T}$, there exist paths from $P$ to both $Q_1$ and $Q_2$;
4. For some $Q_1 \sqsubseteq \neg Q_2 \in \mathcal{T}$, there exist paths from $P^-$ to both $Q_1$ and $Q_2$.

With all pieces in place, we can finally introduce the definition of the translation of a DL-Lite$_R$ ontology into rules.

**Definition 7.** Let $\mathcal{O}$ be a DL-Lite$_R$ ontology. We define $\mathcal{T}^\mathcal{O}$ from $\mathcal{O}$, where $B_1$, $B_2$ are basic concepts, $Q_1$, $Q_2$ basic...
roles, $x$, $y$ variables, and $a$, $b$ individuals, as the smallest set containing:

(e) for every $P \in N_R$:

\[ DP(x) \Leftarrow P(x, y) \quad DP^{d}(x, y) \Leftarrow P^{d}(x, y) \quad RP(y) \Leftarrow P(x, y) \quad RP^{d}(y) \Leftarrow P^{d}(x, y) \]

(a1) for every $A(a) \in D$:

\[ A(a) \Leftarrow \neg \text{NA}(a) \]

(a2) for every $P(a, b) \in O$:

\[ P(a, b) \Leftarrow \neg\text{NP}(a, b) \]

(s1) for every $B_1 \sqsubseteq B_2 \in O$:

\[ \text{tr}(B_2, x) \Leftarrow \text{tr}(B_1, x) \quad \text{tr}^{d}(B_2, x) \Leftarrow \text{tr}^{d}(B_1, x), \neg \text{tr}^{d}(\neg B_2, x) \]

\[ \text{tr}(\neg B_1, x) \Leftarrow \text{tr}(\neg B_2, x) \]

\[ \forall Q_1 \sqsubseteq Q_2 \in O: \quad \text{tr}(Q_2, x, y) \Leftarrow \text{tr}(Q_1, x, y), \neg \text{tr}^{d}(\neg Q_2, x, y) \]

\[ \text{tr}(\exists Q_2, x) \Leftarrow \text{tr}(\exists Q_1, x), \neg \text{tr}(\exists Q_2, x) \]

\[ \text{tr}(\exists Q_2, x) \Leftarrow \text{tr}(\exists Q_1, x), \neg \text{tr}(\exists Q_2, x) \]

\[ \text{tr}(\neg Q_1, x, y) \Leftarrow \text{tr}(\neg Q_2, x, y) \]

\[ \forall B_1 \sqsubseteq B_2 \in O: \quad \text{tr}(\neg B_1, x) \Leftarrow \text{tr}(\neg B_2, x) \]

\[ \text{tr}(\neg B_2, x) \Leftarrow \text{tr}(B_1, x) \]

\[ \forall Q_1 \sqsubseteq Q_2 \in O: \quad \text{tr}(\exists Q_2, x) \Leftarrow \text{tr}(\exists Q_1, x), \neg \text{tr}^{d}(\neg Q_2, x) \]

\[ \text{tr}(\exists Q_2, x) \Leftarrow \text{tr}(\exists Q_1, x), \neg \text{tr}(\exists Q_2, x) \]

\[ \text{tr}(\neg Q_1, x, y) \Leftarrow \text{tr}(\neg Q_2, x, y) \]

\[ \forall A \in \Omega(T): \quad \text{NA}(A(x)) \Leftarrow \]

\[ \forall P \in \Omega(T): \quad \text{NP}(P(x, y)) \Leftarrow \]

\[ \forall P \in \Psi(T): \quad \text{NP}(P(x, y)) \Leftarrow \]

Item (e) ensures that the domain and range of roles is correctly encoded, items (a1) and (a2) translate the ABox, items (s1) and (s2) the positive inclusions, items (n1) and (n2) the negative inclusions, and items (i1), (i2), and (ir) introduce the rules representing unsatisfiable concepts and unsatisfiable and irreflexive roles. Note, that $P^d$ contains the rule representation for both $O$ and $O^d$, which is why items (e)–(s2) contain doubled rules. Of course, if $O$ does not contain negative inclusion axioms, then we can skip all these, as well as items (n1)–(ir) which will not contribute anything anyway in this case. The additional default atoms are added to the doubled rules to be in line with the idea of the doubling of rules in [Alferes et al., 2013]: whenever, e.g., $A(x)$ is “classically false” for some $x$, i.e., $\text{NA}(A(x))$ holds, then we make sure that $A^d(x)$ is derivable as false for that same $x$ from the rules, but not necessarily $A(x)$, thus allowing to detect potential $\text{MKKNF}$-inconsistencies. That is also the reason why neither (i1)–(ir) nor the contrapositives in (s1) and (s2) do produce the doubled counterparts: atoms based on predicates of the forms $N^dP$ or $R^dP$ are not used anywhere. Finally, the doubled rules in (e) do not contain the default negated atom as this case only associates domain and range to a role assertion, either present in the ABox or derived elsewhere. Additionally, predicates $N^dP$ or $R^dP$ are not used anywhere, so such default negated atoms would be of no impact.

We can establish three correspondences between entailment from satisfiable $O$ and the program resulting from the translation $P^d$. First, we consider positive atoms.

Lemma 1. Let $O$ be a $DL$-Lite$_R$ ontology, $A$ a unary and $R$ a binary predicate:

- $O \models A(a)$ if\( P^d \models A(a)$ and $O \models R(a, b)$ if\( P^d \models R(a, b)$.

A similar property holds for (classically) atoms.

Lemma 2. Let $O$ be a $DL$-Lite$_R$ ontology, $A$ a unary and $R$ a binary predicate:

- $O \models \neg A(a)$ if $P^d \models NA(a)$ and $O \models \neg R(a, b)$ if $P^d \models NR(a, b)$.

We can also show the correspondent to Lemma 1 for the doubled predicates.

Lemma 3. Let $O$ be a $DL$-Lite$_R$ ontology, $A$ a unary and $R$ a binary predicate:

- $O^d \models A^d(a)$ if $P^d \models A^d(a)$ and $O^d \models R^d(a, b)$ if $P^d \models R^d(a, b)$.

Thus, we can define a correct partial oracle based on $P^d$.

Theorem 4. Let $K^d = (O, O^d, P^d)$ be a doubled $\text{MKKNF}$ KB and $pT^d_I\cup\ell\cup\ell$ a partial $\text{QL}$ oracle such that $pT^d_I(T, S, L) \iff P^d_O \cup T \cup \ell \models S$. Then $pT^d_I \cup L$ is a correct partial oracle w.r.t. $\text{comp}O$.

Instead of coupling two rule reasoners that interact with each other using an oracle, we can simplify the process altogether and integrate both into one rule reasoner. The resulting approach is decidable with polynomial data complexity.

Theorem 5. Let $K = (O, P)$ be an $\text{MKKNF}$ KB with $O$ in $DL$-Lite$_R$. An $\text{SLGi}(O)$ evaluation of a query in $\text{QL} = (\{0, P^d \cup P^d\})$ is decidable with data complexity in $\text{PTime}$.

4 System Description

In this section, we briefly describe the changes to the architecture of our plug-in and discuss some optimizations implemented w.r.t. the translation described in Sec. 3.

To allow the usage of OWL QL ontologies, changes were essentially made in the translator. First, since now two OWL profiles are supported we have introduced a switch that checks the profile of the loaded/edited ontology. If it is in OWL EL, then NoHR behaves as described in [Ivanov et al., 2013], i.e., the reasoner ELK is used to classify the ontology and return the inferred axioms to translator, which are then translated. Otherwise, we treat $O$ of the hybrid KB based on the translation described in Sec. 3 for OWL QL.

Notably, in Sec. 3, we only considered $DL$-Lite$_R$ while OWL QL includes a number of additional constructs which often can be expressed in $DL$-Lite$_R$. To account for that, we first normalize such expressions to axioms in $DL$-Lite$_R$.

This includes ignoring certain expressions, most of which do not contribute anything to derivations, e.g., $\text{SubClassOf}(\text{owl:Thing})$, while others make the ontology unsatisfiable, such as $\text{ClassAssertion/owl:Nothing a}$, although, as mentioned before, with no effect when querying the translated rules. The details on the normalization can be found in the appendix of the extended paper.
Subsequently, the graph is constructed, for determining unsatisfiable concepts and unsatisfiable and irreflexive roles, after which the translation is performed, which includes a number of optimizations. First, whenever there are no negative inclusions, the doubled rules are omitted in the cases (e)–(s2) of Def. 7. Additionally, case (e) is limited to those rules whose heads appear in the body of another rule. Both steps reduce the overall number of rules created during the translation.

The second group of optimizations is related to tabling in XSB, which contributes to help answering queries very efficiently in a top-down manner, and avoid infinite loops while querying. However, simply declaring all predicates to be tabled is very memory-consuming, so we reduced the number of tabled predicates without affecting loop detection. For example, only predicates that appear in any rule head and in any rule body need to be tabled. In addition, rules with an empty body (facts) can be ignored in the previous criterion, as these will not cause an infinite loop.

5 Evaluation

In this section, we evaluate our system and show that a) preprocessing is even faster when compared to NoHRs EL version, which was already capable of preprocessing large ontologies in a short period of time, b) querying scales well, even for over a million facts/assertions in the ABox, and c) adding rules scales linearly for pre-processing and querying, even for an ontology with many negative inclusions.

Tests were performed on a Notebook running Linux 3.17.6-1-ARCH (x86_64) with 1.8 GHz 4x Intel Core i3 processor and 4 GB of RAM. We used XSB 3.4.0 for querying, ran all tests in a terminal version and Java with "-XX:+AggressiveHeap" option, and report averages over 5 runs.

First, we considered LUBM8 [Guo et al., 2005], a standard benchmark for evaluating queries over a large data set. The ontology itself is already rather simple and we reduced it even a bit further by removing all axioms that are not common to both OWL 2 QL and EL. The resulting ontology has only ninety logical axioms, but this way we can use it with both translators included in NoHR and compare their performance. We created instances of LUBM 1–10 with assertions ranging from roughly 100,000 to over 1,300,000 and performed preprocessing from loading the ontology to loading the translation result in XSB. The results for both translators EL and QL can be found in Fig. 2. Note that the segment “Initialization” is the time for preparing the translation, which for EL includes classifying the ontology, while the larger part of the segment “Other” corresponds to loading the ontology.

We can observe that QL is considerably faster, indeed up to 40s for LUBM10, to a considerable extent due to avoiding classification. Besides that, the preprocessing time increases linearly, and the overall time for preprocessing is acceptable in our opinion as this is only done once before querying.

Next, we also queried the resulting ten rule sets in XSB for both EL and QL using queries from the LUBM benchmark, that were manually transformed from SPARQL to queries usable in XSB. Among the fourteen provided queries, we chose seven, because the others were either no longer meaningful due to removal of certain axioms/DL constructors during the initial simplifications we applied to LUBM, or because initial tests revealed that XSB would run out of memory for a query, simply because, for our test system with 4GB memory, too much data was being gathered in the tables to answer the query. In more detail, we used the queries 1, 2, 3, 4, 5, 7, 10 from the LUBM benchmark. The results are shown for some representatives in Fig. 3. Basically, for queries 1, 3, 4, and 10, no real difference between EL and QL exists and the response time is strictly below 1s. For query 7, there exists a slight difference in favor of EL with 2.5s vs. 1s for LUBM10, whereas for queries 2 and 5 the difference increases. In all cases, the response time grows linearly w.r.t. the increasing size of LUBM, and we can conclude that on average querying in QL is slightly slower. Here, EL compensates for the longer preprocessing, and this effect becomes more visible, the more complex the query is and the more data needs to be gathered to answer it. Intuitively, this can be explained by looking at a simple example with two axioms $A \sqsubseteq B$ and $\exists R \sqsubseteq B$ For EL, classification, yields $A \sqsubseteq B$ and only one axiom is translated and only one derivation step is required in XSB to obtain, say $B(a)$ from $A(a)$. For QL, both axioms are translated directly without classification, using $DR$, but now, two derivation steps would be required in XSB to obtain $B(a)$ from $A(a)$. It thus seems that deciding which of the two forms of translations performs better depends on the kind (and number) of queries we pose.

Finally, with the aim of also testing a more expressive
OWL 2 QL ontology, we used the LIPID ontology,\(^9\) which has, besides 749 subclass axioms, 1,486 class disjointness axioms and 20 inverse object properties in combination with non-monicontic rules. The latter were created by means of the rule generator already used in [Ivanov et al., 2013] with a ratio 1:10 between rules and facts, also introducing some new predicates not present in the ontology itself. We performed the preprocessing step and observed only slight effects due to the increasing amount of rules. The time for processing the ontology was naturally stable for all steps, and overall processing time was between 2 and 3s. Notably, the considerable amount of negative inclusions had no significant impact on time, e.g., when constructing the graph. Then, we posed three simple queries (Query1−3), namely Acyl_Ester, Chain(X), Lipid(X), and Entity(X) to the resulting rule sets in XSB. The results are shown in Fig. 4. As we can see, the response time is still very reasonable, from clearly below 1s to up to 8s. Still, the results in our opinion already show the effect of the arbitrary rules that tend to introduce links between predicates that increase the search space. This can be noted in particular for Query1, where in one case a smaller set of rules results in a higher response time, simply because no generated rule set is a subset of another. We note that performance tests of querying (non-monicontic) rules and ontologies would considerably benefit from real datasets but to the best of our knowledge currently none are available.

6 Conclusions

We have extended NoHR, the Protégé plug-in that allows to query non-monicontic rules and ontologies in OWL 2 EL, to also admit ontologies in OWL 2 QL. While the principal architecture of the tool remains the same, the crucial module that translates the ontology into rules with the help of a classifier simply cannot be re-used, which is why we introduced a novel direct translation for OWL 2 QL ontologies to cover this profile. We have implemented this translation and discussed optimizations. The evaluation shows that it maintains all positive evaluation results of the OWL 2 EL version [Ivanov et al., 2013], and is even faster during pre-processing, as no classification is necessary, in exchange for an on average slightly longer response time during querying.

Besides the OWL 2 EL profile supported by NoHR, and compared to in Sect. 5, also [Gomes et al., 2010; Knorr and Alferes, 2011] both build on the well-founded MKNF semantics [Knorr et al., 2011]. While [Gomes et al., 2010] uses the non-standard CDF framework integrated in XSB, which complicates compatibility to standard OWL tools based on the OWL API, [Knorr and Alferes, 2011] presents an OWL 2 QL oracle based on common rewritings in the underlying DL DL-Lite\(_\text{R}\) [Artale et al., 2009], but would require constant interaction between a rule reasoner and a DL reasoner, which is why we believe it to be less efficient than our approach.

Two related tools are DReW [Xiao et al., 2013] and HD Rules [Drabent et al., 2007], although based on different underlying formalisms to combine ontologies and rules (c.f. [Eiter et al., 2008; Motik and Rosati, 2010] for a comparison), which, again, considerably complicates comparison.

Future work includes the extension to OWL 2 RL, but developing an alternative for OWL 2 QL using the classifier integrated in ontio [Kontchakov et al., 2014] once its OWL API becomes available. XSB, the general reasoner Konclude [Steiglmiller et al., 2014], could shed more light on whether classification or direct translation fares better for proper OWL 2 QL ontologies. The efficiency of the latter reasoner also motivates looking into non-polynomial DLs, with possible influences from recent work on rewriting disjunctive datalog programs [Kaminski et al., 2014]. Finally, we may extend NoHR for OWL 2 QL (and EL) to the paraconsistent semantics [Kaminski et al., 2015] that would provide true support to the already occasionally observed paraconsistent behavior, or alternatively, to either generalizations of hybrid KBs [Gonçalves and Alferes, 2010; Knorr et al., 2014; 2012; Knorr, 2015] or dynamics in hybrid KBs [Slota et al., 2011; Slota and Leite, 2012], or even both [Gonçalves et al., 2014].

References


---

\(^9\)http://bioonto.dcs.aber.ac.uk/ql-ont/
Reasoning with Forest Logic Programs Using Fully Enriched Automata

Cristina Feier
Department of Computer Science
University of Oxford
cristina.feier@cs.ox.ac.uk

Thomas Eiter
Institute of Information Systems
University of Technology
eiter@kr.tuwien.ac.at

Abstract
Forest Logic Programs (FoLP) are a decidable fragment of Open Answer Set Programming (OASP) which have the forest model property. OASP extends Answer Set Programming (ASP) with open domains—a feature which makes it possible for FoLPs to simulate reasoning with the expressive description logic SHOQ. At the same time, the fragment retains the attractive rule syntax and the non-monotonicity specific to ASP. In the past, several tableau algorithms have been devised to reason with FoLPs, the most recent of which established a NEXPTIME upper bound for reasoning with the fragment. While known to be EXPTIME-hard, the exact complexity characterization of reasoning with the fragment was still unknown. In this paper we settle this open question by a reduction of reasoning with FoLPs to emptiness checking of fully enriched automata, a form of automata which run on forests, and which are known to be EXPTIME-complete.

Introduction
Open Answer Set Programming (OASP) (Heymans, Van Nieuwenborgh, and Vermeir 2008) extends Answer Set Programming (ASP) (Gelfond and Lifschitz 1988) with an open domain semantics: programs are interpreted w.r.t. arbitrary domains that might contain individuals which do not occur explicitly in the program. This makes it possible to state generic knowledge using OASP. At the same time, OASP inherits from ASP the negation under the stable model semantics. Thus, ASP bridges two important knowledge representation paradigms: the classical First Order Logic (FOL) open world and the non-monotonic rules closed world. It is part of a broad line of research which includes approaches like DL-safe rules (Motik, Sattler, and Studer 2005), DL+log (Rosati 2006), dl-programs (Eiter et al. 2008), Description Logic Rules (Krötzsch, Rudolph, and Hitzler 2008), r-hybrid knowledge bases (Rosati 2008), Datalog+ (Cali, Gottlob, and Lukasiewicz 2009), MKNF+ knowledge bases (Motik and Rosati 2010), and Nonmonotonic Existential Rules (Magka, Krötzsch, and Horrocks 2013).

In general, OASP is undecidable. To achieve decidability, several fragments have been defined by imposing syntactical restrictions on the shape of rules. Such a fragment are Forest Logic Programs (FoLP) which enjoy the forest model property: a unary predicate is satisfiable iff it is satisfied by a model that can be represented as a labeled forest, where nodes and arcs are labeled with sets of unary predicates and binary predicates, respectively.

FoLPs are quite an expressive fragment as they allow, for instance, the simulation of standard reasoning tasks (like concept satisfiability and KB consistency) with SHOQ ontologies (Feier and Heymans 2013). This property of FoLPs led to f-hybrid KBs: a combination of rules and ontologies which distinguish themselves among other approaches like dl-safe rules, r-hybrid knowledge bases, or MKNF+ knowledge bases, by the fact that they impose no restrictions on the interaction between the signatures of the two components. Such restrictions prevent the need for reasoning with unknown individuals in the rule component. As f-hybrid KBs are based on the simulation of SHOQ KBs within FoLPs, no such restriction is needed. Conceptual modeling using FoLPs is not restricted to simulating reasoning with SHOQ KBs: it is also possible to translate object-role modeling (ORM) models as sets of FoLP rules, e.g. (Heymans 2006).

As they can simulate reasoning within the Description Logic (DL) SHOQ, it follows that reasoning with FoLPs is EXPTIME-hard. However, the exact complexity characterization of FoLPs was still open. Previously, reasoning with FoLPs was addressed by means of tableau-based algorithms: (Feier and Heymans 2013) described a 2NEXPTIME tableau algorithm, while an improved algorithm which runs in the worst case in NEXPTIME has been described in (Feier 2012). While in the latter work it has been speculated that the non-deterministic tableau algorithm can be determined in order to lead to an EXPTIME procedure which would be worst-case optimal, the determinization in the case of FoLPs proved elusive. Such a deterministic worst-case optimal algorithm has been devised for CoLPs, which restrict FoLPs to programs without constants, and simple FoLPs, a fragment in which recursion is restricted: the technique does not scale up to FoLPs (see (Feier 2014)).

In this paper, we settle the open question regarding the exact complexity characterization of FoLPs: by using a reduction to emptiness checking of Fully Enriched Automata
we show that satisfiability checking of unary predicates w.r.t. FOLPs is EXPTIME-complete. Hence, reasoning with FoLP rules and SHOQ ontologies is not harder than reasoning with SHOQ ontologies themselves.

Fully enriched automata have been introduced in (Bonatti et al. 2008) as a tool to reason with hybrid graded μ-calculus, which extends μ-calculus with graded modalities and nominals. They offer an elegant device for our encoding as they accept forests as inputs and also feature a parity acceptance condition that is useful in distinguishing well-supported models (Fages 1991), a fundamental characteristic of (open) answer sets. However, FoLPs exhibit a specific form of the forest model property, in which every node can point back to any root of the forest, and as such the encoding is not without its challenges.

The automata-based method has been previously applied to reason with CoLPs (Heymans, Van Nieuwenborgh, and Vermeir 2006); satisfiability checking of unary predicates w.r.t. a CoLP has been reduced to non-emptyness checking of two-way alternating tree automata (2ATA) (Vardi 1998). 2ATAs have also been used to check consistency of normal bidirectional ASP programs (bd-programms) (Eiter and Simkus 2009), which are a decidable fragment of ASP extended with function symbols that also exhibit the tree model property. In the context of DL, 2ATAs have been employed to check concept satisfiability (Calvanese, Giacomo, and Lenzerini 2002) and satisfiability of ALCQI KBs (Calvanese, Eiter, and Ortiz 2007)–in the latter case, canonical models are forest-shaped, and as such they were encoded as trees in order to be processed using 2ATAs. In the case of FoLPs, it is not clear how such an encoding would work due to the special form of their forest model property. Finally, FEAs were used to encode satisfiability checking of ZOTIQ concepts (Calvanese, Eiter, and Ortiz 2009).

**Preliminaries**

We start by introducing the open answer set syntax and semantics (Heymans, Van Nieuwenborgh, and Vermeir 2008). We assume countably infinite disjoint sets of constants, variables, and predicate symbols. Terms and atoms are defined as usual. We refer to an atom where the predicate symbol is unary or binary, as a unary or binary atom, resp. A literal is an atom or a negated atom not a. We allow for inequality literals of the form s ≠ t, where s and t are terms. A literal that is not an inequality literal will be called a regular literal.

For a set S of literals or (possibly negated) predicates, \( S^+ = \{ a \mid a \in S \} \) and \( S^- = \{ a \mid \text{not } a \in S \} \). If S is a set of (possibly negated) predicates of arity n and \( t_1, \ldots, t_n \) are terms, then \( S(t_1, \ldots, t_n) = \{ l(t_1, \ldots, t_n) \mid l \in S \} \). For a set S of atoms, \( not S = \{ not a \mid a \in S \} \).

A program is a finite set of rules \( r : \alpha \leftarrow \beta \), where \( \alpha \) is a finite set of regular literals and \( \beta \) is a finite set of literals. We denote as head(r)/body(r) the set of α/β, where α/β stands for a disjunction/conjunction.

Atoms, literals, rules, and programs that do not contain variables are ground. For a rule or a program \( R \), let vars(R), preds(R), and cts(R) be the sets of variables, predicates, and constants that occur in \( R \), resp. A universe \( U \) for \( P \) is a non-empty countable superset of the constants in \( P; U \supseteq cts(P) \). We call \( P \) the program obtained from \( P \) by substituting every variable in \( P \) by every element in \( U \). Let \( B_P \) be the set of regular atoms that can be formed from a ground program \( P \).

An interpretation \( I \) of a ground program \( P \) is a subset of \( B_P \). We write \( I \models p(t_1, \ldots, t_n) \) if \( p(t_1, \ldots, t_n) \in I \) and \( I \models not p(t_1, \ldots, t_n) \) if \( I \not\models p(t_1, \ldots, t_n) \). Also, for ground terms \( s, t \), we write \( I \models s \neq t \) if \( s \neq t \). For a set of ground literals \( L, I \models l \) for every \( l \in L \). A ground rule \( r : \alpha \leftarrow \beta \) is satisfied w.r.t. \( I \), denoted \( I \models r \), if \( I \models l \) for some \( l \in \alpha \) whenever \( I \models \beta \). A ground constraint \( \alpha \leftarrow \beta \) is satisfied w.r.t. \( I \) if \( I \not\models \beta \).

For a positive ground program \( P \), i.e., a program without not, an interpretation \( I \) of \( P \) is a model of \( P \) if \( I \) satisfies every rule in \( P \); it is an answer set of \( P \) if it is a minimal model of \( P \). When \( P \) is definite (does not contain disjunction) the minimal model of \( P \) can be computed using the well-known \( TP \) operator: for a set of atoms \( B \), let \( TP(B) = B \cup \{ a \mid a \leftarrow \beta \in P \land \beta \models \beta \} \). Then, let \( TP^0(B) = B \) and \( TP^{i+1}(B) = TP(TP_i(B)) \); the minimal model (answer set) of \( P, M(P) \), is defined as \( \bigcup_{i=0}^{\infty} TP_i(\emptyset) \). The derivation level of an atom \( a \in M(P) \), level(\( a, M(P) \) ), is the least integer \( k \) such that \( a \in TP_k(\emptyset) \). For ground programs \( P \) containing not, the GL-reduct (Gelfond and Lifschitz 1988) w.r.t. \( I \) is defined as \( P^I \), where \( P^I \) contains \( \alpha \leftarrow \beta^+ \) for every \( \beta \leftarrow \beta^+ \in P, I \models \beta^+ \) and \( I \models \alpha \). \( I \) is an answer set of a ground \( P \) if \( I \) is an answer set of \( P^I \).

An open interpretation of a program \( P \) is a pair \( (U, M) \) where \( U \) is a universe for \( P \) and \( M \) is an interpretation of \( P_U \). An open answer set of \( P \) is an open interpretation \( (U, M) \) of \( P \), with \( M \) an answer set of \( P_U \). For every atom \( a \in M \), where \( (U, M) \) is an open answer set of \( P \), level(\( a, M(P^M_U) = M \)) is finite (Heymans 2006).

**Trees and forests:** We introduce notation for trees and forests which extend those in (Vardi 1998). Let · be a concatenation operator between sequences of constants or natural numbers, \( \mathbb{N}^+ \) be the set of positive integers, and \( \mathbb{N}^* \) be the set of all sequences of positive integers formed using the concatenation operator. We denote with \( \in \) the empty sequence: for every constant or natural number \( c, c \cdot c = c \). A tree \( T \) with root \( c \), also denoted as \( T_c \), is a set of nodes, where each node is a sequence of the form \( c \cdot s \), where \( s \in \mathbb{N}^* \), and for every \( x \cdot d \in T_c, d \in \mathbb{N}^+ \), it must be the case that \( x \in T_c \). When the root of the tree is irrelevant, we will simply refer to the tree as \( T \).

Given a tree \( T \), its set of arcs is \( A_T = \{ (x, y) \mid x, y \in T, \exists n \in \mathbb{N}^+, y = x \cdot n \} \). We denote with \( succ_c(x) = \{ y \mid y = x \cdot i, i \in \mathbb{N}^+ \} \) the successors of a node \( x \) in \( T \). For a node \( y = x \cdot i \in T, prec_c(y) = x \).

A forest \( F \) is a set of trees \( \{ T_c \mid c \in C \} \), where \( C \) is a finite set of arbitrary constants. The set of nodes, \( N_F \), and the set of arcs, \( A_F \), of a forest \( F \) are defined as: \( N_F = \bigcup_{T \in F} T \), and \( A_F = \bigcup_{T \in F} A_T \), resp. For a node \( x \in N_F \), let \( succ_c(x) = succ_c(x), \) where \( x \in T \) and \( T \in F \). For a node \( y = x \cdot i \in T \) and \( T \in F, prec_c(y) = prec_c(y) = x \).

An interconnected forest \( EF \) is a tuple \( (F, ES) \), where \( F = \{ T_c \mid c \in C \} \) is a forest and \( ES \subseteq N_F \times C \). The sets of nodes \( N_{EF} \) and arcs \( A_{EF} \) of an interconnected forest \( EF \)
are defined as: \( N_{EF} = N_F \), and \( A_{EF} = A_F \cup ES \), resp.

A \( \Sigma \)-labelled forest is a tuple \((F,f)\) where \( F \) is an interconnected forest/tree and \( f : N_F \rightarrow \Sigma \) is a labelling function, with \( \Sigma \) being a set of arbitrary symbols.

**Forest Logic Programs**

Forest Logic Programs (FoLPs) are a fragment of OASP which have the forest model property. They allow only for unary and binary predicates and tree-shaped rules. The tree-like structure of rules refers to the chaining pattern of rule variables: one variable can be seen as the root of a tree and the others as descendants such that for every arc in the tree, there is a positive binary literal in the body which connects the two corresponding variables. Inequalities between ‘successor’ variables can also appear in the body of such a rule: we will refer to the set of literals in the body of a rule formed only with the ‘root’ variable as the ‘local part’ and to the remaining part as the ‘successor part’. FoLPs allow also for so-called ‘free’ rules, which are rules of the form: \( p(t) \lor \neg p(t) \leftarrow \), where \( p \) is a unary/binary predicate and \( t \) is a unary/binary tuple of terms.

**Definition 1** A forest logic program (FoLP) is an open answer set program with only unary and binary predicates, and s. t. a rule is either:

- a free rule:
  \[
  a(s) \lor \neg a(s) \leftarrow ,
  \]
  \( (1) \)

- or
  \[
  f(s,t) \lor \neg f(s,t) \leftarrow
  \]
  \( (2) \)

- a unary rule:
  \[
  a(s) \leftarrow \beta(s),(\gamma_i(s,t_i),\delta_i(t_i))_{1 \leq i \leq m},\psi
  \]
  \( (3) \)

- or a binary rule:
  \[
  f(s,t) \leftarrow \beta(s),\gamma(s,t),\delta(t),
  \]
  \( (4) \)

where in each rule above:

- \( a \) is a unary predicate, and \( f \) is a binary predicate,
- \( s, t, \) and \( (t_i)_{1 \leq i \leq m} \) are distinct terms,
- \( \beta, \delta, \) and \( (\gamma_i)_{1 \leq i \leq m} \) are sets of (possibly negated) unary predicates,
- \( \gamma \) and \( (\gamma_i)_{1 \leq i \leq m} \) are sets of (possibly negated) binary predicates,
- inequality does not appear in any \( \gamma \): \( \gamma \neq \emptyset \), for \( 1 \leq m \leq k \), and \( \emptyset \neq \gamma \);
- there is a positive atom that connects the head term \( s \) with any successor term which is a variable: \( \gamma^+ \neq \emptyset \), if \( t_i \) is a variable, for every \( 1 \leq i \leq m \), and \( \gamma^+ \neq \emptyset \), if \( t \) is a variable.

A predicate \( q \) is free if it occurs in a free rule in \( P \).

**Example 1** The following program \( P \) is a FoLP: it describes the fact that somebody who has read two different novels is a literature lover (unary rule \( r_1 \)), a novel is something written by a novelist (unary rule \( r_2 \)), and a novelist is somebody who has written at least one novel (unary rule \( r_3 \)). There are three free rules describing binary predicates \( \text{read}, \text{writtenBy}, \) and \( \text{wrote as being free}. \) Finally, there are two facts (unary rules with empty bodies): \( f_1 \) and \( f_2 \).

\[
\begin{align*}
  r_1 & : \text{Lit Lover}(X) \leftarrow \text{read}(X,Y_1), \text{read}(X,Y_2), \text{Novel}(Y_1), \text{Novel}(Y_2), Y_1 \neq Y_2 \\
  r_2 & : \text{Novel}(X) \leftarrow \text{writtenBy}(X,Y), \text{Novelist}(Y) \\
  r_3 & : \text{Novelist}(X) \leftarrow \text{wrote}(X,Y), \text{Novel}(Y) \\
  r_4 & : \text{read}(X,Y) \lor \neg \text{read}(X,Y) \leftarrow \\
  r_5 & : \text{writtenBy}(X,Y) \lor \neg \text{writtenBy}(X,Y) \leftarrow \\
  r_6 & : \text{wrote}(X,Y) \lor \neg \text{wrote}(X,Y) \leftarrow \\
  f_1 & : \text{Novel}(a) \leftarrow \\
  f_2 & : \text{Novelist}(b) \leftarrow
\end{align*}
\]

For a FoLP \( P \), we denote with \( \text{upr}(P), \text{bpr}(P), \text{urul}(P), \) and \( \text{brul}(P) \), the sets of unary and binary predicates and unary and binary rules which occur in \( P \), resp.

For a unary rule \( r \) of type (3), the degree of \( r \), denoted as \( \text{deg}(r) \), is the number \( k \) of successor variables which appear in the rule. Intuitively, it indicates the maximum number of successor individuals needed to make the body of the rule true. The degree of a free rule is 0. For a unary predicate \( p \): \( \text{deg}(p) = \max\{\text{deg}(r) \mid p \in \text{preds}(\text{head}(r))\} \). Finally, the degree of a FoLP \( P \) is \( \text{deg}(P) = \sum_{p \in \text{upr}(P)} \text{deg}(p) \).

It over-approximates the number of successor individuals needed to satisfy all atoms of the form \( p(x) \), where \( p \in \text{upr}(P) \), for a given individual \( x \).

**Example 2** Let \( P \) be the FoLP from Example 1. Then, \( \text{deg}(\text{Lit Lover}) = 2, \text{deg}(\text{Novel}) = 1, \text{deg}(\text{Novelist}) = 1, \) and thus, \( \text{deg}(P) = 4. \)

**Forest models:** The forest model property of an OASP \( P \) is as follows: if an unary predicate \( p \) is satisfiable, then there exists a model which satisfies \( p \) that can be seen as an interconnected forest. The forest contains for each constant in \( P \) a tree having the constant as root, and possibly an additional tree with an anonymous root. The predicate checked to be satisfiable, \( p \), belongs to the label of one of the root nodes.

**Definition 2** Let \( P \) be a program. A predicate \( p \in \text{upr}(P) \) is forest satisfiable w.r.t. \( P \) if there exist an open answer set \( (U,M) \) of \( P \); an interconnected forest \( EF = (\{T_a\} \cup \{T_u \mid a \in \text{cts}(P)\}, ES) \), where \( P \) is a constant, possibly from \( \text{cts}(P) \); and a labelling function \( \text{ef} : \{T_a\} \cup \{T_u \mid a \in \text{cts}(P)\} \cup A_{EF} \rightarrow 2^{\text{preds}(P)} \) such that:

- \( p \in \text{ef}(p) \),
- \( U = N_{EF} \),
- \( \text{ef}(x) \in 2^{\text{upr}(P)} \), when \( x \in T_p \cup \{T_u \mid a \in \text{cts}(P)\} \),
- \( \text{ef}(x) \in 2^{\text{bpr}(P)} \), when \( x \in A T_p \),
- \( M = \{p(x) \mid p \in \text{ef}(x), x \in N_{EF} \cup \{f(x,y) \mid f \in \text{ef}(x,y), (x,y) \in A_{EF}\}\} \), and
- for every \( (z,z \cdot i) \in A_{EF} \): \( \text{ef}(z,z \cdot i) \neq \emptyset \).

We call such a pair \((U,M)\) a forest model. A program \( P \) has the forest model property, if every unary predicate \( p \) that is satisfiable w.r.t. \( P \), is forest satisfiable w.r.t. \( P \).

**Proposition 1** FoLPs have the forest model property.
They accept forests as input. We describe them following (Bonatti et al. 2008).

For a set $Y$, we denote with $B^+(Y)$ the set of positive Boolean formulas over $Y$, where true and false are also allowed and where $\land$ has precedence over $\lor$. For a set $X \subseteq Y$ and a formula $\theta \in B^+(Y)$, we say that $X$ satisfies $\theta$ iff assigning true to elements in $X$ and assigning false to elements in $Y \setminus X$ makes $\theta$ true. For $b > 0$, let $D_b = \{\emptyset, \{0\}, \ldots, \{b\}\} \cup \{\{0\}, [1], \ldots, \{b\}\} \cup \{-1, e\}$. 

A fully enriched automaton (FEA) is a tuple $A = (\Sigma, b, Q, \delta, q_0, F)$, where $\Sigma$ is a finite input alphabet, $b > 0$ is a counting bound, $Q$ is a finite set of states, $\delta : Q \times \Sigma \to B^+(D_0 \times Q)$ is a transition function, $q_0 \in Q$ is an initial state, and $F = \{F_1, F_2, \ldots, F_k\}$, where $F_1 \subseteq F_2 \subseteq \ldots \subseteq F_k = Q$ is a parity acceptance condition. The number $k$ of sets in $F$ is the index of the automaton.

A run of a FEA on a labeled forest $(F, V)$ is an $N_F \times Q$-labeled tree $(T_c, r)$ such that:

- $r(c) = (d, q_0)$, for some root $d$ in $F$, and
- for all $y \in T_c$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \emptyset$, there is a (possibly empty) set $S \subseteq D_b \times Q$ such that $S$ satisfies $\theta$ and for all $(d, s) \in S$, the following hold:
  - if $d = \{-1, e\}$, then $x \cdot d$ is defined and there is $j \in \mathbb{N}^+$ such that $y \cdot j \in T_c$ and $r(y \cdot j) = (x \cdot d, s)$;
  - if $d = \langle n \rangle$, then there is a set $M \subseteq \text{succ}(x)$ of cardinality $n + 1$ such that for all $z \in M$, there is $j \in \mathbb{N}^+$ such that $y \cdot j \in T_c$ and $r(y \cdot j) = (z, s)$;
  - if $d = \langle n \rangle$, then there is a set $M \subseteq \text{succ}(x)$ of cardinality $n$ such that for all $z \in \text{succ}(x) - M$, there is $j \in \mathbb{N}^+$ such that $y \cdot j \in T_c$ and $r(y \cdot j) = (z, s)$;

If $\theta$ above is true, then $y$ does not need to have successors. Moreover, since no set $S$ satisfies $\theta = \emptyset$, there cannot be any run that takes a transition with $\theta = \emptyset$. A run is accepting if each of its infinite paths $\pi$ is accepting, that is if the minimum $i$ for which $\text{Inf}(\pi) \cap F_i \neq \emptyset$, where $\text{Inf}(\pi)$ is the set of states occurring infinitely often in $\pi$, is even. The automaton accepts a forest iff there exists an accepting run of the automaton on the forest. The language of $A$, denoted $L(A)$, is the set of forests accepted by $A$. We say that $A$ is non-empty if $L(A) \neq \emptyset$.

**Theorem 1** (Corollary 4.3 (Bonatti et al. 2008)) Given a FEA $A = (\Sigma, b, Q, \delta, q_0, F)$ with $n$ states and index $k$, deciding whether $L(A) = \emptyset$ is possible in time $(b + 2)^{O(n^2 k^2 \log k \log k)}$.

**From Forest Logic Programs to Fully Enriched Automata**

In this section we encode satisfiability checking of unary predicates with respect to FoLPs as emptiness checking for FEAS.

We start by introducing for every FoLP $P$ and unary predicate $p$ a class of FEAs $A^p_{\rho, \theta}$, where $\rho$ is one of $\text{cts}(P)$.
or a new anonymous individual and \( \theta : cts(P) \cup \{ \rho \} \rightarrow 2^{\text{upr}(P) \cup cts(P) \cup \{ \rho \}} \) is a function which has the following properties: \( \theta(\alpha) = \theta(\beta) \) for every \( \alpha, \beta \in cts(P) \cup \{ \rho \} \). Furthermore, \( \rho \in \theta(c) \), where \( c \) is one of \( cts(P) \cup \{ \rho \} \) and \( c \) is \( \rho \) if \( \rho \notin cts(P) \).

Intuitively, \( A_{\rho, \delta}^P \) will accept forest models of \( P \) with respect to \( P \) encoded in a certain fashion: in particular, for every root in the forest model, the root node will appear in its own label; function \( \theta \) fixes a content for the label of each root of accepted forest models.

In the following, let \( \delta = \delta \rho(q) \). We will also denote with \( \text{PAT}_\rho \) the set \( \{ * \} \cup cts(P) \) of term patterns, where \( * \) stands for a generic anonymous individual: we say that a term \( t \) matches a term pattern \( p \) and write \( t \models p \) iff \( t = pt \) when \( t \) is a constant. In other words, the match trivially holds. Term patterns will be used in our encoding as a unification mechanism between terms occurring in a FoLP (variables and constants) and elements in a universe (anonymous individuals and constants): a variable will match with a constant or an anonymous individual, but a constant will match only with itself. The automata \( A_{\rho, \delta}^P \) will run on forests labelled using the following alphabet: \( \Sigma = 2^\mathbb{S} \), where \( \mathbb{S} = \text{mpr}(P) \cup \{ 1, \ldots, d \} \cup cts(P) \cup \{ \rho \} \cup \{ \{ \rho \}, f \in \text{mpr}(P), t \in \text{PAT}_\rho \} \).

Unlike the case for forest models, the arcs of forests accepted by FEs are not labelled: as such, binary predicates occur in the label of nodes in an adorned form. These adorned predicates are either of the form \( \delta \rho \), in which case they represent a \( \delta \)-link between the predecessor of the labelled node, which has term pattern \( t \) and the node itself, or of the form \( \rho \), which can be considered to link the current node to a constant \( o \) from \( P \) via the binary predicate \( f \). Alternatively, unary predicates, labels might contain natural numbers and constants, which will be used as an addressing mechanism for successors of a given node and nodes which stand for constants in accepted forests, resp. The set of states of the automaton is as follows: \( Q = Q_0 \cup Q_+ \cup Q_\neg \), and with:

\[
Q_0 = \{ q_0, q_1, q_o, q_{-k} \},
\]

\[
Q_+ = \{ q_{t,a}, q_{t,a,r}, q_{t_1,t_2,a}, q_{t_1,t_2,r}, q_{t_1,t_2,u} \},
\]

\[
Q_\neg = \{ q_{t,a,r}, q_{t_1,t_2,a}, q_{t_1,t_2,r}, q_{t_1,t_2,u} \},
\]

where \( \rho \in cts(P) \cup \{ \rho \} \), \( \alpha \in \text{mpr}(P) \), \( f \in \text{mpr}(P) \), \( u \) is of the form \( a, f, \text{not } a \) or \( \text{not } f \), \( k \in \{ 1, \ldots, d \} \), \( r_a \in \text{urad}(P) \), \( r_f \in \text{brul}(P) \), and \( t, t_1, t_2 \in \text{PAT}_\rho \). We will refer to \( Q_+ \) and \( Q_\neg \) as the positive and the negative states of \( A_{\rho, \delta}^P \), resp. Intuitively, positive states are used to motivate the presence of atoms in an open answer set while negative states are used to motivate the absence of atoms in an open answer set. \( Q_0 \) contains some additional states, like \( q_0 \), the initial state, \( q_1 \), a state which will be visited recursively in every node of the forest, \( q_o \), a state corresponding to the initial visit of constant nodes, and \( q_{-k} \), a state which asserts that for every node in an accepted forest there must be at most one successor which has \( k \) in the label.

We describe in detail the transition function of \( A_{\rho, \delta}^P \). The initial transition prescribes that the automaton visits a root of the forest in state \( q_o \), for every \( \rho \in cts(P) \cup \{ \rho \} \):

\[
\delta(q_0, \sigma) = \bigwedge_{o \in cts(P) \cup \{ \rho \}} \left( \langle \text{root}, q_o \rangle \right) \tag{5}
\]

In every such state \( q_o \), it should hold that \( o \) and only \( o \) is part of the label. Furthermore, the automaton justifies the presence and absence of each unary predicate \( a \) and each adorned upward binary predicate in the label\(^1\) by entering states \( q_o,a, q_o,a,f, q_o,a,r, q_o,a,u \) and \( q_o,a,\neg q_o,a,f \), resp. At the same time every successor of the constant node is visited in state \( q_1 \). Let \( q'_o = cts(P) \cup \{ \rho \} - \{ o \} \). Then:

\[
\delta(q_o, \sigma) = o \in \sigma \land \bigwedge_{\alpha \in \rho} o' \notin \sigma \land \bigwedge_{a \in \text{urad}(P)} (\varepsilon, q_o,a) \land \bigwedge_{a \notin \text{urad}(P)} \left( \langle \varepsilon, q_o,a \rangle \land \right) \left( \langle \varepsilon, q_o,a,r \rangle \land \left( \langle [0], q_o \rangle \right) \right) \tag{6}
\]

Whenever the automaton finds itself in state \( q_1 \) it tries to motivate the presence and absence of each unary and adorned binary predicate in its label and then it recursively enters the state into each successor of the current node. It also ensures that for each integer \( 1 \leq k \leq d \), the labels of each but one successor do not contain \( k \):

\[
\delta(q_1, \sigma) = \langle [0], q_1 \rangle \land \bigwedge_{i \leq k \leq d} \left( \langle [i], \neg q_o \rangle \land \left( \langle \varepsilon, q_o,a \rangle \land \bigwedge_{a \in \text{urad}(P)} \left( \langle \varepsilon, q_o,a \rangle \land \left( \langle \varepsilon, q_o,a,r \rangle \land \left( \langle \varepsilon, q_o,a,u \rangle \land \right) \right) \right) \right) \tag{7}
\]

\[
\delta(q_{-k}, \sigma) = k \notin \sigma \tag{8}
\]

To motivate the presence of a unary/binary predicate in the label of a node, the automaton checks whether the given predicate is free (we assume that \( \text{free}(a) \) evaluates to true if \( a \) is free and, to false, otherwise) or finds a unary/binary rule which supports the predicate. Note the distinction concerning the term pattern for the node where the predicate has to hold. In the case of unary predicates, if the node is a constant, there is first a preliminary check that we are at the right node - this is needed as later the automaton will visit all roots in this state. In the case of binary predicates, depending on the term pattern, the label is checked for different types of adorned binary atoms. In all cases, only rules whose head terms match the given term patterns are chosen to motivate the presence of predicates in the label.

\[
\delta(q_{t,a}, \sigma) = a \in \sigma \land \left( \varepsilon, q_o,a \right) \right) \tag{9}
\]

\[
\delta(q_{t,a,r}, \sigma) = a \notin \sigma \lor a \in \text{not}(\sigma) \land \left( \varepsilon, q_o,a,\neg \right) \tag{10}
\]

\[
\delta(q_{t_1,t_2,a}, \sigma) = \langle \varepsilon, q_o,a \rangle \land \left( \langle \varepsilon, q_o,a,\neg \rangle \right) \tag{11}
\]

\(^1\)As constants have no predecessors in the forest, there will be no adorned downward predicates in the label.
\[ \delta(q_{t,o,r}, \sigma) = \bigwedge_{u \in \beta} \bigwedge_{u \in \gamma \cup \delta} (\varepsilon, q_{t,u}) \] (22)

In the following, we describe the transitions of the automaton in the negative states, i.e. the states which are used to motivate the absence of certain unary/binary predicates in the labels of the forest. If one ignores the checks for the absence of unary/adorned binary predicates in the label of the current node, the transition rules can be seen as dual versions of the ones for the counterpart positive states.

\[ \delta(q_{\neg r}, \sigma) = a \notin \sigma \land \bigwedge_{r_a \in \alpha(s) \rightarrow \beta} (\varepsilon, q_{\neg r}) \] (23)

\[ \delta(q_{\neg r}, \sigma) = a \notin \sigma \land \bigwedge_{r_a \in \theta(a) \rightarrow \beta} (\varepsilon, q_{\neg r}) \] (24)

\[ \delta(q_{\neg r}, \sigma) = \bigwedge_{r_f \in \delta(v)} (\varepsilon, q_{\neg r}) \] (25)

\[ \delta(q_{\neg r}, \sigma) = \bigwedge_{r_f \in \delta(v)} (\varepsilon, q_{\neg r}) \] (26)

\[ \delta(q_{\neg r}, \sigma) = \bigwedge_{r_f \in \delta(v)} (\varepsilon, q_{\neg r}) \] (27)

\[ \delta(q_{\neg r}, \sigma) = k \notin \sigma \land \bigwedge_{j \neq k} (\varepsilon, q_{\neg r}) \] (28)

For binary rules: \( r_f : f(s, v) \leftarrow \beta(s), \gamma(s, v), \delta(v), \) when \( v \) is grounded using an anonymous individual, the check involves looking to the predecessor node to see if the local part of the rule is satisfied. When \( v \) is grounded using a constant, the local part of the rule is checked at the current node and the successor part at the respective constant. Note that the first term pattern in the first conjunct in both rules (21) and (22) is irrelevant as \( \beta \) contains only unary predicates.

\[ \delta(q_{t,s}, \sigma) = \bigwedge_{u \in \beta} (\varepsilon, q_{t,u}) \land \bigwedge_{u \in \gamma \cup \delta} (\varepsilon, q_{t,s}) \] (21)

For binary rules: \( r_f : f(s, v) \leftarrow \beta(s), \gamma(s, v), \delta(v), \) when \( v \) is grounded using an anonymous individual, the check involves looking to the predecessor node to see if the local part of the rule is satisfied. When \( v \) is grounded using a constant, the local part of the rule is checked at the current node and the successor part at the respective constant. Note that the first term pattern in the first conjunct in both rules (21) and (22) is irrelevant as \( \beta \) contains only unary predicates.

\[ \delta(q_{t,s}, \sigma) = \bigwedge_{u \in \beta} (\varepsilon, q_{t,u}) \land \bigwedge_{u \in \gamma \cup \delta} (\varepsilon, q_{t,s}) \] (21)

Finally we specify the parity acceptance condition. The index of the automaton is 2, with \( F_1 = \{ q_{t,a}, q_{t,s}, f \mid a \in upr(P), f \in bpr(P), t, s, t, s \in PatP \} \) and \( F_2 = Q \). Intuitively, paths in a run of the automaton correspond to dependencies of literals in the accepted model and by disallowing the infinite occurrence on a path of states associated to atoms in the model we ensure that only well-supported models are accepted.

**Theorem 2** Let \( P \) be a FoLP and let \( p \) be a unary predicate symbol. Then, \( p \) is satisfiable w.r.t. \( P \) iff there exists an automaton \( A_P^P, \theta \) such that \( \mathcal{L}(A_P^P, \theta) \neq \emptyset \).
Proof Sketch. ($\Rightarrow$): Assume $p$ is satisfiable w.r.t. $P$. Then, by Prop. 1, $p$ is satisfied by a forest model $(U, M)$. Let $(E, F, e, f)$ with $EF = (F, E_S)$ be the labelled forest which induces $(U, M)$, as defined in 2. Then, let $t : \{T_s | s \in \text{cts}(P) \cup \{\rho\}\}$ be a labelling function such that for every $y \in \{T_s | s \in \text{cts}(P) \cup \{\rho\}\}$, $l(y)$ is the least set containing: (i) $ef(y, o), o \in \text{cts}(P)$, (ii) $\{\eta^y_j\} f \in ef(y, z), z = \prec P(y, z), \forall z \notin \text{cts}(P)$, (iii) $\{\eta^y_j\} f \in ef(z, y), z = \prec P(y, z), \forall z \notin \text{cts}(P)$, (iv) $\{\eta^y_j\} f \in ef(z, y), z = \prec P(y, z), \forall z \notin \text{cts}(P)$.

We define an automaton $A_{\rho, \theta}^P$ which accepts $(F, l)$.

$\rho$ be the same as its homonym in the considered forest model and let $\theta$ be such that $\theta(o) = l(o)$, for every $o \in \text{cts}(P) \cup \{\rho\}$. We construct a run $(T_e, r)$ of $A_{\rho, \theta}^P$ on $(F, l)$ by first setting $r(c) = (\phi, q_0)$, where $\phi$ is the root of the same forest in $F$. Then, for each $o \in \text{cts}(P) \cup \{\rho\}$, we introduce a successor of $c, c \cdot i$, such that $r(c \cdot i) = (o, q_0)$.

We then proceed inductively with the construction by maintaining the following invariant, for each $w \in T_e$:

- $r(w) = (y, q_{\rho,\alpha})$ implies $\alpha \in l(y)$;
- $r(w) = (y, q_{\rho,\alpha})$ implies $\alpha \notin l(y)$ or $\alpha \in l(\theta(o))$,

and there is a rule $s \in P^M$ derived from $r_a$ such that $\text{head}(s) = a(y)$, $\text{level}(s) = \text{level}((a, M))$.

- $r(w) = (y, q_{\rho,\alpha})$ implies $\alpha \in l(y)$, if $u = a; \alpha \notin l(y)$, if $u = \alpha; \gamma^y_j \notin l(y)$, if $u = f; \gamma^y_j \notin l(y)$, if $u = \alpha$;

and there is a rule $s \in P^M$ derived from $r$ such that $\text{head}(s) = f(\text{prece}(y, z), y), \text{level}(s) = \text{level}(f(\text{prece}(y, z), y), M)$,

- $r(w) = (y, q_{\rho,\alpha})$ implies $\gamma^y_j \in l(y)$, if $u = f; \gamma^y_j \notin l(y)$, if $u = \alpha$;

and there is a rule $s \in P^M$ derived from $r$ such that $\text{head}(s) = f(y, o), \text{level}(s) = \text{level}(f(y, o), M)$,

- $r(w) = (y, q_{\rho,\alpha})$ implies $\gamma^y_j \notin l(y)$, if $u = \alpha$;

and there is a rule $s \in P^M$ derived from $r$ such that $\text{head}(s) = f(y, o), \text{level}(s) = \text{level}(f(y, o), M)$,

- $r(w) = (y, q_{\rho,\alpha})$ implies $\gamma^y_j \notin l(y)$, if $u = \alpha$;

and there is a rule $s \in P^M$ derived from $r$ such that $\text{head}(s) = f(y, o), \text{level}(s) = \text{level}(f(y, o), M)$,

- $r(w) = (y, q_{\rho,\alpha})$ implies $\gamma^y_j \in l(y)$, if $u = f; \gamma^y_j \notin l(y)$, if $u = \alpha$;

and there is a rule $s \in P^M$ derived from $r$ such that $\text{head}(s) = f(y, o), \text{level}(s) = \text{level}(f(y, o), M)$,

- $r(w) = (y, q_{\rho,\alpha})$ implies $\gamma^y_j \notin l(y)$, if $u = \alpha$;

and there is a rule $s \in P^M$ derived from $r$ such that $\text{head}(s) = f(y, o), \text{level}(s) = \text{level}(f(y, o), M)$,

- $r(w) = (y, q_{\rho,\alpha})$ implies $\gamma^y_j \notin l(y)$, if $u = \alpha$;

and there is a rule $s \in P^M$ derived from $r$ such that $\text{head}(s) = f(y, o), \text{level}(s) = \text{level}(f(y, o), M)$.

We show how the invariant is preserved in two cases of the construction:

- Assume $r(w) = (y, q_{\rho,\alpha})$, for some $w \in T_e$. Then, from the IH: $a \in l(y)$, and $a(y) \in M$. Then, there must be a finite $n$ such that $\text{level}(a(y), M) = n$ and some rule $s \in P^M$ such that $\max_{b \in \text{body}(s)}(\text{level}(b, M)) = n - 1$ (from which $a(y)$ has been derived at iteration $n$). Let $r_a$ be the rule in $P$ from which $s$ has been derived and let $w \cdot i$ be a successor of $w$ in $T_e$, such that $r(w \cdot i) = (y, q_{\rho,\alpha})$. The invariant is preserved.

- Assume $r(w) = (y, q_{\rho,\alpha})$, for some $w \in T_e$. Then, from the IH: $a \in l(y)$, $a$ is not free, and there exists some rule $s \in P^M$ derived from $r_a$: $a(y) \leftarrow \beta^y(y) \cup \bigcup_{i \in \text{cts}(P) \cup \{\rho\}}(\gamma^y_i(y, z_i) \cup \delta^y_i(z_i))$ such that $M = \text{body}(s)$ and $\max_{b \in \text{body}(s)}(\text{level}(b, M)) < \text{level}(a(y), M)$. Note that $M = \beta(y) \cup \bigcup_{i = 1}^{\text{max}(y, z_i) \cup \delta_i(z_i)}$. (As $U$ is a forest model, each $z_i$ must be of the form $y - k$, for some $1 \leq k \leq d$, or of the form $o \in \text{cts}(P)$. Let $J_u$ be a multiset such that $j_i = z_i$, if $z_i \in \text{cts}(P)$ and $j_i = k$ if $z_i = y - k$. Then, we introduce the following successors of $w$ in $T_e$ (denoted just by their label):

- $(y, q_{\rho,\alpha})$, for every $u \in \beta$; the invariant holds as $M = \beta(y)$;
- $(y, q_{\rho,\alpha})$, for every $u \in \beta$; the invariant holds as $M = \beta(y) \cup \bigcup_{i = 1}^{\text{max}(y, z_i) \cup \delta_i(z_i)}$.

The invariant ensures the existence of a run. We next show that every run $T_e$ constructed as above is accepting, i.e. on every path of $T_e$, there are finitely many occurrences of states of the form $q_{\rho,\alpha}$ or $q_{\rho,\alpha}^i$. Assume that every element $w \in T_e$ for which $r(w) = (y, q_{\rho,\alpha})$, $r(w) = (y, q_{\rho,\alpha})$, or $r(w) = (y, q_{\rho,\alpha})$ is annotated with $\text{level}(a(y), M)$, $\text{level}(f(\text{prece}(y, z), y), M)$, or $\text{level}(f(y, o), M)$, resp. From the invariant of the construction it follows that level annotations decrease along each path of $T_e$. But the level of every atom in an open answer set is finite. Thus, the number of level annotations, and consequently the number of occurrences of such states must be finite.

($\Leftarrow$): Assume that there exists an automaton $A_{\rho, \theta}^P$ such that $\mathcal{L}(A_{\rho, \theta}^P) \neq \emptyset$. Then, there exists a labelled forest $(F', f')$ and an accepting run $(T_r, r)$ on $(F', f')$ such that $r(c) = (\phi, q_0)$, for some root $\phi$ in $F'$. Let $F$ be the forest containing the nodes $y \in N_F$, for which either (i) there exists some $w \in T_e$ such that $r(w) = (y, q_{\rho,\alpha})$ or (ii) $\text{prece}(y) \in N_F$ and there exists some $w \in T_e$ such that $r(w) = (y, q_{\rho,\alpha}^i, t, w)$. Assume $C$ is the set of roots in $F$. Then let $\eta : N_F \rightarrow \text{cts}(P) \cup \{\rho\}$ be as follows: $\eta(y) = \{o_i, \text{ if } o_i \in l(y), \gamma(\eta) \cdot s, \text{ if } y = c \cdot s, \text{ for } c \in C \}$ and let $l$ be the following labeling function for $N_F$: $l(y) =$...
\[ \begin{cases} \theta(\eta(y)), & \text{if } \eta(y) \in cts(P) \\ f'(y), & \text{if } \eta(y) \notin NF \setminus cts(P) \end{cases} \]

Also, let: \( U = \{ \eta(y) \mid y \in NF \}, M = \{ a(\eta(y)) \mid a \in l(y) \cup r(P) \wedge y \in NF \} \cup \{ f(\eta(z), \eta(z) \cdot i) \mid \gamma(z) \notin l(y) \wedge y = \cdot \cdot y \in NF \} \).

We show that \((U, M)\) is a forest model of \( P \) by showing that:

(i) \( M \) is a model of \( P^U \), i.e. every rule in \( P^U \) is satisfied by \( M \): we check that every rule in \( P^U \) is satisfied.

Let \( r' : a(\eta(y)) \leftarrow \beta^*(\eta(y)), (\gamma_1^* \eta(y)), (\eta(y)) \), \( \delta_1^*(\eta(y)) \) be a rule in \( P^U \) derived from a unary rule \( r : a(s) \leftarrow \beta(s), (\gamma(s), \cdot i), \delta_1(t(s)) \) in \( P \). Let \( J_i \) be the multiset formed of elements \( j_i, 1 \leq i \leq m \), such that:

\[ j_i = \begin{cases} \eta(y_i), & \text{if } \eta(y_i) \in cts(P), \\ k, & \text{if } \eta(y_i) = \eta(x), \cdot k. \end{cases} \]

Assume \( M \models body(r') \), but \( M \not\models a(\eta(y)) \). Then, \( \not\models l(y) \) and there must be some \( w \in T_c \) such that either:

- \( r(w) = (y, q_{\eta(w)}) \). Then, for every rule \( r_a : a(s) \leftarrow \beta \in P \), there must be a node \( w_{r_a} \in T_c \) such that \( r(w_{r_a}) = (y, q_{\eta(w)}) \). This holds also for rule \( r \). Then, one of the following holds:

  * there exists \( w' \in T_c \) such that \( r(w') = (y, q_{\eta(w)}) \), for some \( u \in \beta \); then, either \( u = a \) and \( y, q_{\eta(w)} \in T_c \) and thus \( \not\models a(\eta(y)) \) contradiction, or for some \( u = \not\models a(\eta(y)) \) contradiction.

  * for every multiset \( J_{e_a} \) as in transition rule (14) (including \( J_e \) and above) and every \( j_i \in J_{e_a} \) either:

    - there exist 1 \( k \leq d, 1 \leq i \leq m, \) and \( u \in \gamma_i \cup \delta_i \) such that \( j_i = k \) and for every \( y : g \in F \), there is a node \( w_{g} \in T_c \) such that \( r(w_{g}) = (y, \cdot k, q_{\gamma_i \cup \delta_i}) \); then, there is \( w_{k} \in T_c \) such that \( r(w_{k}) = (y \cdot g, q_{\gamma_i \cup \delta_i}) \) and (1) \( k \notin l(y \cdot k) \) contradiction, (2) \( u = a, \not\models l(y \cdot k) \); \( a(\eta(y \cdot k)) \not\models M \) or \( M \not\models a(\eta(y)) \) and thus \( M \not\models \delta_i(\eta(y_i)) \) contradiction, (3) \( u = \not\models a(\eta(y)) \) and thus \( M \not\models \gamma_1(\eta(y_1)), (\eta(y))) \) contradiction, (4) \( u = f, \not\models l(y \cdot k) \); \( f(\eta(y), \eta(y) \cdot k) \not\models M \) or \( M \not\models f(\eta(y), \eta(y)) \), and thus \( M \not\models \gamma_1(\eta(y)), (\eta(y)) \) contradiction, (5) for some \( u = \not\models a(\eta(y)) \) and Thus, in each case we obtained a contradiction and \( M \models a(\eta(y)) \); every unary rule is satisfied by \( M \). Similarly it can be shown that every binary rule is satisfied as well.


(ii) \( M \) is minimal: from the fact that \((T_c, r)\) is accepting, it follows that every path starting at a state in \( Q^+ \) must be finite. For every node \( w \in T_c \), such that \( r(w) = (y, q) \), for some \( q \in Q^+ \), let:

\[ d(w) = \begin{cases} 0, & \text{if } w \text{ has no successors in } T_c, \\ 1 + \max_{w' \in T_c}(d(w \cdot i)), & \text{otherwise}. \end{cases} \]

We show by induction on \( d(w) \) that:

- \( r(w) = (y, q_{e,a}) \) implies \( a(\eta(y)) \in M(P^U) \),
- \( r(w) = (y, q_{o,a}) \) implies \( a(\eta(y)) \in M(P^U) \),
- \( r(w) = (y, q_{t,a}) \) implies \( P^U \) contains a rule \( a(\eta(y)) \) such that \( M(P^U) \models \beta \),
- \( r(w) = (y, q_{s,x,r}) \) implies \( y = z \cdot i, \) for some \( i, \) and \( P^U \) contains \( f(\eta(z), \eta(z) \cdot i) \) such that \( M(P^U) \models \beta \),
- \( r(w) = (y, q_{t,o,r}) \) implies \( P^U \) contains \( f(\eta(y), o) \) such that \( M(P^U) \models \beta \),
- \( r(w) = (y, q_{s,x,r}) \) implies \( y = z \cdot i, \) and: \( a(\eta(y)) \in M(P^U) \), if \( u = a; \not\models a(\eta(y)) \) \( \not\models M(P^U) \), if \( u = \not\models a; \not\models f(\eta(z), \eta(z) \cdot i) \in M(P^U) \), if \( u = f; \not\models f(\eta(z), \eta(z) \cdot i) \) \( \not\models M(P^U) \), if \( u = \not\models f; \)
- \( r(w) = (y, q_{t,o,u}) \) implies \( a(o) \in M(P^U) \), if \( u = a; \not\models a(\eta(y)) \) \( \not\models M(P^U) \), if \( u = \not\models a; \not\models f(\eta(y), o) \in M(P^U) \), if \( u = f; \not\models f(\eta(y), o) \) \( \not\models M(P^U) \), if \( u = \not\models f; \)
- \( r(w) = (y, q_{t,o,u}) \) implies \( a(o) \in M(P^U) \), if \( u = a; \not\models a(\eta(y)) \) \( \not\models M(P^U) \), if \( u = \not\models a; \not\models f(\eta(y), o) \in M(P^U) \), if \( u = f; \not\models f(\eta(y), o) \) \( \not\models M(P^U) \), if \( u = \not\models f; \)
- \( r(w) = (y, q_{o,u}) \) implies \( a(o) \in M(P^U) \), if \( u = a; \not\models a(\eta(y)) \) \( \not\models M(P^U) \), if \( u = \not\models a; \not\models a(\eta(y)) \) \( \not\models M(P^U) \), if \( u = \not\models a; \not\models f(\eta(y), o) \in M(P^U) \), if \( u = f; \not\models f(\eta(y), o) \) \( \not\models M(P^U) \), if \( u = \not\models f; \)

**Induction base:** assume \( w \) is a leaf in \( T_c \) and \( r(w) = (y, q) \), for some \( q \in Q^+ \) (\( d(w) = 0 \)). Then, one of the following holds:

- \( r(w) = (y, q_{e,a}) \) and \( a \) is free: in this case \( a(\eta(y)) \in M \) and thus \( P^U \) will contain rule \( a(\eta(y)) \). Thus, \( a(\eta(y)) \in M(P^U) \).
- \( r(w) = (y, q_{t,a}, f) \) and \( f \) is free: similar to above.
- \( r(w) = (y, q_{s,x,t}, a) \) and \( u = a \) or \( u = \not\models f; \) from the fact that \((T_c, r)\) is a run and transition rules (17-20), it follows that \( a(\not\models l(y) \cdot \gamma_2(t_2 = *) \cdot \gamma_2(t_2 = o) \not\models l(y) \). Then, the claim follows from the definition of \( M \).

**Induction step:** consider a non-leaf node \( w \in T_c \) (\( d(w) > 0 \)). We will analyse the case when \( r(w) = (y, q_{e,a}) \); then, for every \( u \in \beta \) there is a successor \( w \cdot i_{g,u} \) of \( w \) such that \( r(w \cdot i_{g,u}) = (y, q_{t,a} \cdot u) \) and there exists a multiset \( J_{e_a} \) and successors \( w \cdot m_{k,a} \) and \( w \cdot m_{o,a} \) for \( w \) such that:

- \( r(w \cdot m_{k,a}) = (y, z_{k,i,a}) \), for all pairs \((k, i)\) such that \( j_i = k \) and \( u \in \gamma_i \cup \delta_i; \)
\[ r(w \cdot m_{o,i,u}) = (y, q_{t,o,u}), \text{ for all pairs } (o,i) \text{ such that } j_i = o \text{ and } u \in \gamma_i \cup \delta_i. \]

From the IH, it follows that:

\[ M(P_{U}^M) \models u(y)(y), \text{ for every } u \in \beta, \text{ and thus } M(P_{U}^M) \models \beta(y)(y) (*1). \]

\[ M(P_{U}^M) \models u(\eta(y), \eta(y) \cdot z_k), \text{ for every } u \in \gamma_i \text{ and } M(P_{U}^M) \models u(\eta(y) \cdot z_k), \text{ for every } u \in \delta_i \text{ and thus } M(P_{U}^M) \models \gamma_i(\eta(y) \cdot z_k) \cup \delta_i(\eta(y) \cdot z_k) (*2), \]

\[ M(P_{U}^M) \models u(\eta(y), o), \text{ for every } u \in \gamma_i \text{ and } M(P_{U}^M) \models u(o), \text{ for every } u \in \delta_i, \text{ where } j_i = o, \text{ and thus } M(P_{U}^M) \models \gamma_i(\eta(y), o) \cup \delta_i(o) (*3). \]

From (*1)-(*3) and the fact that \( j_i \neq j_i \) implies \( z_{j_i} \neq z_{j_i} \), it follows that

\[ M(P_{U}^M) \models \beta(\eta(y)) \cup \bigcup_{l \in \subseteq U} \gamma_i(\eta(y), \eta(y) \cdot z_k) \cup \delta_i(\eta(y) \cdot z_k) j_i = k \cup \bigcup_{l \in \subseteq U} \gamma_i(\eta(y), o) \cup \delta_i(o) j_i = o, \psi, \]

which is the body of a grounding of \( r_a \) in \( P_U \) with head \( a(\eta(y)) \). Then, by applying the reduct one obtains that

\[ M(P_{U}^M) \models \beta(\eta(y)) \cup \bigcup_{l \in \subseteq U} \gamma_i(\eta(y), \eta(y) \cdot z_k) \cup \delta_i(\eta(y) \cdot z_k) j_i = k \cup \bigcup_{l \in \subseteq U} \gamma_i(\eta(y), o) \cup \delta_i(o) j_i = o, \]

which is the body of a rule with head \( a(\eta(y)) \) in \( P_{U}^M \).

The other cases can be treated similarly.

Then, as \( a \in \subseteq y \cap \upr(P) \) implies that there exists a node \( w \in T_r \) such that \( r(w) = (y, q_{t,x,y}, f) \), \( j_f \in \subseteq y \) implies that there exists a node \( w \in T_r \) such that \( r(w) = (y, q_{t,x,y}, f) \), and \( \gamma_f \in \subseteq y \) implies that there exists a node \( w \in T_r \) such that \( r(w) = (y, q_{t,o,f}) \) (from transition rule (7)), it follows that \( M \subseteq M(P_{U}^M) \), and thus \( M \) is a minimal model of \( P_{U}^M \).

\[ \square \]

**Theorem 3** Satisfiability checking of unary predicates \( w.r.t. \) FoLPs is \( \text{ExpTIME}-\text{complete}. \)

**Proof Sketch.** That the task is \( \text{ExpTIME}\)-hard follows from the fact that satisfiability checking of unary concepts \( w.r.t. \) \( SHOQ \) ontologies is \( \text{ExpTIME}\)-complete (Schild 1991); the latter has been reduced to satisfiability checking of unary predicates \( w.r.t. \) \( \text{FoLPs in} \) (Feier and Heymans 2013).

For the upper bound, we observe that for a given \( P \) and \( p \), there is an exponential number of automata \( A^\phi_{p,0} \) (as there are an exponential number of possible \( \theta \)). Each such automaton can be constructed in exponential time, and emptiness checking for \( A^\phi_{p,0} \) can be performed again in exponential time. The latter follows from Theorem 1 and the fact that the branching factor and the number of states of \( A^\phi_{p,0} \) are polynomial in the size of \( P \), while its index is constant (viz. 2). Then, from Theorem 2 it follows that satisfiability checking of unary predicates \( w.r.t. \) FoLPs is in \( \text{ExpTIME} \).

\[ \square \]

**Discussion and Conclusion**

We have described a reduction of satisfiability checking of unary predicates \( w.r.t. \) FoLPs to emptiness checking of FEAs. This enabled us to establish a tight complexity bound on this reasoning task for FoLPs. Other reasoning tasks like consistency checking of FoLPs and skeptical and brave entailment of ground atoms can be polynomially reduced to satisfiability checking of unary predicates (Heymans 2006); thus, the complexity result applies to those tasks as well. Also, by virtue of the translation from fKBs to FoLPs, the result applies to fKBs as well; satisfiability checking of unary predicates \( w.r.t. \) fKBs is \( \text{ExpTIME}\)-complete. Thus, reasoning with FoLP rules and \( SHOQ \) ontologies is not harder than reasoning with \( SHOQ \) ontologies themselves.

In the introduction, we mentioned the special form of forest model property as one of the reasons FoLPs cannot be straightforwardly dealt with using 2ATAs. An additional technical difficulty we had to overcome compared to both encodings using 2ATAs and other encodings using FEAs, is the fact that FEAs, unlike 2ATAs, cannot explicitly address successor nodes in runs: they are suitable for encoding graded modalities/number restrictions where the same property has to hold/not to hold at a given number of successors.

In our encoding, in order to assert that different properties hold at different successors (as the structure of FoLP rules allows), we had to devise an addressing mechanism for successors. Furthermore, as terms occurring in any position in FoLP rules might be constants, our encoding also made use of a unification mechanism.

Finally, as our result shows that FoLPs have the same complexity as CoLPs, we plan to further investigate the extension of the deterministic AND/OR tableau algorithm for CoLPs (Feier 2014) to FoLPs. As explained in (Feier 2014), such an extension is far from trivial.

**References**


Combining rules and ontologies via parametrized logic programs

Ricardo Gonçalves
NOVA LINCS
Faculdade de Ciências e Tecnologia
Universidade NOVA de Lisboa

Abstract

Parametrized logic programs are very expressive logic programs that generalize normal logic programs under the stable model semantics, by allowing complex formulas of a parameter logic to appear in the body and head of rules. In this paper we explore the use of description logics as parameter logics, and show the expressivity of this framework for combining rules and ontologies.

Introduction

Parametrized logic programming (Gonçalves and Alferes 2010) was introduced as an extension of answer set programming (Gelfond and Lifschitz 1988) with the motivation of providing a meaning to theories combining both logic programming connectives with other logical connectives, and allowing complex formulas using these connectives to appear in the head and body of a rule. The main idea is to fix a monotonic logic \( L \), called the parameter logic, and build up logic programs using formulas of \( L \) instead of just atoms. The obtained parametrized logic programs have, therefore, the same structure of normal logic programs, being the only difference the fact that atomic symbols are replaced by formulas of \( L \).

When applying this framework, the choice of the parameter logic depends on the domain of the problem to be modeled. As examples, (Gonçalves and Alferes 2010) shows how to obtain the answer-set semantics of logic programs with explicit negation, a paraconsistent version of it, and also the semantics of MKNF hybrid knowledge bases (Motik and Rosati 2010), using an appropriate choice of the parameter logic. In (Gonçalves and Alferes 2012) deontic logic programs are introduced using standard deontic logic (von Wright 1951) as the parameter logic. Moreover, in (Gonçalves and Alferes 2013) the decidability and implementation of parametrized logic was discussed.

Parametrized logic programming can thus be seen as a framework which allows to add non-monotonic rule based reasoning on top of an existing (monotonic) language. This view is quite interesting, in particular in those cases where we already have a monotonic logic to model a problem, but we are still lacking some conditional or non-monotonic reasoning. In these situations, parametrized logic programming offers a modular framework for adding such conditional and non-monotonic reasoning, without having to give up of the monotonic logic at hand.

In recent years, there has been a considerable amount of effort devoted to combining Description Logics (DLs) with logic programming non-monotonic rules – see, e.g., related work in (Eiter et al. 2008; Motik and Rosati 2010).

In this paper we explore precisely the use of description logics as parameter logics, and show the expressivity of the resulting framework for combining rules and ontologies.

Parametrized logic programs

Parametrized logic programs are very expressive logic programs that generalize normal logic programs under the stable model semantics, by allowing complex formulas of a parameter logic to appear in the body and head of rules. In this section we introduce the syntax and semantics of normal parametrized logic programs (Gonçalves and Alferes 2010).

Language

The syntax of a normal parametrized logic program has the same structure of that of a normal logic program. The only difference is that the atomic symbols of a normal parametrized logic program are replaced by formulas of a parameter logic, which is restricted to be a monotonic logic. Let us start by introducing the necessary concepts related with the notion of (monotonic) logic.

Definition 1 A (monotonic) logic is a pair \( L = (L, \vdash_L) \) where \( L \) is a set of formulas and \( \vdash_L \) is a Tarskian consequence relation (Wójcicki 1988) over \( L \), i.e., satisfying the following conditions, for every \( T \cup \Phi \subseteq L \).

1. Reflexivity: if \( \varphi \in T \) then \( T \vdash_L \varphi \);
2. Cut: if \( T \vdash_L \varphi \) for all \( \varphi \in \Phi \), and \( \Phi \vdash_T \psi \) then \( T \vdash_L \psi \);
3. Weakening: if \( T \vdash_L \varphi \) and \( T \subseteq \Phi \) then \( \Phi \vdash_L \varphi \).

When clear from the context we write \( \vdash \) instead of \( \vdash_L \).

Let \( \text{Th}(L) \) be the set of logical theories of \( L \), i.e. the set of subsets of \( L \) closed under the relation \( \vdash_L \). One fundamental characteristic of the above definition of monotonic logic is that it has as a consequence that, for every (monotonic) logic...
\( L \), the tuple \( \langle \text{Th}(L), \subseteq \rangle \) is a complete lattice with smallest element the set \( \text{Tho} = \emptyset \) of theorems of \( L \) and the greatest element the set \( L \) of all formulas of \( L \). Given a subset \( A \) of \( L \) we denote by \( A^c \) the smallest logical theory of \( L \) that contains \( A \). \( A^c \) is also called the logical theory generated by \( A \) in \( L \).

In what follows we consider fixed a (monotonic) logic \( L = (L, \vdash_L) \) and call it the parameter logic. The formulas of \( L \) are dubbed (parametrized) atoms and a (parametrized) literal is either a parametrized atom \( \varphi \) or its negation \( \neg \varphi \), where as usual \( \not \) denotes negation as failure. We dub default literal those of the form \( \not \varphi \).

**Definition 2** A normal \( L \) parametrized logic program is a set of rules
\[
\varphi \leftarrow \psi_1, \ldots, \psi_n, \neg \delta_1, \ldots, \neg \delta_m \tag{1}
\]
where \( \varphi, \psi_1, \ldots, \psi_n, \delta_1, \ldots, \delta_m \in L \).

A definite \( L \) parametrized logic program is a set of rules without negations as failure, i.e. of the form
\[
\varphi \leftarrow \psi_1, \ldots, \psi_n \text{ where } \varphi, \psi_1, \ldots, \psi_n \in L.
\]

As usual, the symbol \( \leftarrow \) represents rule implication, the symbol \( \not \) represents default negation. A rule as \( (1) \) has the usual reading that \( \varphi \) should hold whenever \( \psi_1, \ldots, \psi_n \) hold and \( \delta_1, \ldots, \delta_m \) are not known to hold. If \( n = 0 \) and \( m = 0 \) then we just write \( \varphi \leftarrow \).

Given a rule \( r \) of the form \( (1) \), we define
\[
\text{head}(r) = \varphi, \quad \text{body}^+(r) = \{\psi_1, \ldots, \psi_n\}, \quad \text{body}^-(r) = \{\delta_1, \ldots, \delta_m\} \quad \text{and} \quad \text{body}(r) = \text{body}^+(r) \cup \text{body}^-(r).
\]

A parametrized logic program \( P \) we define \( \text{form}(P) \) to be the set of all formulas of the parameter language \( L \) appearing in \( P \), i.e.,
\[
\text{form}(P) = \bigcup_{r \in P} \{\text{head}(r) \} \cup \text{body}(r).
\]
We also define the set \( \text{head}(P) = \{\text{head}(r) : r \in P\} \).

**Semantics**

Given this general language of parametrized logic programs, we define its stable model semantics, as generalization of the stable model semantics (Gelfond and Lifschitz 1988) of normal logic programs.

In the traditional approach an interpretation is just a set of atoms. In a parametrized logic program, since we substitute atoms by formulas of a parameter logic, the first idea is to take sets of formulas of the parameter logic as interpretations. The problem is that, contrary to the case of atoms, the parametrized atoms are not independent of each other. This interdependence is governed by the consequence relation of the parameter logic. For example, if we take classical propositional logic (CPL) as the parameter logic, we have that if the parametrized atom \( p \land q \) is true then so are the parametrized atoms \( p \) and \( q \). If we take, for example, standard deontic logic SDL (von Wright 1951) as parameter, we have that, since \( \text{O}(p \land q), \text{O}(\neg p)^{\text{SDL}}, \text{O}(q) \), any SDL logical theory containing both \( \text{O}(p \land q) \) and \( \text{O}(\neg p) \) also contains \( \text{O}(q) \).

To account for this interdependence, we use logical theories (sets of formulas closed under the consequence of the logic) as the generalization of interpretations, thus capturing the above mentioned interdependence.

**Definition 3** A (parametrized) interpretation is a logical theory of \( L \).

**Definition 4** An interpretation \( T \) satisfies a rule
\[
\varphi \leftarrow \psi_1, \ldots, \psi_n, \neg \delta_1, \ldots, \neg \delta_m
\]
if \( \varphi \in T \) whenever \( \psi_i \in T \) for every \( i \in \{1, \ldots, n\} \) and \( \delta_j \not\in T \) for every \( j \in \{1, \ldots, m\} \).

An interpretation is a model of logic program \( P \) if it satisfies every rule of \( P \). We denote by \( \text{Mod}_L(P) \) the set of models of \( P \).

The ordering over interpretations is the usual one: If \( T_1 \) and \( T_2 \) are two interpretations then we say that \( T_1 \leq T_2 \) if \( T_1 \subseteq T_2 \). Moreover, given such ordering, minimal and least interpretations may be defined in the usual way.

As in the case of non parametrized programs, we start by assigning semantics to definite parametrized programs. Recall that the stable model of a definite logic program is its least model. In order to generalize this definition to the parametrized case we need to establish that the least parametrized model exists for every definite \( L \) parametrized logic program.

**Theorem 1** Every definite \( L \) parametrized logic program has a least model.

We denote by \( S^\text{le}_P \) the least model of a definite program \( P \).

It is important to note that Theorem 1 holds for every choice of the parameter logic \( L \).

The stable model semantics of a normal \( L \) parametrized logic program is defined using a Gelfond-Lifschitz like operator.

**Definition 5** Let \( P \) be a normal \( L \) parametrized logic program and \( T \) an interpretation. The GL-transformation of \( P \) modulo \( T \) is the program \( P^T \) obtained from \( P \) by performing the following operations:

- remove from \( P \) all rules which contain a literal \( \not \varphi \) such that \( T \vdash_L \varphi \);
- remove from the remaining rules all default literals.

Since \( P^T \) is a definite \( L \) parametrized program, it has an unique least model \( J \). We define \( \Gamma(T) = J \).

Stable models of a parametrized logic program are then defined as fixed points of this \( \Gamma \) operator.

**Definition 6** An interpretation \( T \) of an \( L \) parametrized logic program \( P \) is a stable model of \( P \) if \( \Gamma(T) = T \). A formula \( \varphi \) is true under the stable model semantics, denoted by \( P \models_{\text{sms}} \varphi \) if it belongs to all stable models of \( P \).

An important feature of parametrized logic programming is that its stable model semantics is independent of the semantics of the parameter logic, since the central concept is the consequence relation of the parameter logic.

Let us now show an example of how parametrized logic programs can be used to combine a monotonic formalism with a non-monotonic one. We choose three different logics over the same propositional language.
Example 1 (Propositional logic programs) Let us now consider a full propositional language \( L \) built over a set \( P \) of propositional symbols using the usual connectives \( (\neg, \lor, \land, \rightarrow) \). Many consequence relations can be defined over this language. We will then illustrate the expressivity of the parameter logic in the framework of parametrized logic programming. We will then illustrate the expressivity of the parameter logic in the framework of parametrized logic programming.

In this section we discuss the use of description logics as parameter logics in the framework of parametrized logic programming. We will then illustrate the expressivity of the parameter logic in the framework of parametrized logic programming.

Let now \( \mathcal{L} = (L, \neg, \land, \lor, \rightarrow, \top) \) be Classical Propositional Logic (CPL) over the language \( L \). Let us study the semantics of CPL. Note that every logical theory of CPL that does not contain neither \( p \) nor \( \neg p \) satisfies \( P_1 \). In particular, the set \( Taut \) of tautologies of CPL is a model of \( P_1 \). So, \( S_{P_1}^{CPL} = Taut \). This means that \( p, \neg p, q, \neg q \notin S_{P_1}^{CPL} \). We also have that \( S_{P_2}^{CPL} = \{p\} \). So, in the case of \( P_2 \) we have that \( p \in S_{P_2}^{CPL} \).

In the case of \( P_3 \) its stable models are the theories of CPL that contain \( p \) and do not contain \( q \) and \( \neg q \). Therefore, we can conclude that \( p \in S_{P_3}^{CPL} \). In the case of \( P_0 \), since \( (p \lor \neg p) \in T \) for every logical theory \( T \) of CPL we can conclude that the only stable model of \( P_0 \) is the set \( T \) of theorems of CPL. Therefore \( p \notin S_{P_0}^{CPL} \).

Consider now \( \mathcal{L} = (L, \neg, \land, \lor, \top) \) the 4-valued Belnap paraconsistent logic Four. Consider the program \( P_0 \). Contrarily to the case of CPL, in Four it is not the case that \( \neg p, (p \lor q) \lor \bot \). Therefore we have that \( q, s \notin S_{P_0}^{Four} \).

Let now \( \mathcal{L} = (L, \neg, \land, \lor, \top) \) be the propositional intuitionistic logic IPL. It is well-known that \( q \lor \neg q \) is not a theorem of IPL. Therefore, according to the case of CPL, we can conclude that \( p \notin S_{P_2}^{CPL} \). Using the same idea for program \( P_2 \) we can conclude, contrarily to the case of CPL, that \( p \in S_{P_0}^{CPL} \).

Combining rules and ontologies

In this section we discuss the use of description logics as parameter logic in the framework of parametrized logic programming. We will then illustrate the expressivity of the resulting framework to combine non-monotonic rules and ontologies.

In what follows, and for simplicity, we use description logic \( ALC \) (Schmidt-Schaub and Smolka 1991). We start by briefly recalling the syntax and semantics of \( ALC \). For a more general and thorough introduction to DLs we refer to (Baader et al. 2010).

The language of \( ALC \) is defined over countably infinite sets of concept names \( N_C \), role names \( N_R \), and individual names \( N_I \) as shown in the upper part of Table 1. Building on these, complex concepts are introduced in the middle part of Table 1, which, together with atomic concepts, form the set of concepts. We conveniently denote individuals by \( a \) and \( b \), (atomic) roles by \( R \) and \( S \), atomic concepts by \( A \) and \( B \), and concepts by \( C \) and \( D \). All expressions in the lower part of Table 1 are axioms. A concept equivalence \( C \equiv D \) is an abbreviation for \( C \in D \) and \( D \in C \). Concept and role assertions are ABox axioms and all other axioms TBox axioms, and an ontology is a finite set of axioms.

The semantics of \( ALC \) is defined in terms of interpretations \( I = (\Delta^I, \tau^I) \), which consist of a non-empty domain \( \Delta^I \) and an interpretation function \( \tau^I \). The latter is defined for (arbitrary) concepts, roles, and individuals as in Table 1. Moreover, an interpretation \( I \) satisfies an axiom \( \alpha \), written \( I \models \alpha \), if the corresponding condition in Table 1 holds. If \( I \) satisfies all axioms in an ontology \( O \), then \( I \) is a model of \( O \), written \( I \models O \). If \( O \) has at least one model, then it is called consistent, otherwise inconsistent. Also, \( O \) entails axiom \( \alpha \), written \( O \models \alpha \), if every model of \( O \) satisfies \( \alpha \).

Given the consequence relation of \( ALC \) we can now illustrate how \( ALC \) can be used as parameter logic.

Example 2 The following program \((P_1)\) is an adaptation of an example taken from (Motik and Rosati 2007), which uses MKNF knowledge bases to combine rules and ontologies. The scenario is about determining the car insurance pre-
mium based on various information about the driver.

NotMarried ≡ ¬Married ←
NotMarried ⊑ HighRisk ←
∃Spouse.⊤⊑ Married ...

Every stable model of \( P_3 \) contains Discount(Bill), so the Stable Model Semantics of \( P_3 \) does not entail ¬Married(Bill) nor HighRisk(Bill).

Consider now program \( P_4 \) obtained by adding to \( P_2 \) the facts: Spouse(Bob, Ann) ←, p(Bob) ←, and p(Ann) ←. Every stable model of \( P_4 \) contains Discount(Bob), and so it entails Discount(Bob).

Conclusions

In this paper we have discussed the use of the framework of parametrized logic programming for combining non-monotonic rules and ontologies. This approach is quite expressive since it allows complex DL axioms to appear both in the body and in the head of non-monotonic rules.

In (Gonçalves and Alferes 2010) the authors show how parametrized logic programming can capture the semantics of MKNF hybrid knowledge bases (Motik and Rosati 2010) by an appropriate choice of the parameter logic. As future work we aim to study the relation between parametrized logic programs and other frameworks for combining rules and ontologies, e.g., the DL-programs of (Eiter et al. 2008).

References


On the Influence of Incoherence in Inconsistency-tolerant Semantics for Datalog±

C. A. Deagustini and M. V. Martinez and M. A. Falappa and G. R. Simari
AI R&D Lab., Dep. of Computer Science and Engineering,
Universidad Nacional del Sur, Bahía Blanca, Argentina
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET),
Avenida Rivadavia 1917, Ciudad Autónoma de Buenos Aires, Argentina

Abstract

The concept of incoherence naturally arises in ontological settings, specially when integrating knowledge. In this work we study a notion of incoherence for Datalog± ontologies based on the definition of satisfiability of a set of existential rules regarding the set of integrity constraints in a Datalog± ontology. We show how classical inconsistency-tolerant semantics for query answering behaves when dealing with atoms that are relevant to unsatisfiable sets of existential rules, which may hamper the quality of answers—even under inconsistency-tolerant semantics, which is expected as they were not designed to confront such issues. Finally, we propose a notion of incoherency-tolerant semantics for query answering in Datalog±, and present a particular one based on the transformation of classic Datalog± ontologies into defeasible Datalog± ones, which use argumentation as its reasoning machinery.

Introduction and Motivation

The problem of inconsistency in ontologies has been widely acknowledged in both the Semantic Web and Database Theory communities, and several methods have been developed to deal with it, e.g., (Arenas, Bertossi, and Chomicki 1999; Lembo et al. 2010; Lukasiewicz, Martinez, and Simari 2012; Black, Hunter, and Pan 2009; Bienvenu 2012; Martinez et al. 2014). The most widely accepted semantics for querying inconsistent databases is that of consistent answers (Arenas, Bertossi, and Chomicki 1999) (or $AR$ semantics in (Lembo et al. 2010) for ontological languages), which yields the set of atoms that can be derived despite all possible ways of repairing the inconsistency. In this semantics often an assumption is made that the set of ontological knowledge $\Sigma$ expresses the semantics of the data and as such there is no internal conflict on the set of constraints, which is not subject to changes over time. This means first, that the set of constraints is always satisfiable, in the sense that their application do not inevitably yield a consistency problem; second, as a result of the previous observation, it must be the case that the conflicts come from the data contained in the database instance and that is the part of the ontology that must be modified in order to restore consistency.

Although to consider the constraints as always satisfiable is a reasonable assumption to make, specially in the case of a single ontology, in this work we will focus on a more general setting and consider that both data and constraints can change through time and become conflicting. In this more general scenario, as knowledge evolves (and so the ontology that represents it) not only data related issues can appear, but also constraint related ones. The problem of conflicts among constraints is known in the Description Logics community as incoherence (Flouris et al. 2006; Qi and Hunter 2007). As they were not developed to consider this kind of issue, several of the well-known inconsistency-tolerant semantics for query answering fail at computing good quality answers in the presence of incoherence. In this paper we focus on a particular family of of ontological languages, namely Datalog± (Cali, Gottlob, and Lukasiewicz 2012a).

We show how incoherence can arise in Datalog± ontologies, and how the reasoning technique based on the use of defeasible elements in Datalog± and an argumentative semantics introduced by Martinez et al. (2014) can tolerate such issues, thus resulting in a reasoning machinery suitable of dealing with both incoherent and inconsistent knowledge.

This work integrates three different building blocks: first, we introduce the notion of incoherence for Datalog± ontologies, relating it to the problem of satisfiability of concepts for Description Logics; second, we show how such notion affects most of well-known inconsistency-tolerant semantics which, since they were not designed to confront such issues, can go up to the point of not returning any useful answer; finally, we propose a definition for incoherency-tolerant semantics, introducing an alternative semantics based on an argumentative reasoning process over the transformation of Datalog± ontologies to their correspondent defeasible Datalog± ontologies. We show how this semantics behaves in a satisfactory way in the presence of incoherence, as the process can return as answers atoms that trigger incoherency, which we show that cannot be done by classical inconsistency-tolerant semantics.

Preliminaries

First, we briefly recall some basics on Datalog± (Cali, Gottlob, and Lukasiewicz 2012a). We assume (i) an infinite universe of (data) constants $\Delta$ (which constitute the “normal” domain of a database), (ii) an infinite set of (labeled) nulls...
A conjunctive query (CQ) over $\mathcal{R}$ has the form $Q(X) = \exists Y \Phi(X, Y)$, where $\Phi(X, Y)$ is a conjunction of atoms (possibly equalities, but not inequalities) with the variables $X$ and $Y$, and possibly constants, but without nulls. In this work we restrict our attention to atomic queries. A Boolean CQ (BCQ) over $\mathcal{R}$ is a CQ of the form $Q$, often written as the set of all its atoms, without quantifiers. The set of answers for a CQ $Q$ over $\mathcal{D}$ and $\Sigma$, denoted $\text{ans}(Q, D, \Sigma)$, is the set of all tuples $t$ such that $t \in Q(D)$. The answer for a BCQ $Q$ over $\mathcal{D}$ and $\Sigma$ is $\text{Yes}$, denoted $D \cup \Sigma \models Q$, if and only if $\text{ans}(Q, D, \Sigma) \neq \emptyset$. It is important to remark that BCQs $Q$ over $\mathcal{D}$ and $\Sigma$, denoted $D \cup \Sigma \models Q$, may be evaluated on the chase for $D$ and $\Sigma$, i.e., $D \cup \Sigma \models Q$ is equivalent to $\text{chase}(D, \Sigma) \models Q$. 

A conjunctive query (CQ) over $\mathcal{R}$ has the form $Q(X) = \exists Y \Phi(X, Y)$, where $\Phi(X, Y)$ is a conjunction of atoms (possibly equalities, but not inequalities) with the variables $X$ and $Y$, and possibly constants, but without nulls. In this work we restrict our attention to atomic queries. A Boolean CQ (BCQ) over $\mathcal{R}$ is a CQ of the form $Q$, often written as the set of all its atoms, without quantifiers. The set of answers for a CQ $Q$ over $\mathcal{D}$ and $\Sigma$, denoted $\text{ans}(Q, D, \Sigma)$, is the set of all tuples $t$ such that $t \in Q(D)$. The answer for a BCQ $Q$ over $\mathcal{D}$ and $\Sigma$ is $\text{Yes}$, denoted $D \cup \Sigma \models Q$, if and only if $\text{ans}(Q, D, \Sigma) \neq \emptyset$. It is important to remark that BCQs $Q$ over $\mathcal{D}$ and $\Sigma$, denoted $D \cup \Sigma \models Q$, may be evaluated on the chase for $D$ and $\Sigma$, i.e., $D \cup \Sigma \models Q$ is equivalent to $\text{chase}(D, \Sigma) \models Q$.
Example 1 Consider the following KB.

\[
D : \{a_1 : \text{can_sing(simone)}; \\
a_2 : \text{rock_singer(adl)}; \\
a_3 : \text{sing_loud(ronnie)}; \\
a_4 : \text{has_fans(ronnie)}; \\
a_5 : \text{manage(band, richard)}\}
\]

\[
\Sigma_{NC} : \{\tau_1 : \text{sore_throat}(X) \land \text{can_sing}(X) \rightarrow \perp, \\
\tau_2 : \text{unknown}(X) \land \text{famous}(X) \rightarrow \perp\}
\]

\[
\Sigma : \{\nu_1 : \text{manage}(X,Y) \land \text{manage}(X,Z) \rightarrow Y = Z\}
\]

\[
\Sigma_{T} : \{\sigma_1 : \text{rock_singer}(X) \rightarrow \text{sing_loud}(X), \\
\sigma_2 : \text{sing_loud}(X) \rightarrow \text{sore_throat}(X), \\
\sigma_3 : \text{has_fans}(X) \rightarrow \text{famous}(X), \\
\sigma_4 : \text{rock_singer}(X) \rightarrow \text{can_sing}(X)\}
\]

Following the classical notion of consistency, we say that a consistent Datalog\(\pm\) ontology has a non-empty set of models.

Consistency. A Datalog\(\pm\) ontology KB = \((D, \Sigma)\) is consistent iff \(\text{mods}(D, \Sigma) \neq \emptyset\). We say that KB is inconsistent otherwise.

Incoherence in Datalog\(\pm\)

The problem of obtaining consistent knowledge from an inconsistent knowledge base is natural in many computer science fields. As knowledge evolves, contradictions are likely to appear, and these inconsistencies have to be handled in a way such that they do not affect the quality of the information obtained from the knowledge base.

In the setting of Consistent Query Answering (CQA), database repairing, and inconsistency-tolerant query answering in ontological languages (Arenas, Bertossi, and Chomicki 1999; Lembo et al. 2010; Lukasiewicz, Martinez, and Simari 2012), often the assumption is made that the set of constraints \(\Sigma\) expresses the semantics of the data in the component \(D\), and as such there is no internal conflict on the set of constraints and these constraints are not subject to changes over time. We argue that it is also important to identify and separate the sources of conflicts in Datalog\(\pm\) ontologies. In the previous section we defined inconsistency of a Datalog\(\pm\) ontology based on the lack of models. From an operational point of view, conflicts appear in a Datalog\(\pm\) ontology whenever a NC or an EGD is violated, that is, whenever the body of one such constraint can be mapped to either atoms in \(D\) or atoms that can be obtained from \(D\) by the application of the TGDs in \(\Sigma_T \subseteq \Sigma\). Besides these conflicts, we will also focus on the relationship between the set of TGDs and the set of NCs and EGDs, as it could happen that (a subset of) the TGDs in \(\Sigma_T\) cannot be applied without always leading to the violation of the NCs or EGDs. Note that in this case clearly the data in the database instance is not the problem, as any database in which these TGDs are applicable will inevitably produce an inconsistent ontology. This issue is related to that of unsatisfiability problem of a concept in an ontology and it is known in the Description Logics community as incoherence (Flouris et al. 2006; Qi and Hunter 2007). Incoherence can be particularly important when combining multiple ontologies since the constraints imposed by each one of them over the data could (possibly) represent conflicting modellings of the application at hand. Clearly, the notions of incoherence and inconsistency are highly related; in fact, Flouris et al. (2006) establish a relation between incoherence and inconsistency, considering the former as a particular form of the latter.

Our proposed notion of incoherence states that given a set of incoherent constraints \(\Sigma\) it is not possible to find a set of atoms \(D\) such that \(\text{KB} = (D, \Sigma)\) is a consistent ontology and at the same time all TGDs in \(\Sigma_T \subseteq \Sigma\) are applicable in \(D\). This means that a Datalog\(\pm\) ontology KB can be consistent even if the set of constraints is incoherent, as long as the database instance does not make those dependencies applicable. On the other hand, a Datalog\(\pm\) ontology KB can be inconsistent even when the set of constraints is coherent. Consider, as an example, the following KB = \(\{(\text{tall}(\text{peter}), \text{small}(\text{peter})), (\text{tall}(X) \land \text{small}(X) \rightarrow \perp)\}\), where the (empty) set of dependencies is trivially coherent; the ontology is, nevertheless, inconsistent.

In the last decades, several approaches to handling inconsistency were developed in Artificial Intelligence and Database Theory (e.g., (Konieczny and Pérez 2002; Delgrande and Jin 2012; Arenas, Bertossi, and Chomicki 1999)). Some of the best known approaches deal with inconsistency by removing from the theory atoms, or a combination of atoms and constraints or rules. A different approach is to simultaneously consider all possible ways of repairing the ontology by deleting or adding atoms, as in most approaches to Consistent Query Answering (Arenas, Bertossi, and Chomicki 1999) (CQA for short). However, these data-driven approaches might not be adequate for an incoherent theory and may produce meaningless results. As we stated before, an incoherent set \(\Sigma\) renders inconsistent any ontology whose database instance is such that the TGDs are applicable; in particular cases this may lead to the removal of every single atom in a database instance in an attempt to restore consistency, resulting in an ontology without any valuable information, when it could be the case that it is the set of constraints that is ill-defined.

Before formalizing the notion of incoherence that we use in our Datalog\(\pm\) setting we need to identify the set of atoms relevant to a given set of TGDs. Intuitively, we say that a set of atoms \(A\) is relevant to a set \(T\) of TGDs if the atoms in the set \(A\) are such that the application of \(T\) over \(A\) generates the atoms that are needed to apply all dependencies in \(T\), i.e., \(A\) triggers the application of every TGD in \(T\). Formally, the definition of atom relevancy is as follows:

Definition 1 (Relevant Set of Atoms for a Set of TGDs)

Let \(R\) be a relational schema, \(T\) be a set of TGDs, and \(A\) a (possibly existentially closed) non-empty set of atoms, both over \(R\). We say that \(A\) is relevant to \(T\) iff for all \(\sigma \in T\) of the form \(\forall X \exists Y \Phi(X,Y) \rightarrow \exists Z \Psi(X,Z)\) it holds that \(\text{chase}(A,T) \models \exists X \exists Y \Phi(X,Y)\).

When it is clear from the context, if a singleton set \(A = \{a\}\) is relevant to \(T \subseteq \Sigma\), we just say that atom \(a\) is relevant to \(T\). The following example illustrates atom relevancy.
Example 2 (Relevant Set of Atoms) Consider the following constraints:

\[ \Sigma_T = \{ \sigma_1 : \text{supervises}(X,Y) \rightarrow \text{supervisor}(X), \]
\[ \sigma_2 : \text{supervisor}(X) \land \text{take_decisions}(X) \rightarrow \text{leads_department}(X,D), \]
\[ \sigma_3 : \text{employee}(X) \rightarrow \text{works_in}(X,D) \} \]

First, let us consider the set \( A_1 = \{ \text{supervises(walter, jessie)}, \text{take_decisions(walter),} \text{employee(jessie)} \} \). This set is a relevant set of atoms to the set of constraints \( \Sigma_T = \{ \sigma_1, \sigma_2, \sigma_3 \} \), since \( \sigma_1 \) and \( \sigma_2 \) are directly applicable to \( A_1 \) and \( \sigma_3 \) becomes applicable when we apply \( \sigma_1 \) (i.e., the chase entails the atom \( \text{supervisor(walter)} \)), which together with \( \text{take_decisions(walter)} \) triggers \( \sigma_3 \).

However, the set \( A_2 = \{ \text{supervises(walter, jessie)}, \text{take_decisions(gus)} \} \) is not relevant to \( \Sigma_T \). Note that even though \( \sigma_1 \) is applicable to \( A_2 \), the TGDs \( \sigma_2 \) and \( \sigma_3 \) are never applied in \( \text{chase}(A_2, \Sigma_T) \), since the atoms in their bodies are never generated in \( \text{chase}(A_2, \Sigma_T) \). For instance, consider the TGD \( \sigma_2 \in \Sigma_T \). In the chase of \( \Sigma_T \) over \( D \) we create the atom \( \text{supervisor(walter)} \), but nevertheless we still cannot trigger \( \sigma_3 \) since we do not have and cannot generate the atom \( \text{take_decisions(walter)} \), and the atom \( \text{take_decisions(gus)} \) that is already in \( A_2 \) does not match the constant value.

We now present the notion of coherence for Datalog\(^\pm\), which adapts the one introduced by Flouris et al. for DLs (Flouris et al. 2006). Our conception of (in)coherence is based on the notion of satisfiability of a set of TGDs w.r.t. a set of constraints. Intuitively, a set of dependencies is satisfiable when there is a relevant set of atoms that triggers the application of all dependencies in the set and does not produce the violation of any constraint in \( \Sigma_{nc} \cup \Sigma_r \), i.e., the TGDs can be satisfied along with the NCS and EGDs in \( KB \).

**Definition 2 (Satisfiability of a set of TGDs w.r.t. a set of constraints)** Let \( R \) be a relational schema, \( T \subseteq \Sigma_r \), be a set of TGDs, and \( N \subseteq \Sigma_{nc} \cup \Sigma_r \), both over \( R \). The set \( T \) is satisfiable w.r.t. \( N \) iff there is a set \( A \) of (possibly existentially closed) atoms over \( R \) such that \( A \) is relevant to \( T \) and \( \text{mods}(A, T \cup N) \neq \emptyset \). We say that \( T \) is unsatisfiable w.r.t. \( N \) iff \( T \) is not satisfiable w.r.t. \( N \). Furthermore, \( \Sigma_r \) is satisfiable w.r.t. \( \Sigma_{nc} \cup \Sigma_r \) iff there is no \( T \subseteq \Sigma_r \) such that \( T \) is unsatisfiable w.r.t. some \( N \) with \( N \subseteq \Sigma_{nc} \cup \Sigma_r \).

In the rest of the paper sometimes we write that a set of TGDs is (un)satisfiable omitting the set of constraints, we do this in the context of a particular ontology where we have a fixed set of constraints \( \Sigma_{nc} \cup \Sigma_r \). Also, through the paper we denote by \( U(KB) \) the set of minimal unsatisfiable sets of TGDs in \( \Sigma_r \) for \( KB \) (i.e., unsatisfiable set of TGDs such that every proper subset of it is satisfiable). The following example illustrates the concept of satisfiability of a set of TGDs in a Datalog\(^\pm\) ontology.

**Example 3 (Unsatisfiable sets of dependencies)** Consider the following sets of constraints.

\[ \Sigma_{nc}^1 = \{ r : \text{risky_job}(P) \land \text{unstable}(P) \rightarrow \bot \} \]
\[ \Sigma_r^1 = \{ \sigma_1 : \text{dangerous_work}(W) \land \text{works_in}(W, P) \rightarrow \text{risky_job}(P), \]
\[ \sigma_2 : \text{in_therapy}(P) \rightarrow \text{unstable}(P) \} \]

The set \( \Sigma_{nc}^1 \) is a satisfiable set of TGDs, and even though the simultaneous application of \( \sigma_1 \) and \( \sigma_2 \) may violate some formula in \( \Sigma_{nc}^1 \cup \Sigma_r^1 \), that does not hold for every relevant set of atoms. Consider as an example the relevant set \( D_1 = \{ \text{dangerous_work(policeman)}, \text{works_in(policeman, martyr), in_therapy(rust)} \} \). \( D_1 \) is a relevant set for \( \Sigma_{nc}^1 \), however, as we have that \( \text{mods}(D_1, \Sigma_{nc}^1 \cup \Sigma_1^1 \cup \Sigma_2^1) \neq \emptyset \) then \( \Sigma_r^1 \) is satisfiable.

On the other hand, as an example of unsatisfiability consider the following constraints:

\[ \Sigma_{nc}^2 = \{ \tau_1 : \text{sorethroat}(X) \land \text{can} \_ \text{sing}(X) \rightarrow \bot \} \]
\[ \Sigma_r^2 = \{ \sigma_1 : \text{rock} \_ \text{singer}(X) \rightarrow \text{sing} \_ \text{load}(X), \]
\[ \sigma_2 : \text{sing} \_ \text{load}(X) \rightarrow \text{sorethroat}(X), \]
\[ \sigma_3 : \text{rock} \_ \text{singer}(X) \rightarrow \text{can} \_ \text{sing}(X) \} \]

The set \( \Sigma_r^2 \) is an unsatisfiable set of dependencies, as the application of TGDs \( \{ \sigma_1, \sigma_2, \sigma_3 \} \) on any relevant set of atoms will cause the violation of \( \tau_1 \). For instance, consider the relevant atom \( \text{rock_singer(axl)} \): we have that the application of \( \Sigma_r^2 \) over \( \{ \text{rock} \_ \text{singer(axl)} \} \) causes the violation of \( \tau_1 \) when considered together with \( \Sigma_{nc}^2 \), therefore \( \text{mods}(\{ \text{rock} \_ \text{singer(axl)} \}) \cup \Sigma_{nc}^2 \cup \Sigma_r^2 \cup \Sigma_{nc}^1 = \emptyset \). Note that any set of relevant atoms will cause the violation of \( \tau_1 \).

We are now ready to formally define coherence for a Datalog\(^\pm\) ontology. Intuitively, an ontology is coherent if there is no subset of their TGDs that is unsatisfiable w.r.t. the constraints in the ontology.

**Definition 3 (Coherence)** Let \( KB = (D, \Sigma) \) be a Datalog\(^\pm\) ontology defined over a relational schema \( R \), and \( \Sigma = \Sigma_r \cup \Sigma_{nc} \cup \Sigma_E \), where \( \Sigma_r \) is a set of TGDs, \( \Sigma_{nc} \) a set of separable EGDs and \( \Sigma_E \) a set of negative constraints. \( KB \) is coherent iff \( \Sigma_r \) is satisfiable w.r.t. \( \Sigma_{nc} \cup \Sigma_E \). Also, \( KB \) is said to be incoherent iff it is not coherent.

**Example 4 (Coherence)** Consider the sets of dependencies and constraints defined in Example 3 and an arbitrary database instance \( D \). Clearly, the Datalog\(^\pm\) ontology \( KB_1 = (D, \Sigma_{nc}^1 \cup \Sigma_E^1 \cup \Sigma_r^1) \) is coherent, while \( KB_2 = (D, \Sigma_r^2 \cup \Sigma_{nc}^2 \cup \Sigma_E^2) \) is incoherent.

Finally, we look deeper into the relation between incoherence and inconsistency. Looking into Definitions 2 and 3 we can infer that an incoherent \( KB \) will induce an inconsistent \( KB \) when the database instance contains any set of atoms that is relevant to the unsatisfiable sets of TGDs. This result is captured in the following proposition.

**Proposition 1** Let \( KB = (D, \Sigma) \) be a Datalog\(^\pm\) ontology where \( \Sigma = \Sigma_r \cup \Sigma_{nc} \cup \Sigma_E \). If \( KB \) is incoherent and there exists \( A \subseteq D \) such that \( A \) is relevant to some unsatisfiable set \( U \in U(KB) \) then \( KB = (D, \Sigma) \) is inconsistent.

**Example 5 (Relating Incoherence and Inconsistency)** As an instance of the relationship expressed in Proposition 1, consider once again the ontology presented in Example 1. As hinted previously in Example 3, there we have the
set \( A \subset D = \{ \text{rock singer}(axl) \} \) and the unsatisfiable set of TGDs \( U \subset \Sigma_T = \{ \sigma_1 : \text{rock singer}(X) \rightarrow \text{sing loud}(X), \sigma_2 : \text{sing loud}(X) \rightarrow \text{sore throat}(X), \sigma_4 : \text{rock singer}(X) \rightarrow \text{can sing}(X) \} \). Since \( A \) is relevant to \( U \) the conditions in Proposition 1 are fulfilled, and indeed the ontology \( KB = (D, \Sigma) \) from Example 1 is inconsistent since \( \tau_1 \in \Sigma_T \) is violated.

**Incoherence influence on classic inconsistency-tolerant semantics**

We have established the relation between incoherence and inconsistency. As explained, classic inconsistency-tolerant techniques do not account for coherence issues since they assume that such kind of problems will not appear. Nevertheless, if we consider that both components in the ontology evolve (perhaps being collaboratively maintained by a pool of users) then certainly incoherence is prone to arise.

In the following we show that it may be important for inconsistency-tolerant techniques to consider incoherence in ontologies as well, since if not treated appropriately an incoherent set of TGDs may lead to the trivial solution of removing every single relevant atom in \( D \) (which in the worst case could be the entire database instance). This may be adequate for some particular domains, but does not seem to be a desirable outcome in the general case.

Although classical query answering in Datalog\(^+\) is not tolerant to inconsistency issues, a variety of inconsistency-tolerant semantics have been developed in the last decade for ontological languages, including lightweight Description Logics (DLs), such as \( \mathcal{EL} \) and \( \mathcal{DL-Lite} \) (Lukasiewicz, Martinez, and Simari 2012). In this section we analyze how incoherence influence in several inconsistency-tolerant semantics for ontological languages: \( AR \) semantics (Lembo et al. 2010), \( CAR \) semantics (Lembo et al. 2010), and provide some insights for sound approximations of \( AR \) and of \( CAR \).

We present the basic concepts needed to understand the different semantics for query answering on Datalog\(^+\) ontologies and then show how entailment under such semantics behaves in the presence of incoherence. The notion of repair in relational databases is a model of the set of integrity constraints that is maximally close, i.e., “as close as possible” to the original database.

Depending on how repairs are obtained we can have different semantics. In the following we recall \( AR \)-semantics (Lembo et al. 2010), one of the most widely accepted inconsistency-tolerant semantics, along with an alternative to \( AR \) called \( CAR \)-semantics.

**AR Semantics.** The \( AR \) semantics corresponds to the notion of consistent answers in relational databases (Arenas, Bertossi, and Chomicki 1999). Intuitively, an atom \( a \) is said to be \( AR \)-consistently entailed from a Datalog\(^+\) ontology \( KB \), denoted \( KB \models_{AR} a \) if \( a \) is classically entailed from every ontology that can be built from every possible A-box repair (a maximally consistent subset of the \( D \) component that after its application to \( \Sigma_T \) respects every constraint in \( \Sigma_T \cup \Sigma_N \)). We denote by \( KB \not\models_{AR} a \) the fact that \( a \) cannot be \( AR \)-consistently inferred from \( KB \).

The extent entailment to set of atoms straightforwardly, i.e., for a set of atoms \( A \) it holds that \( KB \models_{AR} A \) iff for every \( a \in A \) it holds that \( KB \models_{AR} a \), and \( KB \not\models_{AR} A \) otherwise.

**CAR Semantics.** As noted by Lembo et al. (2010), the \( AR \) semantics is not independent from the form of the knowledge base; it is easy to show that given two inconsistent knowledge bases that are logically equivalent, contrary to what one would expect, their respective repairs do not coincide. To address this, another definition of repairs was also proposed by Lembo et al. (2010) that includes knowledge that comes from the closure of the database instance with respect to the set of TGDs. Since the closure of an inconsistent ontology yields the whole language, they define the consistent closure of an ontology \( KB = (D, \Sigma_T \cup \Sigma_N \cup \Sigma_{NC}) \) as the set \( CCL(KB) = \{ \alpha \mid \alpha \in \mathcal{H}(\Sigma_T) \text{ s.t. } \exists \Sigma \subseteq D \text{ and } mods(S_\alpha \cup \Sigma_N \cup \Sigma_{NC}) \neq \emptyset \text{ and } (S, \Sigma_T \cup \Sigma_N) \models \alpha \} \). A Closed ABox repair of a Datalog\(^+\) ontology \( KB \) is a consistent subset \( D' \) of \( CCL(KB) \) such that it maximally preserves the database instance (Lembo et al. 2010). It is said that an atom \( a \) is \( CAR \)-consistently entailed from a Datalog\(^+\) ontology \( KB \), denoted by \( KB \models_{CAR} a \) iff \( a \) is classically entailed from every ontology built from each possible closed ABox repair. We extend entailment to set of atoms straightforwardly, i.e., for a set of atoms \( A \) it holds that \( KB \models_{CAR} A \) iff for every \( a \in A \) it holds that \( KB \models_{CAR} a \), and \( KB \not\models_{CAR} A \) otherwise.

Incoherence has great influence when calculating repairs, as can be seen in the following result: independently of the semantics (i.e., \( AR \) or \( CAR \)) no atom that is relevant to an unsatisfiable set of TGDs belongs to a repair of an incoherent KB.

**Lemma 1** Let \( KB = (D, \Sigma) \) be an incoherent Datalog\(^+\) ontology where \( \Sigma = \Sigma_T \cup \Sigma_N \cup \Sigma_{NC} \) and \( R(KB) \) be the set of \( (\text{A-box or Closed A-box}) \) repairs of \( KB \). If \( A \subseteq D \) is relevant to some unsatisfiable set \( U \in U(KB) \) then \( A \not\subseteq R \) for every \( R \in R(KB) \).

The proof of Lemma 1 follows from Proposition 1, since any set of atoms relevant to an unsatisfiable set of TGDs will be conflictive with \( \Sigma_{NC} \cup \Sigma_N \), thus not qualifying to be part of a proper repair.

**Example 6** Consider the atom \( \text{rock singer}(axl) \) from the ontology presented in Example 1. As we have explained in Example 5, such atom is relevant to \( U \subseteq \Sigma_T = \{ \sigma_1 : \text{rock singer}(X) \rightarrow \text{sing loud}(X), \sigma_2 : \text{sing loud}(X) \rightarrow \text{sore throat}(X), \sigma_4 : \text{rock singer}(X) \rightarrow \text{can sing}(X) \} \). It is easy to show that as a result of this the atom does not belong to any A-Box or Closed A-Box repair. Consider the case of A-Box repairs. We have that they are maximally consistent subsets of the component \( D \). We have that \( mods(\text{rock singer}(axl), \Sigma) = \emptyset \), as the NC \( \tau_1 : \text{sore throat}(X) \land \text{can sing}(X) \rightarrow \bot \) is violated. Moreover, clearly this violation happens for every set \( A \subseteq D \) such that \( \text{rock singer}(axl) \in A \), and thus we have that \( mods(A, \Sigma) = \emptyset \), i.e., \( \text{rock singer}(axl) \) cannot be part of any A-Box repair for the KB.

In an analogous way we can show that for any \( D' \subseteq D \) such that \( mods(D', \Sigma) \neq \emptyset \) it holds that
Let $KB = (D, \Sigma)$ be an incoherent Datalog$^\pm$ ontology where $\Sigma = \Sigma_T \cup \Sigma_{\text{re}} \cup \Sigma_{\text{nc}}$. If $A \subseteq D$ is relevant to some unsatisfiable set $U \subseteq \Sigma_T$, then $KB \not\models_{\text{AR}} A$ and $KB \not\models_{\text{CAR}} A$.

The proof follows from Lemma 1: since a relevant set of atoms does not belong to any repair then it cannot be part of the answers of the AR and CAR semantics. As a corollary, in the limit case that every atom in the database instance is relevant to some unsatisfiable subset of the TGDs in the ontology then the set of AR-answers (resp. CAR-answers) is empty.

Corollary 1 Let $KB = (D, \Sigma)$ be an incoherent Datalog$^\pm$ ontology where $\Sigma = \Sigma_T \cup \Sigma_{\text{re}} \cup \Sigma_{\text{nc}}$, and let $A_{\text{AR}}$ and $A_{\text{CAR}}$ be the set of atoms AR-consistently and CAR-consistently entailed from $KB$, respectively. If for every $a \in D$ there exists $A \subseteq D$ such that $a \in A$ and $A$ is a minimal set of TGDs relevant to some $U \in \mathcal{U}(KB)$ then $A_{\text{AR}} = \emptyset$ and $A_{\text{CAR}} = \emptyset$.

Since they follow from Proposition 1, both Proposition 2 and Corollary 1 can be straightforwardly extended to other repair based inconsistency-tolerant semantics such as ICAR and ICR (Lembo et al. 2010).

Example 7 Consider once again $KB$ in Example 1, and the atom $a_2 : \text{rock_singer}(axl)$ in $D$. Such atom is relevant to the unsatisfiable set $U \subseteq \Sigma_T = \{\sigma_1 : \text{rock_singer}(X) \rightarrow \text{sing_loud}(X), \sigma_2 : \text{sing_loud}(X) \rightarrow \text{sore_throat}(X), \sigma_3 : \text{rock_singer}(X) \rightarrow \text{can_sing}(X)\}$, and indeed it holds that $KB \not\models_{\text{AR}} \text{rock_singer}(axl)$ and $KB \not\models_{\text{CAR}} \text{rock_singer}(axl)$. As explained in Example 6, this is because $\text{rock_singer}(axl)$ cannot belong to any repair since its consistent application to $\Sigma$ is not feasible, i.e., $\text{mod}(\text{rock_singer}(axl), \Sigma) = \emptyset$.

Incoherence-tolerant semantics

We have shown how incoherence affects classic inconsistency-tolerant semantics up to the point of not returning any meaningful answer (since they were not developed to consider such kind of issues). In this section we propose the notion of tolerance to incoherence for query answering semantics. Such semantics will allow to be able to obtain useful answers from incoherent ontologies. We continue this section by showing an alternative semantics for Datalog$^\pm$ based on the use of argumentative inference that is tolerant to incoherence. For the elements of argumentation we refer the reader to (Besnard and Hunter 2008; Rahwan and Simari 2009).

Definition 4 (Incoherence-tolerant semantics) Let $KB = (D, \Sigma)$ be a Datalog$^\pm$ ontology where $\Sigma = \Sigma_T \cup \Sigma_{\text{re}} \cup \Sigma_{\text{nc}}$. A query answering semantics $S$ is said to be tolerant to incoherence (or incoherence-tolerant) iff there exists $A \subseteq D$ and $U \in \mathcal{U}(KB)$ such that $A$ is relevant to $U$ and it holds that $KB \models_S A$.

Intuitively, a query answering semantics is tolerant to incoherence if it can entail atoms that trigger incoherent sets of TGDs as answers. Clearly, from Proposition 2 it follows that inconsistency-tolerant semantics based on repairs are not tolerant to incoherence.

Observation 1 AR and CAR semantics are not incoherence-tolerant semantics.

An Incoherence-tolerant Semantics via Argumentative Inference

We begin by recalling Defeasible Datalog$^\pm$ (for the interested reader, a more complete presentation of the framework can be found in (Martinez et al. 2014)), and then we move on to show the behaviour of this semantics in the presence of incoherence.

Defeasible Datalog$^\pm$ (Martinez et al. 2014) is a variation of Datalog$^\pm$ that enables argumentative reasoning in Datalog$^\pm$ by means of transforming the information encoded in a KB to represent statements whose acceptance can be challenged. To do this, a Datalog$^\pm$ ontology is extended with a set of em defeasible atoms and defeasible TGDs; thus, a Defeasible Datalog$^\pm$ ontology contains both (classical) strict knowledge and defeasible knowledge. The set of defeasible TGDs allows to express weaker connections between pieces of information than in a classical TGDs. Defeasible TGDs are rules of the form $T(X, Y) \rightarrow \exists Z \Psi(X, Z)$, where $T(X, Y)$ and $\Psi(X, Z)$ are conjunctions of atoms. As in DeLP’s defeasible rules (García and Simari 2004), defeasible TGDs are used to represent weaker connections between the body and the head of a rule. Defeasible TGDs are written using the symbol “$\rightarrow$”, while the classical (right) arrow “$\rightarrow$” is reserved to strict TGDs and NCs.

Defeasible Datalog$^\pm$ Ontologies. A defeasible Datalog$^\pm$ ontology $KB$ consists of a finite set $F$ of ground atoms, called facts, a finite set $D$ of defeasible atoms, a finite set of TGDs $\Sigma_T$, a finite set of defeasible TGDs $\Sigma_D$, and a finite set of binary constraints $\Sigma_{\text{re}} \cup \Sigma_{\text{nc}}$.

The following example shows a defeasible Datalog$^\pm$ ontology that encodes the knowledge from Example 1 changing some of the facts and TGDs to defeasible ones.

Example 8 The information from the ontology presented in Example 1 can be better represented by the following defeasible Datalog$^\pm$ ontology $KB = (F, D, \Sigma_T, \Sigma_D, \Sigma_{\text{nc}})$ where $F = \{\text{can_sing}(\text{simone}), \text{rock_singer}(axl), \text{sing_loud}(\text{ronnie}), \text{has_fans}(\text{ronnie})\}$ and $D = \{\text{manage}(\text{band}, \text{richard})\}$. Note that we have changed the fact stating that Richard manages band1 to a defeasible one, since reports indicates that the members of band1 are looking for a new manager. The sets of TGDs, and defeasible TGDs are now given by the following sets: note that we have changed some of the TGDs into defeasible TGDs to make clear that the connection between the head and body is weaker.
\[ \Sigma_T = \{ \text{sing loud}(X) \rightarrow \text{sore throat}(X), \text{rock singer}(X) \rightarrow \text{can sing}(X) \} \\
\Sigma_D = \{ \text{rock singer}(X) \succ \text{sing loud}(X), \text{has fans}(X) \succ \ldots \cup \Sigma_{NC}, \text{otherwise will be called non-conflicting.} \}
\]

Derivations from a defeasible Datalog\(^+\) ontology rely in the application of (strict or defeasible) TGDs. Given a defeasible Datalog\(^+\) ontology \( KB = (F, D, \Sigma_T, \Sigma_D, \Sigma_{NC}) \), a (strict or defeasible) TGD \( \sigma \) is applicable if there exist a homomorphism mapping the atoms in the body of \( \sigma \) into \( F \cup D \). The application of \( \sigma \) on \( KB \) generates a new atom from the head of \( \sigma \) if it is not already in \( F \cup D \), in the same way as explained in the preliminaries of this work.

The following definitions follow similar ones first introduced by Martinez et al. (2012). Here we adapt the notions to defeasible Datalog\(^\pm\) ontologies. An atom has a derivation from a KB iff there is a finite sequence of applications of (strict or defeasible) TGDs that has the atom as its last component.

**Definition 5** Let \( KB = (F, D, \Sigma_T, \Sigma_D, \Sigma_{NC}) \) be a defeasible Datalog\(^\pm\) and \( L \) an atom. An annotated derivation \( \partial \) of \( L \) from \( KB \) consists of a finite sequence \([R_1, R_2, \ldots, R_n]\) such that \( R_n = L \) and each atom \( R_i \) is either: (i) \( R_i \) is a fact or defeasible atom, i.e., \( R_i \in F \cup D \), or (ii) there exists a TGD \( \sigma \in \Sigma_T \cup \Sigma_D \) and a homomorphism \( h \) such that \( h(\text{head}(\sigma)) = R_i \) and \( \sigma \) is applicable to the set of all atoms and defeasible atoms that appear before \( R_i \) in the sequence. When no defeasible atoms and no defeasible TGDs are used in a derivation, we say the derivation is a strict derivation, otherwise it is a defeasible derivation.

Note that there is non-determinism in the order in which the elements in a derivation appear; TGDs (strict and defeasible) can be reordered, and facts and defeasible atoms could be added at any point in the sequence before they are needed to satisfy the body of a TGD. These syntactically distinct derivations are, however, equivalent for our purposes. It is possible to introduce a canonical form for representing them and adopt that canonical form as the representative of all of them. For instance, we might endow the elements of the program from which the derivation is produced with a total order; thus, it is possible to select one derivation from the set of all the derivations of a given literal that involve the same elements by lexicographically ordering these sequences. When no confusion is possible, we assume that a unique selection has been made.

We say that an atom \( a \) is strictly derived from \( KB \) iff there exists a strict derivation for \( a \) from \( KB \), denoted with \( KB \vdash a \). A derivation \( \partial \) for \( a \) is minimal if no proper sub-derivation \( \partial' \) of \( \partial \) (every member of \( \partial' \) is a member of \( \partial \)) is also an annotated derivation of \( a \). Considering minimal derivations in a defeasible derivation avoids the insertion of unnecessary elements that will weaken its ability to support the conclusion by possibly introducing unnecessary points of conflict. Given a derivation \( \partial \) for \( a \), there exists at least one minimal sub-derivation \( \partial' \subseteq \partial \) for an atom \( a \). Thus, through the paper we only consider minimal derivations (Martinez et al. 2014).

Example 9 From the defeasible Datalog\(^{\pm}\) ontology in Example 8, we can get the following (minimal) annotated derivation for atom \( \text{sore throat}(axl) \):

\[
\partial = [\text{rock singer}(axl), \\
\text{rock singer}(X) \succ \text{sing loud}(X), \\
\text{sing loud}(axl), \\
\text{sore loud}(X) \rightarrow \text{sore throat}(X), \\
\text{sore throat}(axl)]
\]

Then, we have that \( KB \vdash \text{sore throat}(axl) \) and that \( KB \vdash \text{sore throat}(axl) \).

Classical query answering in defeasible Datalog\(^\pm\) ontologies is equivalent to query answering in Datalog\(^\pm\) ontologies.

**Proposition 3** (Martinez et al. 2014) Let \( L \) be a ground atom, \( KB = (F, D, \Sigma_T, \Sigma_D, \Sigma_{NC}) \) be a defeasible Datalog\(^\pm\) ontology, \( KB' = (F \cup D, \Sigma_T \cup \Sigma_{NC}) \) is a classical Datalog\(^\pm\) ontology where \( \Sigma_T = \Sigma_T \cup \{ \Gamma(X, Y) \rightarrow \exists Z \Psi(X, Z) | T(X, Y) \rightarrow \exists Z \Psi(X, Z) \} \). Then, \( KB' \models L \) iff \( KB \vdash L \) or \( KB' \vdash L \).

Proposition 3 states the equivalence between derivations from defeasible Datalog\(^\pm\) ontologies and entailment in traditional Datalog\(^\pm\) ontologies whose database instance corresponds to the union of facts and defeasible atoms, and the set of TGDs corresponds to the union of the TGDs and the strict version of the defeasible TGDs. As a direct consequence, all the existing work done for Datalog\(^\pm\) directly applies to defeasible Datalog\(^\pm\). In particular, it is easy to specify a defeasible Chase procedure over defeasible Datalog\(^\pm\) ontologies, based on the revised notion of application of (defeasible) TGDs, whose result is a universal model. Therefore, a (B)CQ \( Q \) over a defeasible Datalog\(^\pm\) ontology can be evaluated by verifying that \( Q \) is a classical consequence of the chase obtained from the defeasible Datalog\(^\pm\) ontology.

**Argumentation-based Reasoning in Defeasible Datalog\(^\pm\)**

Conflicts in defeasible Datalog\(^\pm\) ontologies come, as in classical Datalog\(^\pm\), from the violation of NCs or EGDs. Intuitively, two atoms are in conflict relative to a defeasible Datalog\(^\pm\) ontology whenever they are both derived from the ontology (either strictly or defeasible) and together map to the body of a negative constraint or they violate an equality-generating dependency.

**Definition 6** Given a set of NCs \( \Sigma_{NC} \) and a set of non-conflicting EGDs \( \Sigma_e \), two ground atoms (possibly with nulls) \( a \) and \( b \) are said to be in conflict relative to \( \Sigma_e \cup \Sigma_{NC} \) iff there exists an homomorphism \( h \) such that \( h(\text{body}(v)) = a \land b \) for some \( v \in \Sigma_{NC} \), or \( h(v) \neq h(Y_j) \) for some \( v \in \Sigma_e \) where \( h(X_i) \) is a term in \( a \) and \( h(Y_j) \) is a term in \( b \).

In what follows, we say that a set of atoms is a conflict set of atoms relative to \( \Sigma_{NC} \cup \Sigma_{NC} \), if and only if there exist at least two atoms in the set that are in conflict relative to \( \Sigma_e \cup \Sigma_{NC} \), otherwise will be called non-conflicting. Whenever is clear from the context we omit the set of NCs and EGDs.
Example 10 Consider the NC \{sore\_throat(X) \land can\_sing(X) \rightarrow \bot\} in \Sigma_{NC} from the defeasible ontology in Example 8. In this case, the set of atoms \{sore\_throat(axl), can\_sing(axl)\} is a conflicting set relative to \Sigma_{NC}. However, this is not the case for the set \(S = \{\text{rock\_singer}(axl)\}\); even when such set generates a violation when applied to the set of TGDs, it is not conflicting in itself.

Whenever defeasible derivations of conflicting atoms exist, we use a dialectical process to decide which information prevails, i.e., which piece of information is such that no acceptable reasons can be put forward against it. Reasons are supported by arguments; an argument is an structure that supports a claim from evidence through the use of a reasoning mechanism. We maintain the intuition that led to the classic definition of arguments by Simari and Loui (1992), as shown in the following definition.

Definition 7 Let KB be a defeasible Datalog\(^\pm\) ontology and \(L\) a ground atom. A set \(A\) of facts, defeasible atoms, TGDs, and defeasible TGDs used in an annotated derivation \(\partial\) of \(L\) is an argument for \(L\) constructed from KB iff \(\partial\) is a \(\subseteq\)-minimal derivation and no conflicting atoms can be defeasible derived from \(A \cup \Sigma_T\). An argument \(A\) for \(L\) is denoted \(\langle A, L \rangle\), and \(\Lambda_{KB}\) will be the set of all arguments that can be built from KB.

Example 11 Consider the derivation \(\partial\) in Example 9. We have that \{sore\_throat(axl), \} is an argument in \(\Lambda_{KB}\). Figure 1 shows the argument.

![Argument Diagram](image)

Figure 1: An argument for sore\_throat(axl).

Answers to atomic queries are supported by arguments built from the ontology. However, it is possible to build arguments for conflicting atoms, and so arguments can attack each other. We now adopt the definitions of counter-argument and attacks for defeasible Datalog\(^\pm\) ontologies from (García and Simari 2004). First, an argument \(\langle B, L' \rangle\) is a sub-argument of \(\langle A, L \rangle\) if \(B \subseteq A\). Argument \(\langle A_1, L_1 \rangle\) counter-argues, rebuts, or attacks \(\langle A_2, L_2 \rangle\) at literal \(L\), iff there exists a sub-argument \(\langle A, L \rangle\) of \(\langle A_2, L_2 \rangle\) such that \(L\) and \(L_1\) conflict.

Example 12 Consider derivation \(\partial\) from Example 9 and let \(A\) be the set of (defeasible) atoms and (defeasible) TGDs used in \(\partial\). \(A\) is an argument for sore\_throat(axl). Also, we can obtain a minimal derivation \(\partial'\) for can\_sing(axl) where \(B\), the set of (defeasible) atoms and (defeasible) TGDs used in \(\partial'\), is such that no conflicting atoms can be defeasibly derived from \(B \cup \Sigma_T\).

As \{sore\_throat(axl), can\_sing(axl)\} is conflicting relative to \(\Sigma_{NC}\), we have that \(\langle A, \text{sore\_throat(axl)} \rangle\) and \(\langle B, \text{can\_sing(axl)} \rangle\) attack each other.

![Attack Diagram](image)

Figure 2: Attack between arguments.

Once the attack relation is established between arguments, it is necessary to analyze whether the attack is strong enough so one of the arguments can defeat the other. Given an argument \(A\) and a counter-argument \(B\), a comparison criterion is used to determine if \(B\) is preferred to \(A\) and, therefore, defeats \(A\). For our defeasible Datalog\(^\pm\) framework, unless otherwise stated, we assume an arbitrary preference criterion \(\succ\) among arguments where \(A \succ B\) means that \(B\) is preferred to \(A\) and thus defeats it. More properly, given two arguments \(\langle A_1, L_1 \rangle\) and \(\langle A_2, L_2 \rangle\) we say that argument \(\langle A_1, L_1 \rangle\) is a defeater of \(\langle A_2, L_2 \rangle\) iff there exists a sub-argument \(\langle A, L \rangle\) of \(\langle A_2, L_2 \rangle\) such that \(\langle A, L \rangle\) counter-argues \(\langle A, L \rangle\) at \(L\), and either \(\langle A_1, L_1 \rangle \succ \langle A, L \rangle\) (it is a proper defeater) or \(\langle A_1, L_1 \rangle \not \succ \langle A, L \rangle\) (it is a blocking defeater).

Finally, the combination of arguments, attacks and comparison criteria gives raise to Datalog\(^\pm\) argumentation frameworks.

Definition 8 Given a Defeasible Datalog\(^\pm\) ontology KB defined over a relational schema \(R\), a Datalog\(^\pm\) argumentation framework \(\mathcal{\Phi}\) is a tuple \(\langle R, \Lambda_{KB}, \succ \rangle\), where \(\succ\) specifies a preference relation defined over \(\Lambda_{KB}\).

To decide whether an argument \(\langle A_0, L_0 \rangle\) is undefeated within a Datalog\(^\pm\) argumentation framework, all its defeaters must be considered, and there may exist defeaters for their counter-arguments as well, giving raise to argumentation lines. The dialectical process considers all possible admissible argumentation lines for an argument, which together form a dialectical tree. An argument line for \(\langle A_0, L_0 \rangle\) is defined as a sequence of arguments that starts at \(\langle A_0, L_0 \rangle\), and every element in the sequence is a defeater of its predecessor in the line (García and Simari 2004). Note that for defeasible Datalog\(^\pm\) ontologies arguments in an argumentation line can contain both facts and defeasible atoms.

Different argumentation systems can be defined by setting a particular criterion for proper attack or defining the admissibility of argumentation lines. Here, we adopt the one...
from (García and Simari 2004), which states that an argumentation line has to be finite, and no argument is a sub-argument of an argument used earlier in the line; furthermore, when an argument \( (A_i, L_i) \) is used as a blocking defeater for \( (A_{i-1}, L_{i-1}) \) during the construction of an argumentation line, only a proper defeater can be used for defeating \( (A_i, L_i) \).

The dialectical process considers all possible admissible argumentation lines for an argument, which together form a dialectical tree. Dialectical trees for defeasible Datalog \( ^\pm \) ontologies are defined following (García and Simari 2004), and we adopt the notion of coherent dialectical tree from (Martínez, García, and Simari 2012), which ensures that the use of defeasible atoms is coherent in the sense that conflicting defeasible atoms are not used together in supporting (or attacking) a claim. We denote with \( \text{Args}(T) \) the set of arguments in \( T \).

**Definition 9** Let \( \langle A_0, L_0 \rangle \) be an argument from a Datalog \( ^\pm \) argumentation framework \( \mathcal{F} \). A dialectical tree for \( \langle A_0, L_0 \rangle \) from \( \mathcal{F} \), denoted \( T(\langle A_0, L_0 \rangle) \), is defined as follows:

1. The root of the tree is labeled with \( \langle A_0, L_0 \rangle \).
2. Let \( N \) be a non-root node of the tree that is labeled \( \langle A_n, L_n \rangle \). Let \( C^0 = \{\langle A_0, L_0 \rangle, \langle A_1, L_1 \rangle, \ldots, \langle A_n, L_n \rangle\} \) be the sequence of labels of the path from the root to \( N \). Let \( \langle B_1, Q_1 \rangle, \langle B_2, Q_2 \rangle, \ldots, \langle B_k, Q_k \rangle \) be all the defeaters for \( \langle A_n, L_n \rangle \). For each defeater \( \langle B_i, Q_i \rangle \) \( (1 \leq i \leq k) \), such that the argumentation line \( C^0 = \{\langle A_0, L_0 \rangle, \langle A_1, L_1 \rangle, \langle A_2, L_2 \rangle, \ldots, \langle A_n, L_n \rangle, \langle B_i, Q_i \rangle\} \) is admissible, the node \( N \) has a child \( N_i \) labeled \( \langle B_i, Q_i \rangle \). If there is no defeater for \( \langle A_n, L_n \rangle \) or there is no \( \langle B_i, Q_i \rangle \) such that \( C^0 \) is admissible, then \( N \) is a leaf.

Argument evaluation, i.e., determining whether the root node of the tree is defeated or undefeated, is done by means of a marking or labeling criterion. Each node in an argument tree is labelled as either defeated (\( D \)) or undefeated (\( U \)). We denote the dialectical tree built for the argument \( A \) supporting claim \( L \) as \( T(\langle A, L \rangle) \), \( \text{Args}(T) \) the set of arguments in \( T \), and the root of \( T(\langle A, L \rangle) \) with \( \text{root}(\langle A, L \rangle) \). Also, \( \text{marking}(N) \), where \( N \) is a node in a dialectical tree, denotes the value of the marking for node \( N \) (either \( U \) or \( D \)).

**Definition 10** Let \( KB \) be a Defeasible Datalog \( ^\pm \) ontology and \( \mathcal{F} \) the corresponding Datalog \( ^\pm \) argumentation framework where \( \succ \notin \mathcal{F} \) is an arbitrary argument comparison criterion. An atom \( L \) is warrantied in \( \mathcal{F} \) (through \( T \)) iff there exists an argument \( \langle A, L \rangle \) such that \( \text{marking}(\text{root}(T(\langle A, L \rangle))) = U \). We say that \( L \) is entailed from \( KB \) (through \( \mathcal{F} \)), denoted with \( KB \models_{\mathcal{F}} L \), iff it is warrantied in \( \mathcal{F} \).

**Example 13** Suppose that we have the query \( Q = \text{can\_sing(axl)} \), i.e., we want to know whether or not Axl can sing. Consider the conflict between arguments \( A \) and \( B \) shown in Example 12. As we have stated, we do not define any particular criterion \( \succ \) to solve attacks. Nevertheless, for the sake of example assume now that we are indeed using a criterion \( \succ \) that is such that \( B \succ A \). Under such supposition we have labelled the dialectical tree shown in Figure 3.

![Figure 3: A labelled dialectical tree for atom \( \text{can\_sing(axl)} \).](image)

As can be seen in the dialectical tree, if we assume that \( B \succ A \) then we have reasons to think that Axl cannot sing due to its throat being sore.

In Definition 10 we specify a semantics based on the use of argumentative inference. From now on we denote such semantics as \( D^2 \) (Defeasible Datalog \( ^\pm \)). Such semantics relies on the transformation of classic Datalog \( ^\pm \) ontologies to defeasible ones and then obtaining answers from the transformed one. So, we begin by establishing how a classic Datalog \( ^\pm \) ontology can be transformed to a defeasible one. Intuitively, the transformation of a classic ontology to a defeasible one involves transforming every atom and every TGD in the classic ontology to its defeasible version.

**Definition 11 (Transformation between ontologies)** Let \( KB = (D, \Sigma_F \cup \Sigma_E \cup \Sigma_{D\text{ont}}) \) be a classic Datalog \( ^\pm \) ontology. Then, its transformation to a defeasible Datalog \( ^\pm \) ontology, denoted \( D^2(KB) \), is a defeasible ontology \( KB' = (F', \Sigma_F', \Sigma_E', \Sigma_{D\text{ont}}, \Sigma_{D\text{ont}}') \) where \( F = \emptyset, D' = D, \Sigma'_F = \emptyset \) and \( \Sigma'_Z = \{T(X, Y) \rightarrow Z^\prime \Psi(X, Z) \mid T(X, Y) \rightarrow Z \Psi(X, Z) \} \).

Next, we define the set of answers in \( D^2 \) for an atomic query. Intuitively, a literal is an answer for a classical query and \( \succ \) a comparison criterion. Then, an atom \( L \) is an answer for \( Q \) from \( KB \) under \( D^2 \), denoted \( KB \models_{D^2} L \), iff \( KB \models_{\mathcal{F}} L \) where \( \mathcal{F} = (\mathcal{L}_{F}, \mathcal{L}_{D\text{ont}}, \succ) \).

Note that the semantics is parametrized by the comparison criterion \( \succ \), which helps to solve conflicts when they arise.
Influence of incoherence in Defeasible Datalog±

Now, we focus on the behaviour of Defeasible Datalog± regarding atoms relevant to unsatisfiable sets of TGDs. It can be shown that the argumentation framework $\mathcal{F} = (\mathcal{L}_R, A_{D(KB)}, \succ)$ is such that one relevant atom $L$ to an unsatisfiable set is warranted (and thus an answer), provided that the comparison criterion $\succ$ is such that $\text{marking}(\text{root}(\mathcal{T}_G((A, L)))) = U$ for some dialectical tree $\mathcal{T}_G((A, L))$ built upon $\mathcal{F}$. It is interesting to see that such comparison criterion can always be found: intuitively, it suffices to arbitrary establish $A$ as the most preferred argument in $A_{D(KB)}$ (note however that other criteria can have the exact same result).

**Proposition 4** Let $KB$ be a Datalog± ontology defined over a relational schema $R$, and $KB'$ be a Defeasible Datalog± ontology such that $D(KB) = KB'$. Finally, let $L \in D'$ and $U \in U(KB)$ such that $L$ is relevant to $U$. Then, it holds that there exists $\succ$ such that $KB \models_{D^L} L$.

**Corollary 2** (Corollary from Proposition 4) Given a Datalog± ontology $KB$ there exists $\succ$ such that $D^L_{\succ}$ applied to $KB$ is tolerant to incoherence.

As an example of the above corollary, consider again the running example.

**Example 14** Let $KB' = D(KB)$ be the defeasible transformation of $KB$ in Example 1, where the sets $\Sigma_\pi$ and $\Sigma_{\Sigma_C}$ are the same, $F = \emptyset$, $D = \{\text{can}_\text{sing}(\text{simone}), \text{rock}_\text{singer}(\text{axl}), \text{sing}_\text{loud}(\text{ronnie}), \text{has}_\text{fans}(\text{ronnie}), \text{manage}_\text{band}(\text{band}_1, \text{richard})\}$, and $\Sigma_{\pi} = \{\text{rock}_\text{singer}(X) \succ \text{sing}_\text{loud}(X), \text{sing}_\text{loud}(X) \succ \text{sore}_\text{throat}(X), \text{has}_\text{fans}(X) \succ \text{famous}(X), \text{rock}_\text{singer}(X) \succ \text{can}_\text{sing}(X)\}$.

Here, we have the dialectical tree with argument $\langle \text{rock}_\text{singer}(\text{axl}) \rangle$, $\text{rock}_\text{singer}(\text{axl})$ as its undefeated root, since no counterargument for it can be built (Figure 4).

[Diagram: A labelled dialectical tree for atom $\text{rock}_\text{singer}(\text{axl})$.]

Then, clearly $KB' \models_{\mathcal{F}} \text{rock}_\text{singer}(\text{axl})$, and thus $KB \models_{D^L} \text{rock}_\text{singer}(\text{axl})$.

Note that in Example 14 the atom $\text{rock}_\text{singer}(\text{axl})$ is warranted under any criterion comparison $\succ$, and thus we have not needed to perform any restriction on the criterion.

**Conclusions**

Incoherence is an important problem in knowledge representation and reasoning, specially when integrating different sources of information. Nevertheless, most of the works in query answering for Datalog± ontologies and DLs have focused on consistency issues making the assumption that the set of constraints correctly represents the semantics of the data and therefore any conflict can only come from the data itself.

In this work we have introduced the concept of incoherence for Datalog± ontologies, relating it to the presence of sets of TGDs such that their application inevitably yield to violations in the set of negative constraints and equality-generating dependencies. We have shown how incoherence affects classic inconsistency-tolerant semantics to the point that for some incoherent ontologies these semantics may produce no useful answer. Finally, we have introduced the concept of incoherence-tolerant semantics, and shown a particular semantics satisfying that property. Nevertheless, it is important to remark that our definition of incoherence-tolerant semantics is not tied to our particular proposal or the Datalog± language, and that there exists other frameworks that also falls under our definition, as it is the case of the work (also argumentation-based) by Black et al. (2009) where dialogue games between agents are used to solve queries under Description Logics ontologies that can be incoherent.

**References**


Paraconsistent Relational Model: A Quasi-Classic Logic Approach

Badrinath Jayakumar and Rajshekar Sunderraman
Department of Computer Science, Georgia State University, Georgia, USA
{bjayakumar2, raj}@cs.gsu.edu

Abstract

The well-founded model for any general deductive database computed using the paraconsistent relational model, based on four-valued logic, does not support inference rules such as disjunctive syllogism. In order to support disjunctive syllogism, we utilize the generalization of the relational model to quasi-classic (QC) logic, whose inference power is much closer to classical logic. As the paraconsistent relational model is capable of representing incomplete and inconsistent data, we propose an algorithm to find QC model for inconsistent positive extended disjunctive deductive databases. To accomplish this, in addition to using existing generalized algebraic operators, we introduce two new operators $FOCUS_C$ and $FOCUS_D$.

1 Introduction

Most database systems that are currently in use are based on the relational model. This model defines the falsity of its attributes based on the absence of information (closed world assumption). However, the model is not suitable for all applications. For example, if a database does not contain a person’s allergy to a medicine, when a doctor queries the person’s allergy to the medicine, the database will then return false. To represent negative information in databases, we use paraconsistent databases (Bagai and Sunderraman 1995). Bagai and Sunderraman developed a framework to represent negative facts in relational databases, which is based on four-valued logic. The four-valued relation represents both positive and negative information, and negative facts are derived from open world assumption. The authors (Bagai and Sunderraman 1995) also developed an application for the paraconsistent relational model that finds the weak well-founded semantics of general deductive databases. For general deductive databases and disjunctive deductive databases, various paraconsistent semantics have been proposed (Bagai and Sunderraman 1996b; Subrahmanian 1990; Sunderraman 1997), all based upon Belnap’s four-valued model (Belnap Jr 1977). Even for logic programs, especially disjunctive logic programs, various paraconsistent semantics are proposed (Alcântara, Damásio, and Pereira 2004; Arieli 2002; Damásio and Pereira 1998; Alcântara, Damásio, and Pereira 2002). All of which still come under Belnap’s four-valued model.

However, such multi-valued logic does not support disjunctive syllogism (Hunter 1998). For example, suppose a knowledge base contains $\{\text{passed} \lor \neg\text{failed}\}$ about a student. When new information about the student comes to the knowledge base $\{\neg\text{failed}\}$, the four-valued logic gives two paraconsistent minimal models (Sakama and Inoue 1995) $\{\{\text{passed}, \neg\text{failed}\}, \{\text{failed}, \neg\text{failed}\}\}$. Hence, $\neg\text{failed}$ is the only logical consequence of the two models. Whether the student has passed or not cannot be inferred with four-valued logic.

It is very clear from the above example that this four-valued logic does not behave as expected in such situations. In order to get accurate models, it is required to use a stronger form of four-valued logic for improving the ability of reasoning. Therefore, we use QC logic (Hunter 2000). In this paper, the algorithm to find the model for positive extended disjunctive deductive databases is proposed by using QC logic with the paraconsistent relational model. To avoid getting into an infinite iteration, it is assumed that the program does not have recursions, and constraints are also not used. In the model construction, for any disjunctive clause, it is necessary to impose a link between each relation in union and its complementary of the relation in order to enable non-trivial classical conclusions.

The approach presented in this paper differs from the QC model (Zhang, Lin, and Ren 2009) in its algebraic nature. In this paper, we construct an algebraic equation for every clause in which expressions of the equation containing cartesian products operates with a set of tuples instead of one tuple at a time. In other words, the expression that represents disjunctive head operates on one tuple at a time. In addition to that, algebraic expressions can be optimized by using various laws of equality.

The rest of the paper is organized as follows. In section 2, we introduce the preliminaries to understand the paper; in section 3, we define a paraconsistent disjunctive relational model; in section 4, we provide the key idea to understand the paper; in section 5, we explain the algorithm with an example; in section 6, we state the conclusion and future work for this paper.

2 Preliminaries

Before we explain the details of actual contributions of this paper, we must briefly review positive extended disjunctive
deductive databases and the paraconsistent relational model (Bagai and Sunderraman 1996a; Zhang, Lin, and Ren 2009; Bagai and Sunderraman 1995).

Syntax. Given a first order language \( \mathcal{L} \), a disjunctive deductive database \( P \) (Minker and Seipel 2002) consists of logical inference rules of the form

\[
\tau(\text{rule}) = l_0 \lor \cdots \lor l_n \leftarrow l_{n+1} \ldots l_m
\]

\( l_0 \ldots l_n \) is called head of the rule and \( l_{n+1} \ldots l_m \) is called body of the rule. A rule is called fact if the rule has no body. A rule is called denial rule if the rule has only body and no head. A rule is called definite clause or horn clause, if the rule has only one literal in the head and has some literal in the body. A rule is called positive disjunctive rule if the rule has both body and head. Concretely, the rule \( \tau \) is called positive extended disjunctive rule, if \( l_0, \ldots, l_n, l_{n+1}, \ldots, l_m \) are either positive or negative \((-\cdot-)\) literals.

For the given syntax of a positive extended disjunctive deductive database, we reproduce the fixed point semantics of \( P \) (Zhang, Lin, and Ren 2009).

Fixed Point Semantics. Let \( P \) be a positive extended disjunctive deductive database and \( \mathcal{I} \) be a set of interpretations, then \( T_P(\mathcal{I}) = \bigcup_{\mathcal{I} \in \mathcal{I}} T_P(I) \)

\[
T_P(I) = \begin{cases}
\emptyset, & \text{if } l_{n+1}, \ldots, l_m \subseteq I \text{ for some ground constraint } l \leftarrow l_{n+1} \ldots l_m \text{ from } P.

\{J \mid \text{for each ground rule } r_i; l_0 \lor \cdots \lor l_n \leftarrow l_{n+1} \ldots l_m \text{ such that } \{l_{n+1}, \ldots, l_m\} \subseteq I, J = I \cup \bigcup_{r_i} J' \text{ where } J' \in \text{Lits} \{\text{focus}(l_0 \lor \cdots \lor l_n, I)\}, & \text{otherwise.}
\end{cases}
\]

In the definition of \( T_P(I) \), focus removes complementary literals from disjunction \( \text{focus}(l_0 \lor l_1, I) = l_0 \) where \( I = \{l_1\} \). If all disjunctions \( l_0, l_1 \) are available in \( I \) as complementary literals, then the disjunction of literals becomes the conjunction of literals. Lits of conjunction gives a set of conjunctions. On the other hand, Lits of disjunction is a collection of sets where every set in the collection contains a disjunction.

The \( T_P \) definition contains the constraint. We write it for the sake of completeness, but our contribution in this paper will not address the constraint.

The following two propositions are vital for our result.

Proposition 1. For any positive extended disjunctive deductive database \( P \), \( T_P \) is finite and \( T_P \uparrow \uparrow n = T_P \uparrow \omega \) where \( n \) is a successor ordinal and \( \omega \) is a limit ordinal.

Proposition 2. For any positive extended disjunctive deductive database \( P \), Minimal QC Model \( P \) = \( \min(\mu(T_P \uparrow \omega)) \uparrow 1 \) where \( \min() \) stands for sets with a minimum number of literals.

As we stated earlier, we use the paraconsistent relation model (Bagai and Sunderraman 1995) to find the QC model for any given positive extended disjunctive deductive database. In the following, we define the paraconsistent relational model and its operators.

\( ^1\mu(T_P \uparrow \omega) = \{I \mid I \in T_P \uparrow \omega \text{ and } I \in T_P(\{I\})\} \)

Paraconsistent relations move forward a step to complete the database. Unlike normal relations where we only retain information believed to be true of a particular predicate, we also retain what is believed to be false of a particular predicate in the paraconsistent relational model. Let a relation scheme \( \Sigma \) be a finite set of attribute names, where for any attribute name \( A \in \Sigma \), \( \text{dom}(A) \) is a non-empty domain of values for \( A \). A tuple on \( \Sigma \) is any map \( t: \Sigma \rightarrow \bigcup_{A \in \Sigma} \text{dom}(A) \), such that \( t(A) \in \text{dom}(A) \) for each \( A \in \Sigma \). Let \( \tau(\Sigma) \) denote the set of all tuples on \( \Sigma \). An ordinary relation on scheme \( \Sigma \) is thus any subset of \( \tau(\Sigma) \). A paraconsistent relation on a scheme \( \Sigma \) is a pair \(< R^+, R^- > \) where \( R^+ \) and \( R^- \) are ordinary relations on \( \Sigma \). Thus, \( R^+ \) represents the set of tuples believed to be true of \( R \), and \( R^- \) represents the set of tuples believed to be false.

Algebraic Operators. Two types of algebraic operators are defined here: i) Set Theoretic Operators, and ii) Relational Theoretic Operators.

Set Theoretic Operators. Let \( R \) and \( S \) be two paraconsistent relations on scheme \( \Sigma \).

Union. The union of \( R \) and \( S \), denoted \( R \cup S \), is a paraconsistent relation on scheme \( \Sigma \), given that \( (R \cup S)^+ = R^+ \cup S^+ \) \( , (R \cup S)^- = R^- \cap S^- \) \( . \)

Complement. The complement of \( R \), denoted \( \neg R \), is a paraconsistent relation on scheme \( \Sigma \), given that \( \neg R^+ = R^- , \neg R^- = R^+ \)

Intersection. The intersection of \( R \) and \( S \), denoted \( R \cap S \), is a paraconsistent relation on scheme \( \Sigma \), given that \( (R \cap S)^+ = R^+ \cap S^+ \) \( , (R \cap S)^- = R^- \cup S^- \) \( . \)

Difference. The difference of \( R \) and \( S \), denoted \( R \setminus S \), is a paraconsistent relation on scheme \( \Sigma \), given that \( (R \setminus S)^+ = R^+ \setminus S^+ \) \( , (R \setminus S)^- = R^- \cup S^- \) \( . \)

Example 1. Let \( \{a, b, c\} \) be a common domain for all attribute names, and let \( R \) and \( S \) be the following paraconsistent relations on schemes \( \{X\} \) and \( \{X\} \) respectively:

\[ R^+ = \{(a), (b), (c)\} \]
\[ S^+ = \{(c), (b), (a)\} \]

\[ R \cup S \] is

\[ (R \cup S)^+ = \{(a), (b), (c)\} \]
\[ (R \cup S)^- = \{\} \]

\[ R \cap S \] is

\[ (R \cap S)^+ = \{(b)\} \]
\[ (R \cap S)^- = \{(a), (c)\} \]

\[ \neg R \] is

\[ \neg R^+ = \{(c)\} \]
\[ \neg R^- = \{(a), (b)\} \]
Relation Theoretic Operators. Let $\Sigma$ and $\Delta$ be relation schemes such that $\Sigma \subseteq \Delta$, and $R$ and $S$ be paracommon relations on schemes $\Sigma$ and $\Delta$.

Join. The join of $R$ and $S$, denoted $R \sqcup S$, is a paracommon relation on scheme $\Sigma \cup \Delta$ given that

$$
(R \sqcup S)^+ = (R^+ \Sigma \cup \Delta) \cup (S^- \Sigma \cup \Delta)
$$

Projection. The projection of $R$ onto $\Delta$, denoted $\pi_\Delta(R)$, is a paracommon relation on scheme $\Delta$ given that

$$
\pi_\Delta(R)^+ = \pi_\Delta(R^+ \Sigma \cup \Delta)
$$

Selection. Let $F$ be any logic formula involving attribute names in $\Sigma$, constant symbols, and any of these symbols $\{=, \neg, \land, \lor\}$. Then the selection of $R$ by $F$, denoted $\sigma_F(R)^+$, is a paracommon relation on scheme $\Sigma$, given that

$$
\sigma_F(R)^+ = \{t | t^\Sigma \subseteq R^+ \Sigma \cup \Delta \land F\}
$$

In the rest of the paper, relations mean paracommon relations. In order to find the QC model easily in our algorithm, we create a copy for a given relation. For any given relation $R$, the copy of $R$ is $R'$. Both $R$ and $R'$ are different relations with the same attributes and the same tuples. $R'$ is called an exact relation and $R'$ is called a copy relation. In addition to that, the replica of $R$ is $R$, where replica $R$ has the same name, the same tuples, and the same attributes. We assume that a relation and its replica should not appear in the same set, but it can appear in different sets. If two relations (a relation and its replica) appear in the same set, then we merge the tuples and write it as one relation.

Our main contributions of the paper start from the disjunctive relation.

3 Disjunctive Relation

Let a disjunctive relation scheme $\Sigma$ be a finite set of attribute sets, where for any attribute set $A \subseteq \Sigma$, $\text{dom}(a)$ is a non-empty domain of values for each $a \in A$. Let $\tau(\Sigma^2)$ denote the set of all tuples on $\Sigma^2$. A disjunctive relation, $\mathbf{DR}$, over the scheme $\Sigma^2$ consists of two components $(\mathbf{DR}^+, \mathbf{DR}^-)$, where $\mathbf{DR}^+ \subseteq \tau(\Sigma^2)$ and $\mathbf{DR}^- \subseteq \tau(\Sigma^2)$. $\mathbf{DR}^+$ is the component that consists of a set of tuples. While the tuples in $\mathbf{DR}^+$ typically represent the disjunction of facts, they also sometimes represent the conjunction of facts. At the same time, $\mathbf{DR}^-$ is the component that consists of a set of tuples. Each tuple in this component represents a conjunction of facts. In the case where the tuple is a singleton, both $\mathbf{DR}^+$ and $\mathbf{DR}^-$ have a definite fact that has neither disjunction or conjunction.

Let $T$ be a tuple in $\mathbf{DR}$, then for all $t \in T$, $\text{Att}(t)$ be an attribute set that represents the element in the tuple of the disjunctive relation $\mathbf{DR}$, and let $\text{Att}(R)$ be an attribute set that represents the relation $R$ over the scheme $\Sigma$.

The disjunctive relational model is very much different from the disjunctive database introduced by Molinaro et al. (Molinaro, Chomicki, and Marcinkowski 2009). The disjunctive database (Molinaro, Chomicki, and Marcinkowski 2009) is based on the relational model (not the paracommon relational model), whereas disjunctive relational model is based on the paracommon relational model. Moreover unlike the disjunctive relational model, disjunctive databases (Molinaro, Chomicki, and Marcinkowski 2009) are well-built for repairs. At the same time, the disjunctive relational model is also different from the relational model introduced by Sunderraman that can handle disjunction (Sunderraman 1997). In Sunderraman’s relation model for disjunction, all the disjuncts should have the same arity in the relation. However, in the disjunctive relational model, it can be different.

In the following, we define rename operators, mapping, and necessary functions, which all play a key role in constructing the QC model. In addition to that, we give an example that relates the usage of the operators, mapping, and the functions which help to comprehend the algorithm.

**Rename Operators.** Rename operators change the attributes for any relation. We define two rename operators: i) Attribute Rename, and ii) Copy Attribute Rename.
THEOREM 1. Let \( R \) be a relation over scheme \( \Sigma \) and \( \Sigma = \{ A_1 \ldots A_m, R.A_1 \ldots R.A_m \} \), then
\[
\Theta_{A_1 \ldots A_m \rightarrow R.A_1 \ldots R.A_m}(R)
\]
and
\[
\Theta_{R.A_1 \ldots R.A_m \rightarrow A_1 \ldots A_m}(R)
\]
This operator (\( \Theta \)) is used to maintain uniqueness of attributes between any two relations.

THEOREM 2. Let \( R \) be a relation over scheme \( \Sigma \) and \( \Sigma = \{ A_1 \ldots A_m \} \). Then
\[
\Theta_{A_1 \ldots A_m \rightarrow R.A_1 \ldots R.A_m}(R)
\]
and
\[
\Theta_{R.A_1 \ldots R.A_m \rightarrow A_1 \ldots A_m}(R)
\]
**Tuple Mapping to Disjunctive Relation.** The algebraic equivalent for disjunction (\( \lor \)) is union. So, we represent the disjunctive information in \( P \) as paraconsistent unions (\( \uparrow \)) of relations. But it is not very flexible to construct the QC model with paraconsistent unions (\( \uparrow \)) of relations. So, we map the information in relations to a disjunctive relation \( DR \). Let \( R_1 \ldots R_n \) be relations over schemes \( \Sigma_1 \ldots \Sigma_n \) where every \( \Sigma_i \subseteq \Sigma \) and \( 1 \leq i \leq n \). Then a set of attribute sets for any \( DR \) obtained from \( R_1 \lor \ldots \lor R_n \) is \( \{ \Sigma_1 \ldots \Sigma_n \} \). Now, we map the tuples of relations containing paraconsistent unions to a disjunctive relation. For each \( t \in T \), \( T \) is a tuple for any disjunctive relation (\( DR \)). Then \( t: \Sigma \rightarrow \bigcup_{A \in Att(R_i)} dom(A) \) such that \( t(A) \in dom(A) \) for every \( i \) in \( R_1 \lor \ldots \lor R_n \) where \( Att(t) = Att(R_i) \). Informally, a disjunctive relation can be considered a collection of relations that has unions. It is intuitive to map each disjunctive relation back to its base relations because every \( t \in T \) of any disjunctive relation represents the corresponding tuple in the relation \( R_i \).

In our approach, we need to find the name of the underlying relation for any given element in \( T \). Hence, the following definition:

**NRelation.** For any \( t \in T \) where \( T \) is a tuple for any disjunctive relation (DR) that is mapped from \( R_1 \lor \ldots \lor R_n \),
\[
NRelation(t) := \{ R_i | \text{Att}(t) = \text{Att}(R_i) \text{ for any } i \text{ in } R_1 \lor \ldots \lor R_n \}
\]

**NRelation** returns the corresponding relation name (either positive or negative) for the given \( t \) where \( t \in T \) and \( T \in DR \).

As we stated earlier, the positive component of disjunctive relations contains tuples that are typically disjunctive, but the positive component of disjunctive relations can be conjunctive as well. As we are specifically handling the inconsistencies associated with disjunction, we collect the tuples that are disjunctive.

**DISJ.** Let \( DR \) be a disjunctive relation that is mapped from \( R_1 \lor \ldots \lor R_n \). Then
\[
DISJ(DR) = \{ T | \forall T \in DR^+ \text{ such that } T \text{ is disjunctive} \}
\]

The following example is very specific, but helps to understand the algorithm clearly.

**Example 3.** Let \( R_1, R_2 \) and \( C \) be relations over schemes \( \{ X \}, \{ Y, Z \} \) and \( \{ X, Y, Z \} \) and domain for every attribute is \( \{ a, b, c \} \). Then, we have the following equation:
\[
(\bar{\pi}_{X,Y,Z}(R_1(\bar{X} \cup \bar{R}_2(\bar{Y},\bar{Z}))))(\bar{X},\bar{Y},\bar{Z}) = (\bar{\pi}_{X,Y,Z}(C(\bar{X},\bar{Y},\bar{Z})))^+(\bar{X},\bar{Y},\bar{Z})
\]

where \( C^+ = \{ \{ a, b, c \} \}, R_1^+ = \{ \{ b \} \text{ and } R_2^+ = \{ \{ a, c \}, \{ b, c \} \} \)

**Solution.** Before the tuples of \( C \) are distributed to \( R_1 \) and \( R_2 \), it is imperative to note that \( R_1 \) and \( R_2 \) contain definite tuples, which are not disjunctive (conjunctive). The first step is to map the definite tuples of \( R_1 \) and \( R_2 \) to a disjunctive relation. The definite tuples have no disjunction (conjunction) in any disjunctive relation. So, we rename the attributes (\( \Theta \)) of \( R_1 \) and \( R_2 \). Then we map the definite tuples to \( DR \).

In the rest of the paper, we differentiate positive and negative parts of a relation (disjunctive relation) with a double line in every relation (disjunctive relation) diagram.

\[
DR = \left\{ \begin{array}{l}
(R_1.X) & (R_2.Y, R_2.Z) \\
(a) & (b, c) \\
(b, c) & (b, c)
\end{array} \right\}
\]

The next step is to distribute the tuples from \( C \) to each individual relation in any union after applying \( \Theta \) to \( R_1 \) and \( R_2 \). It is necessary to apply \( \Theta \) before the distribution of tuples from \( C \) because we changed the attributes of \( R_1 \) and \( R_2 \) before we map the definite tuples.

\[
R_1 = \left\{ \begin{array}{l}
(\bar{X}) \\
(a) \end{array} \right\} \text{ and } R_2 = \left\{ \begin{array}{l}
(\bar{Y}, \bar{Z}) \\
(b, c) \\
(a, c) \\
(b, c)
\end{array} \right\}
\]

The next step is to again rename (\( \Theta \)) the attributes.

\[
R_1 = \left\{ \begin{array}{l}
(R_1.X) \\
(a) \end{array} \right\} \text{ and } R_2 = \left\{ \begin{array}{l}
(R_2.Y, R_2.Z) \\
(b, c) \\
(a, c) \\
(b, c)
\end{array} \right\}
\]

Then we map the newly added tuples of \( R_1 \lor \lnot R_2 \) to \( DR \).

\[
DR = \left\{ \begin{array}{l}
(R_1.X) & (R_2.Y, R_2.Z) \\
(a) \lor (b, c) \\
(b, c) & (a, c)
\end{array} \right\}
\]

To find the relation name for any given element in the tuple \( T \) where \( T \in DR \), we use \( NRelation \). So, \( NRelation((a, c)) = \lnot R_2 \).

In addition to that, \( DISJ(DR) \) is \( (a) \lor (b, c) \).

In the following section we introduce \( FOCUS_C \) and \( FOCUS_{SD} \), which are very essential for handling inconsistencies.

**4 Key Idea for QC logic**

It is very important to note that the paraconsistent relation portrays a belief system rather than a knowledge system. The key idea of QC logic is given by the resolution rule of inference, which computes the focused belief. If the assumptions are considered as beliefs for the resolution, then
the resolvent is called the focused belief. This ensures non-
trivial reasoning in QC logic. As an individual can be both
true and false for a given relation in the relational model,
we decouple the link during the model construction.
This is accomplished with the help of FOCUS_D and FOCUS_C.

**FOCUSD.** Let DR be a disjunctive relation on scheme Σ
and MR be a set of relations. Then

\[ \text{FOCUSD}(DR, MR) = \{ T \mid \forall t \in \text{DISJ}(DR) \land \exists t \in T \land \exists R \in MR \land \text{Att}(R) = \text{Att}(N\text{Relation}(t)) \land (N\text{Relation}(t) \text{ is positive } \land t \in R^+ \rightarrow (T = T \setminus t)) \lor (N\text{Relation}(t) \text{ is negative } \land t \in R^- \rightarrow (T = T \setminus t))) \} \]

As a special case, for a given tuple T where T ∈ DR+,
if FOCUSD removes every element t in tuple T, then we
convert the tuple T into a conjunction of the elements in
the tuple. This is similar to focus that we defined in the
Preliminaries section.

**CONJ.** Let DR be a disjunctive relation that is mapped
to \( R_1 \cup \ldots \cup R_n \). For any \( T \in DR \),

\[ \text{CONJ}(T) := \{ t_1 \land \ldots \land t_n \mid \forall t_i \in T \land n \leq |T| \} \]

Using CONJ, we define FOCUS C.

**FOCUS C.** Let DR be a disjunctive relation on scheme Σ
and MR be a set of relations. Then

\[ \text{FOCUSC}(DR, MR) = \{ \text{CONJ}(T) \mid \forall T \in DR^+ \land \forall t \in T \land \exists R \in MR \land \text{Att}(R) = \text{Att}(N\text{Relation}(t)) \land (N\text{Relation}(t) \text{ is positive } \land t \in R^+ \lor (N\text{Relation}(t) \text{ is negative } \land t \in R^-)) \} \]

FOCUSD removes any element t where \( t \in T \) and
\( T \in DR \), that satisfies the predicate of FOCUS D. Similarly,
FOCUS C introduces conjunction among every \( t \in DR \)
where \( T \in DR \), that satisfies the predicate of FOCUS C.
In any DR, any tuple T that contains conjunction should
never be affected by FOCUS D.

**Example 4.** Extending from Example 3. Let MR = \{ \( R_2 \) \}
where \( R_2^+ = \{(a,c),(b,c)\} \). \( R'_2 \) is called a copy relation.

**Solution.**

\[ MR = \{ R_2 \} \]

\[ R'_2 = \begin{pmatrix}
(a,c) \\
(b,c)
\end{pmatrix}
\]

We know that,

\[ DR = \begin{pmatrix}
\{R_1,X\} & \{R_2,Y,R_2,Z\} \\
(a) & (a,c) \\
(b) & (b,c)
\end{pmatrix}
\]

The attributes of \( R'_2 \) is different from \( R_1 \) and \( R_2 \).
Apply \( \Theta(R'_2) \), \( \Theta(\Theta(R_1)) \) and \( \Theta(\Theta(R_2)) \). So,
FOCUS C (FOCUS D) can compare the attributes and perform
necessary actions on T.

Now we apply focus to DR,

\[ DR = \text{FOCUS \text{\( D \)}} (DR, MR) \]

In this case, in DR, the underlying relation for \( (b,c) \) is
\( \neg R_1 \) but \( (b,c) \) lies in the positive part of \( R'_2 \) in MR.
Therefore, FOCUS D removes it.

\[ \text{DR} = \begin{pmatrix}
\{R_1,X\} & \{R_2,Y,R_2,Z\} \\
(a) & (a,c) \\
(b) & (b,c)
\end{pmatrix}
\]

Apply \( \Theta(R'_2) \), \( \Theta(\Theta(R_1)) \) and \( \Theta(\Theta(R_2)) \). These operations revert the relation back to its old attribute names. To reiterate, DR* contains tuples which in turn can contain disjunction. From the base DR, multiple DR can be obtained by applying disjunction in tuples. Each newly created DR from the base DR should not lose any tuple set; otherwise, it leads to incorrect models. The following definition addresses the issue.

**Proper Disjunctive Relation (PDR).** Let DR be a base
disjunctive relation. A proper disjunctive relation is a set,
which contains all disjunctive relations that can be formed
from DR by applying disjunction in tuples. Concretely,
for every disjunctive relation (DRi), which is obtained from
DR by applying disjunction, \( \tau(DR^\uparrow_i) = \tau(DR^\uparrow) \)
where \( 1 \leq i \leq (2^n - 1) \tau(DR^\uparrow) \) such DRi is a PDR.

**Example 5.** Continuing from Example 4.

**Solution.** The next step is to create a set of proper disjunctive relation from DR.

\[ PDR = \{ PDR^1 \} \]

\[ PDR^1 = \begin{pmatrix}
\{R_1,X\} & \{R_2,Y,R_2,Z\} \\
(a) & (a,c) \\
(b) & (b,c)
\end{pmatrix}
\]

The size of PDR is 1. Correspondingly, there should be one replica of base relations.
\[ \{(\pi_{\{X,Y,Z\}}(R_1 \cup \neg R_2 \cup R_2,Y,R_2,Z))[X,Y,Z]\} \]

For every p in PDR, reverse map tuples to a set of base relations.

\[ R_1 = \begin{pmatrix}
\{R_1,X\} & \{R_2,Y,R_2,Z\} \\
(a) & (a,c) \\
(b) & (b,c)
\end{pmatrix} \]

Finally, rename (\( \Theta \)) each attribute name of every relation
back to its old name. Hence, \( R_1 \) attribute is \( X \) and \( R_2 \)
attribute is \( Y, Z \).

To individualize the relation, we have the following
definition.

**Relationalize.** Let \( R_1 \cup \ldots \cup R_n \) and \( R_1, \ldots, R_n \) be relations
on scheme \( \Sigma \).

\[ \text{Relationalize}(\pi_{\{X\}}(R_1 \cup \ldots \cup R_n)[\Sigma]) = \{ R_1, \ldots, R_n \} \]

The relational operator removes the unions amongelations and the projection for it. By doing so, the operator
produces a set of relations. If there is a select operation associated
with the expression, then apply the operation before
Relationalize is applied. Relationalize is in accordance to
Lits, which is one of the key operators for finding the QC
model (Zhang, Lin, and Ren 2009).

**Example 6.** Continuing from Example 5.

**Solution.** Relationalize(\( \pi_{\{X,Y\}}(R_1 \cup \neg R_2)[Y,X] \)) = \{ R_1, R_2 \}
\[
R_1 = \{X\} \quad \text{and} \quad -R_2 = \{(Y, Z)\} \quad \text{or} \quad R_2 = \{Y, Z\} \quad \text{(a,c)} \\
\]

Now \(R_1\) and \(R_2\) are called focused relations because \(R_1\) and \(R_2\) have no inconsistency.

During QC model construction, we encounter a set of redundant relation sets. In order to remove it, we define the following.

Minimize. Let \(\{ R_1 \ldots R_m \} \) and \(\{ R_{j1} \ldots R_{jn} \} \) be two sets of relations where \(m \leq n\).

\[
\text{Minimize}(\{ \{ R_1 \ldots R_m \} \mid R_{i1} = R_{j1} \land \text{Att}(R_{i1}) = \text{Att}(R_{j1}) \land \tau(R_{i1}) = \tau(R_{j1}) \} : = \{ \{ R_1 \ldots R_m \} \mid R_{i1} = R_{j1} \land \text{Att}(R_{i1}) = \text{Att}(R_{j1}) \land \\
\tau(R_{i1}) = \tau(R_{j1}) \} \quad \text{such that} \forall i, 1 \leq i \leq m \land \exists j, 1 \leq j \leq n)
\]

By using the definitions and operators in Section 3 and Section 4, we propose an algorithm in the following section.

5 QC Model for Positive Extended Disjunctive Deductive Databases

By using the algebra of the relational model, we present a bottom-up method for constructing the QC model for the positive extended disjunctive deductive database. The algorithm that we present in this section is an extension of the algorithm proposed by Bagai and Sunderraman (Bagai and Sunderraman 1995). The reader is referred to request to QC logic programs (Zhang, Lin, and Ren 2009) and QC logic (Hunter 2000). The QC model’s construction involves two steps. The first step is to convert \(P\) into a set of relation definitions for the predicate symbols occurring in \(P\). These definitions are of the form

\[
U_r = D_{U_r},
\]

where \(U_r\) is the paraconsistent union of the disjunctive head predicate symbols of \(P\), and \(D_{U_r}\) is an algebraic expression involving predicate symbols of \(P\). Here \(r\) refers to the equation number, \(1 \leq r \leq N\), where \(N\) refers to a total number of equations. The second step is to iteratively evaluate the expressions in these definitions to incrementally construct the relations associated with the predicate symbols. The first step is called SERALIZE and the second step is called Model Construction.

Algorithm. SERALIZE

Input. A positive extended disjunctive deductive database clause \(l_0 \vee \cdots \vee l_m \leftarrow l_{m+1} \ldots l_n\). For any \(i, 0 \leq i \leq m, l_i\) is either of the form \(p_i(A_{i1} \ldots A_{ik})\) or \(\neg p_i(A_{i1} \ldots A_{ik})\). Let \(V_i\) be the set of all variables occurring in \(l_i\).

Output. An algebraic expression involving paraconsistent relations.

Method. The expression is constructed by the following steps:

1. For each argument \(A_{ij}\) of literal \(l_i\), construct argument \(B_{ij}\) and condition \(C_{ij}\) as follows:
   (a) If \(A_{ij}\) is a constant \(a\), then \(B_{ij}\) is any brand new variable and \(C_{ij} = a\).
   (b) If \(A_{ij}\) is a variable, such that for each \(k, 1 \leq k < j, A_{ik} \neq A_{ij}\), then \(B_{ij}\) is \(A_{ij}\) and \(C_{ij}\) is true.
   (c) If \(A_{ij}\) is a variable, such that for some \(k, 1 \leq k < j, A_{ik} = A_{ij}\), then \(B_{ij}\) is a brand new variable and \(C_{ij}\) is \(A_{ij} = B_{ij}\).

2. Let \(\hat{t}_i\) be the atom \(p_i(B_{i1} \ldots B_{ik})\), and \(F_i\) be the conjunction \(C_{i1} \land \cdots \land C_{ik}\). If \(l_i\) is a positive literal, then \(Q_i\) is the expression \(\pi_{\hat{t}_i}(\hat{Q}_i(\hat{t}_i))\). Otherwise, let \(Q_i\) be the expression \(\pi_{\hat{t}_i}(\hat{Q}_i(\hat{t}_i))\).

As a syntactic optimisation, if all conjuncts of \(F_i\) are true (i.e. all arguments of \(l_i\) are distinct variables), then both \(\hat{Q}_i\) and \(\pi_{\hat{t}_i}\) are reduced to identity operations, and are hence dropped from the expression.

3. Let \(U\) be the union (\(\cup\)) of the \(Q_i\)'s thus obtained, \(0 \leq i \leq n\). The output expression is \((\hat{Q}_i(\pi_{\hat{t}_i}(\hat{Q}_i(\hat{t}_i))))[B_{01} \ldots B_{nm}]\) where \(DV\) is the set of distinct variables occurring in all \(l_i\).

4. Let \(E\) be the natural join (\(\bowtie\)) of the \(Q_i\)'s thus obtained, \(n+1 \leq i \leq m\). The output expression is \((\hat{Q}_i(\pi_{\hat{t}_i}(\hat{Q}_i(\hat{t}_i))))[B_{01} \ldots B_{nm}]\). As in step 2, if all conjuncts are true, then \(\hat{Q}_i\) is dropped from the output expression.

From the algebraic expression of the algorithm, we construct a system of equations.

For any positive extended disjunctive deductive database \(P\), \(\text{EQN}(P)\) is a set of all equations of the form \(U_r = D_{U_r}\), where \(U_r\) is a union of the head predicate symbols of \(P\), and \(D_{U_r}\) is the union \(\cup\) of all expressions obtained by the algorithm \(\text{SERALIZE}\) for clauses in \(P\) with the same \(U_r\) in their head. If all literals in the head are the same for any two rules, then \(U_r\) is the same for those two rules.

The final step is then to construct the model by incrementally constructing the relation values in \(P\). For any positive extended disjunctive deductive database, \(P_E\) are the non-disjunctive-facts (clauses in \(P\) without bodies), and \(P_B\) are the disjunctive rules (clauses in \(P\) with bodies). \(P_E\) refers to a set of all ground instances of clauses in \(P_E\). Then, \(P_I = P_E \cup P_B\).

The following algorithm finds the QC model for \(P\).

Algorithm. Model Construction

Input. A positive extended disjunctive deductive database \((P)\)

Output. Minimal QC Model for \(P\).

Method. The values are computed by the following steps.

1. (Initialization)
   (a) Compute \(\text{EQN}(P)\) using the algorithm \(\text{SERALIZE}\) for each clause in \(P\).
   (b) \(\text{SModel}=\emptyset\). For each predicate symbol \(p\) in \(P_E\), set \(p^+=\{a_1 \ldots a_k | p(a_1 \ldots a_k), a_1 \ldots a_k \in P_E\}\), and \(p^- = \emptyset\) or \(p^- = \{a_1 \ldots a_k | \neg p(a_1 \ldots a_k), a_1 \ldots a_k \in P_E\}\) and \(p^+ = \emptyset\)

   \(\text{SModel}=p\)

   End for.

2. (Rule Application)
   (a) \(\text{DModel}=\emptyset\).

   For every \(SModel\), create copies of the relations in \(SModel\) and replace the \(SModel\) with the copies.

   (b) For every equation \(r\) of the form \(U_r = D_{U_r}\), create \(DR_r\) and insert the tuples from the copies in \(SModel\) into the corresponding exact relation in the equation \(r\). Then map the definite tuples for the relations in \(U_r\) to
Compute the expression $D_{U_r}$ and set the relations in $U_r$ with $D_{U_r}'$.  

(c) Map the newly added tuples of $U_r$ to $D_{U_r}$. Apply $\Theta$ and $\Omega$ to every relation in $U_r$. Also apply $\Theta$ to every relation in SMModel. Then

$$DR_r = FOCUS_C(DR_r, SMModel)$$

$$DR_r = FOCUS_D(DR_r, SMModel)$$

Repeat $FOCUS_D$ until there is no change in $DR_r$. When there is no change is $DR_r$, apply $\Theta$ to every relation in SMModel and apply $\Omega$ and $\Theta$ to every relation in $U_r$.

(d) Create a set of proper disjunctive relations ($PDR_r$) from the focused $DR_r$.

(e) Delete all tuples for the relations in $U_r$ and create multiple replicas of $U_r$, which is denoted by the set $C_r$, where $|C_r| = |PDR_r|$.

(f) Re-map each $p$ in $PDR_r$ to $C$ where $C \in C_r$.

For every $C \in C_r$,

$C = Relationalize(C)$

$\pi^* C_r$ contains a collection of sets of relations. */

$DMModel = DMModel \cup C_r$

$\pi^* Merging$ relations of every equation */

(g) Once all equations are evaluated for the current SMModel, perform the following: i) for every $M \in DMModel and for every exact relation for SMModel that is not in $M$, create the exact relation in $M$, and ii) for every $M \in DMModel and for every exact relation for SMModel that is in $M$, insert the tuples from the copy relation in SMModel into the exact relation. Then add $DMModel to TempModel$.

(h) Once every SMModel is applied, start from step 2 (a) with SMModel=$Minimize (TempModel)$ and stop when there is no change in SMModel.

3. Minimal QC Model: Pick one (many) set (s) in SMModel whose sum of the size of all relations in the set (s) is (are) minimal.

It is very intuitive from the algorithm that if the computation of $D_{U_r}$ is empty for any SMModel, then discard the SMModel. We found that the algorithm should be extended a little to accommodate disjunctive facts, duplicate variables in disjunctive literals, and constants in disjunctive literals.

The following example shows how the algorithm works. In the example, to show the difference between any two sets, we superscript the set with a number.

**Example 7.** Let $P$ be a positive extended disjunctive deductive database. It has the following facts and rules:

$$r(a,c), p(a), p(c), \neg f(a,b), s(c)$$

$w(X) \lor g(X) \lor \neg p(X) \leftarrow r(X, Y), s(Y)$

$w(X) \lor g(X) \lor \neg p(X) \leftarrow f(X, Y)$

**Solution.** By step 1 (a) in initialization, $w(X) \lor g(X) \lor \neg p(X) \leftarrow r(X, Y), s(Y)$ is serialized to

$$(\pi_X(w(X) \cup g(X) \cup \neg p(X)))[X] = (\pi_X(r(X, Y) \lor s(Y)))^+[X]$$

and $w(X) \lor g(X) \lor \neg p(X) \leftarrow f(X, Y)$ is serialized to

$$(\pi_X(w(X) \cup g(X) \cup \neg p(X)))[X] = (\pi_X(f(X, Y)))^+[X]$$

Both equations that are obtained after serialization have the same left-hand side expression. So, it is written as one equation (as show in (1)). $EQN(P1) = r.$

We skip a step (2 (d)) here. After relationalizing the set of relations (step 2 (f)), we write:

$$\{w.X \}, \{g.X \}, \{p.X \}$$

$$\{X, Y \}$$

$$\{a, c \}$$

After step 2 (a), $SMModel = \{\{r', p', s', f'\} (COPIES)\}$ where

$$r' = \{X, Y \}$$

$$p' = \{Y \}$$

$$s' = \{Y \}$$

$$f' = \{a, b \}$$

In step 2 (b), there is only one SMModel and an equation. It is necessary to insert the tuples from the copies in SMModel to the corresponding relations in the equation. $DMModel = \emptyset$. Then map the definite tuples to $DR_1$ for the current SMModel.

$$DR_1 = \{w.X \}, \{g.X \}, \{p.X \}$$

Compute the equation and assign it to $U_1$. Map the newly added (disjunctive) tuples to $DR_2$.

$$DR_1 =$$

$$\{w.X \}, \{g.X \}, \{p.X \}$$

By step 2 (c), $DR_2 = FOCUS_D(DR_1, SMModel)$

$$DR_2 = \{w.X \}, \{g.X \}, \{p.X \}$$

By step 2 (d), $PDR_1 = \{PDR_1^1, PDR_1^2, PDR_1^3\}$

$$PDR_1^1 = \{w.X \}, \{g.X \}, \{p.X \}$$

$$PDR_1^2 = \{w.X \}, \{g.X \}, \{p.X \}$$

$$PDR_1^3 = \{w.X \}, \{g.X \}, \{p.X \}$$

Map every $p$ in $PDR_1$ back to a set of base relations. We skip a step (2 (d)) here. After relationalizing the set of relations (step 2 (f)), we write:

$$C_1 = \{\{w, p\}^1, \{g, p\}^2, \{w, g, p\}^3\}$$
In this paper, we proposed an algorithm to find the QC model for any positive extended disjunctive deductive database. We also introduced the disjunctive relational model to represent the relations containing paraconsistent unions. The algorithm that we presented here is based on the algorithm that is used to compute the well-founded model for general deductive databases by using the relational model (Bagai and Sunderraman 1996a). In query-intensive applications, this precomputation of the model enables efficient processing of subsequent queries. Though we find the model for any given positive extended disjunctive deductive database, the algorithm does not find models for the databases with recursions and constraints. One direction of future work could be expanding the algorithm to allow recursions and constraints. Moreover, the model that we construct is too strong; it causes disjunction introduction to fail, but it is supported by QC logic. To compute the QC entailment, it is necessary to have both weak and strong models. So another direction of future work could be finding the weak models for the same program so that QC entailment could be achieved. The creation of many proper disjunctive databases is expensive, given the QC logic model computation, and are probably not worth the extra computation. We notice that we have not stated or proven the complexities of the algorithm, and we have also left it for future work.

6 Conclusion

References


Merging Incommensurable Possibilistic DL-Lite Assertional Bases

Salem Benferhat and Zied Bouraoui and Sylvain Lagrue
Univ Lille Nord de France, F-59000 Lille, France
UArtois, CRIL - CNRS UMR 8188, F-62300 Lens, France
{benferhat.bouraoui,lagrue}@cril.fr

Julien Rossit
Univ Paris Descartes, LIPADE-France
julien.rossit@parisdescartes.fr

Abstract
This short paper studies the problem of merging of different independent data sources linked to a lightweight ontology under the incommensurability assumption. In general, data are often provided by several and potentially conflicting sources of information having different levels of priority. To encode prioritized assertional bases, we use possibilistic DL-Lite logic. We investigate an egalitarian merging strategy that minimize dissatisfaction between the source involved in the merging process. We provide a safe way to merge incommensurable possibilistic DL-Lite assertional bases using the notion of compatible scales.

Introduction
Description Logics (DLs) are a well-known family of logic-based formalisms used to represent knowledge of a particular domain and make it available for reasoning. DLs are recognized as powerful frameworks that support ontologies. A DL knowledge base is formed by a terminological base, called TBox, and an assertional base, called ABox. The TBox contains intentional (or generic) knowledge of the application domain whereas the ABox stores data (individuals or constants) that instantiate terminological knowledge.

In the last years, there has been an increasingly interest in Ontology-based Data Access (OBDA) that studies how to query a set of data linked to a unified TBox (ontolgy). A lot of attention was given to DL-Lite, a family of lightweight DLs specifically dedicated for applications that use huge volumes of data, in which query answering is the most important reasoning task. DL-Lite offers a very low computational complexity for the reasoning process.

In many applications, data are often provided by several and potentially conflicting sources having different reliability levels. Moreover, a given source may provide different sets of uncertain data with different confidence levels. In such situation, there are two main attitudes that may be followed: the first attitude consists first in gathering sets of assertions provided by each sources which gives generally an inconsistent (prioritized or flat) assertional base and then coping with inconsistencies when performing inference using different inconsistency-tolerant inference strategies (e.g. (Lembo et al. 2010; Bienvenu 2012; Bienvenu and Rosati 2013)). The second one consists in merging the assertional bases using some aggregation strategies.

Knowledge bases merging or belief merging (e.g. (Bloch et al. 2001; Konieczny and Pérez 2002)), is a problem largely studied within the propositional logic setting. It focuses on aggregating pieces of information issued from distinct, and may be conflicting or inconsistent, sources in order to obtain a unified point of view by taking advantage of pieces of information provided by each source. Several merging approaches have been proposed which depend on the nature and the representation of knowledge such as merging propositional knowledge bases (e.g. (Konieczny and Pérez 2002)), prioritized knowledge bases (e.g. (Delgrande, Dubois, and Lang 2006)) or weighted logical knowledge bases (e.g. (Benferhat, Dubois, and Prade 1997)). Recently, some works (e.g. (Noy and Musen 2000; Kotis, Vouros, and Stergiou 2006; Moguillansky and Falappa 2007; Cóbe, Resina, and Wassermann 2013)), have proposed to merge ontologies.

In (Benferhat, Bouraoui, and Loukil 2013), the authors study the counterpart of the min-based merging (Benferhat, Dubois, and Prade 1997) when uncertain pieces of information are represented by a possibilistic DL-Lite knowledge base. The min-based merging operator, well-known as idempotent conjunctive operator, is suitable when sources are assumed to be dependent. In (Benferhat et al. 2014), a min-based merging operator based on conflict resolution is proposed to merge uncertain DL-Lite assertional bases linked to the same terminological base (i.e. a TBox) seen as merging integrity constraints.

This paper goes one step further by extending the min-based possibilistic merging operator in the case where uncertainty scales used by different sources are incommensurable. We will follow the idea of egalitarian merging operator proposed in (Benferhat, Lagrue, and Rossit 2007) based on the concept of comparable scales. In this paper, we assume that the TBox is coherent and fully certain and only assertional facts (ABoxes) issued from distinct sources may be somewhat certain.

A compatible scale is a re-assignment of certainty degrees to assertional facts such that the initial plausibility ordering inside each ABox (source) is preserved. We show, in particular, that merging a set of ABoxes under incommensurable assumption comes down to apply min-based possi-
bilistic merging of ABox with respect to each compatible scale.

The rest of the paper is organized as follows: Section 2 gives brief preliminaries on DL-Lite. Section 3 recalls DL-Liteπ an extension of DL-Lite within a possibility theory setting. Section 4 investigates min-based merging of multiple and uncertain DL-Lite ABoxes under the incommensurability assumption. Section 5 concludes the paper.

A brief refresh on DL-Lite
For the sake of simplicity, we only present DL-Litecore the core fragment of all the DL-Lite family (Calvanese et al. 2007) and we will simply use DL-Lite instead of DL-Litecore. However, results of this paper are valid for DL-LiteR and DL-LiteF, two important members of the DL-Lite family. The DL-Lite language is defined as follows:

\[ R \rightarrow P | P^\neg | B \rightarrow A | \exists R | C \rightarrow B \neg B \]

where \( A \) is an atomic concept, \( P \) is an atomic role and \( P^{-} \) is the inverse of \( P \). \( B \) (resp. \( C \)) is called basic (resp. complex) concept and role \( R \) is called basic role. A DL-Lite knowledge base (knowledge base) is a pair \( K=(\mathcal{T}, \mathcal{A}) \) where \( \mathcal{T} \) is the TBox and \( \mathcal{A} \) is the ABox. The TBox \( \mathcal{T} \) includes a finite set of inclusion assertions of the form \( B \subseteq C \) where \( B \) and \( C \) are concepts. The ABox \( \mathcal{A} \) contains a finite set of assertions on atomic concepts and roles of the form \( A(a) \) and \( P(a,b) \) where \( a \) and \( b \) are two individuals.

The semantics of a DL-Lite knowledge base is given in term of interpretations. An interpretation \( I=(\Delta^2, \mathcal{I}) \) consists of a non-empty domain \( \Delta^2 \) and an interpretation function \( \mathcal{I} \) that maps each individual \( a \) to \( a^2 \in \Delta^2 \), each \( A \) to \( A^2 \subseteq \Delta^2 \) and each role \( P \) to \( P^2 \subseteq \Delta^2 \times \Delta^2 \). Furthermore, the interpretation function \( \mathcal{I} \) is extended in a straightforward way for complex concepts and roles: \( (\neg B)^2 = \Delta^2 \setminus B^2 \), \( (P^{-})^2 = \{(y,x) | (x,y) \in P^2 \} \) and \( (\exists R)^2 = \{x \mid \exists y \text{ s.t. } (x,y) \in R^2 \} \). An interpretation \( I \) is said to be a model of a concept inclusion axiom, denoted by \( I \models B \subseteq C \), if \( B^2 \subseteq C^2 \). Similarly, we say that \( I \) satisfies a concept (resp. role) assertion, denoted by \( I \models A(a) \) (resp. \( I \models P(a,b) \)), iff \( a^2 \in A^2 \) (resp. \( (a^2, b^2) \in P^2 \)).

An interpretation \( I \) is said to be a model of a knowledge base \( K=(\mathcal{T}, \mathcal{A}) \), denoted by \( I \models K \), iff \( I \models \mathcal{T} \) and \( I \models \mathcal{A} \) where \( I \models \mathcal{T} \) (resp. \( I \models \mathcal{A} \)) means that \( I \) is a model of all axioms in \( \mathcal{T} \) (resp. \( \mathcal{A} \)). A knowledge base \( K \) is said to be consistent if it admits at least one model, otherwise \( K \) is said to be inconsistent. A DL-Lite TBox \( \mathcal{T} \) is said to be coherent if there exists at least a concept \( C \) such that for each interpretation \( I \) which is a model of \( \mathcal{T} \), we have \( C^2 \neq \emptyset \). Note that within a DL-Lite setting, the inconsistency problem is always defined with respect to some ABox since a TBox may be incoherent but never inconsistent.

Possibilistic DL-Lite
In this section, we recall the main notions of possibilistic DL-Lite framework (Benferhat and Bouraoui 2013), denoted by DL-Liteπ, as an adaptation of DL-Lite within a possibility theory setting (Dubois and Prade 1988). DL-Liteπ provides an excellent mechanism to deal with uncertainty and to ensure reasoning under inconsistency while keeping a computational complexity identical to the one used in standard DL-Lite.

Possibility Distribution over DL-Lite Interpretation
Let \( \Omega \) be a universe of discourse composed by a set of DL-Lite interpretations \( \mathcal{I}(\Delta, \mathcal{I}) \in \Omega \). The semantic counterpart of a DL-Liteπ is given by a possibility distribution, denoted by \( \pi \), which is a mapping from \( \Omega \) to the unit interval \([0,1]\) that assigns to each interpretation \( \mathcal{I} \in \Omega \) a possibility degree \( \pi(\mathcal{I}) \in [0,1] \) that represents its compatibility or consistency with respect to the set of available knowledge. When \( \pi(\mathcal{I})=0 \), we say that \( I \) is impossible and it is fully inconsistent with the set of available knowledge, whereas when \( \pi(\mathcal{I})=1 \), we say that \( I \) is totally possible and it is fully consistent with the set of available knowledge. For two interpretations \( \mathcal{I} \) and \( \mathcal{I}' \), when \( \pi(\mathcal{I})>\pi(\mathcal{I}') \) we say that \( I \) is more consistent or more preferred than \( I' \) w.r.t available knowledge. Lastly, \( \pi \) is said to be normalized if there exists at least one totally possible interpretation, namely \( \exists \mathcal{I} \in \Omega, \pi(\mathcal{I})=1 \), otherwise, we say that \( \pi \) is sub-normalized. The concept of sub-normalization reflects the presence of conflicts in the set of available information.

Given a possibility distribution \( \pi \) defined on a set of interpretations \( \Omega \), one can define two measures on a DL-Lite axiom \( \varphi \). A possibility measure \( \Pi(\varphi)=\max_{\mathcal{T} \in \Omega} \{ \pi(\mathcal{I}) : I \not\models \varphi \} \) that evaluates to what extent an axiom \( \varphi \) is compatible with the available knowledge encoded by \( \pi \) and a necessity measure \( \ni(\varphi)=1-\max_{\mathcal{T} \in \Omega} \{ \pi(\mathcal{I}) : I \models \varphi \} \) that evaluates to what extent \( \varphi \) is certainly entailed from available knowledge encoded by \( \pi \).

DL-Liteπ Knowledge Base
Let \( \mathcal{L} \) be a DL-Lite description language, a DL-Liteπ knowledge base is a set of possibilistic axioms of the form \( \langle \varphi, W(\varphi) \rangle \) where \( \varphi \) is an axiom expressed in \( \mathcal{L} \) and \( W(\varphi) \in [0,1] \) is the degree of certainty/priority of \( \varphi \). Namely, a DL-Liteπ knowledge base \( K \) is such that \( K=\{ \langle \varphi_i, W(\varphi_i) \rangle : i=1,...,n \} \). Only somewhat certain information are explicitly represented in a DL-Liteπ knowledge base. Namely, axioms with a null weight \( W(\varphi_i)=0 \) are not explicitly represented in the knowledge base. The weighted axiom \( \langle \varphi, W(\varphi) \rangle \) means that the certainty degree of \( \varphi \) is at least equal to \( W(\varphi) \). A DL-Liteπ knowledge base \( K \) will also be represented by a couple \( K=(\mathcal{T}, \mathcal{A}) \) where both elements in \( \mathcal{T} \) and \( \mathcal{A} \) may be uncertain. It is important to note that, if we consider all \( W(\varphi_i)=1 \) then we found a classical DL-Lite knowledge base: \( K^*=(\pi(\varphi_i, W(\varphi_i)) \in K) \).

Given \( K=(\mathcal{T}, \mathcal{A}) \) a DL-Liteπ knowledge base, we define the \( \alpha \)-cut of \( K \) (resp. \( \mathcal{T} \) and \( \mathcal{A} \)), denoted by \( K_{\geq \alpha} \) (resp. \( T_{\geq \alpha} \) and \( A_{\geq \alpha} \)), the subbase of \( K \) (resp. \( \mathcal{T} \) and \( \mathcal{A} \)) composed of axioms having weights at least greater than \( \alpha \).

We say that \( K \) is consistent if the standard knowledge base obtained from \( K \) by ignoring the weights associated with axioms is consistent. In case of inconsistency, we attach to \( K \) an inconsistency degree. The inconsistency degree of a DL-Liteπ knowledge base \( K \), denoted by \( \text{Inc}(K) \), is syntacti-
cally defined as follow: \( \text{Inc}(K) = \max\{W(\varphi_i) : K \cup W(\varphi_i) \text{ is inconsistent}\} \).

Given a DL-Liteπ knowledge base \( K \), one can associate to it a joint possibility distribution, denoted by \( \pi_K \), defined over the set of all interpretations \( \mathcal{I} = \Delta, \mathcal{I} \) by associating to each interpretation its level of consistency with the set of available knowledge, that is, with \( K \). Namely:

**Definition 1.** The possibility distribution induced from a DL-Liteπ knowledge base \( K \) is defined as follows: \( \forall \mathcal{I} \in \mathcal{V} : \pi_K(\mathcal{I}) = \begin{cases} 1 & \text{if } \forall \varphi_i. W(\varphi_i) \in K, \mathcal{I} \models \varphi_i \\ 1 - \max\{W(\varphi_i) : (\varphi_i, W(\varphi_i)) \in K, \mathcal{I} \not\models \varphi_i\} & \text{otherwise} \end{cases} \)

A DL-Liteπ knowledge base \( K \) is said to be inconsistent if its joint possibility distribution \( \pi_K \) is normalized. If not, \( K \) is said to be inconsistent and its inconsistency degree is defined semantically as follow: \( \text{Inc}(K) = 1 - \max\{\pi_K(\mathcal{I})\} \).

It was shown in (Benferhat and Bourrouilh 2013) that computing the inconsistency degree of a DL-Liteπ knowledge base comes from the extension of the algorithm presented in (Calvanese et al. 2007) by modifying it to query for individuals with a given certainty degree.

**Example 1.** Let \( K = (T, A) \) be a DL-Liteπ knowledge base where \( T = \{\{A \cup B, 1\}, \{B \cup \neg C, 9\}\} \) and \( A = \{\{A(a), \ldots, B\}, \{C(b), \ldots\}\} \). The possibility distribution \( \pi_K \) associated to \( K \) is computed using Definition 1 as follows where \( \Delta = \{a, b\} \):

<table>
<thead>
<tr>
<th>( I )</th>
<th>( \Delta )</th>
<th>( \pi_K(I) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_1 )</td>
<td>( A = {a}, B = {}, C = {} )</td>
<td>0</td>
</tr>
<tr>
<td>( I_2 )</td>
<td>( A = {a}, B = {a}, C = {} )</td>
<td>1</td>
</tr>
<tr>
<td>( I_3 )</td>
<td>( A = {}, B = {}, C = {} )</td>
<td>.4</td>
</tr>
<tr>
<td>( I_4 )</td>
<td>( A = {a, b}, B = {a, b}, C = {} )</td>
<td>.5</td>
</tr>
</tbody>
</table>

Table 1: Example of a possibility distribution induced from a DL-Liteπ knowledge base

One can observe that \( \pi_K(I_2) = 1 \) meaning that \( \pi_K \) is normalized, and thus, \( K \) is consistent.

**Fusion-based on compatible scalings**

This section studies min-based probabilistic merging operator in the case where uncertainty scales used by the different sources are incommensurable. Throughout this section, we assume that the TBox is coherent and fully certain and only assertional facts (ABoxes) may be somewhat certain. We first present merging using min-based operator of DL-Lite assertional bases under incommensurability assumption.

**Merging using the min-based operator**

Let \( A = \{A_1, \ldots, A_n\} \) be a set of \( n \) uncertain ABoxes issued from \( n \) distinct sources, and let \( T \) be a DL-Lite TBox representing the integrity constraints to be satisfied. Let us assume that \( \pi_1, \ldots, \pi_n \) are possibility distributions provided by \( n \) sources of information that share the same domain of interpretations (namely \( \Delta_T = \ldots = \Delta_T^\pi \)), and that all possibility distributions use the same scale to represent uncertainty. We suppose that each ABox is consistent with \( T \), namely each possibility distribution \( \pi_i \) that encodes \( K_i = (T, A_i) \) is normalized. For the sake of simplicity, we use \( \pi_{A_i} \) instead of \( \pi_K \) to denote the possibility distribution associated to each \( K_i = (T, A_i) \).

Given \( n \) commensurable ABoxes, merging aims to compute \( \Delta_T(A) \), an ABox representing the result of the fusion of these ABoxes. In the literature, different methods for merging have been proposed. In this section, we perform merging of \( A_1, \ldots, A_n \) a set of ABoxes with respect to a TBox \( T \) using min-based merging operator proposed to aggregate DL-Liteπ knowledge bases. This operator is a direct extension of the well-known idempotent conjunctive operator (e.g. (Benferhat, Dubois, and Prade 1997)) within possibilistic DL-Lite setting. It is recommended when distinct sources that provide information are assumed to be dependent.

We first introduce the notion of profile associated with an interpretation \( I \), denoted by \( \nu_A(I) \), and defined by

\[
\nu_A(I) = \langle \pi_{A_1}(I), \ldots, \pi_{A_n}(I) \rangle.
\]

Namely, \( \nu_A(I) \) represents the possibility values of an interpretation \( I \) with respect to each source.

From a semantics point of view, the result of merging is a possibility distribution \( \Delta_T(A) \) obtained using two steps: i) the possibility degrees \( \pi_{A_i}(I) \)'s are first combined with a merging operator (here we use the minimum operator), and the interpretations with height degrees are kept. This leads to define an order relation, denoted by \( <_{\text{min}} \), between interpretations as follows: an interpretation \( I \) is preferred to another interpretation \( I' \) if the minimum element of the profile of \( I \) is higher than the minimum element of the profile of \( I' \). More formally:

**Definition 2** (Definition of \( <_{\text{min}} \)). Let \( A = \{A_1, \ldots, A_n\} \) be a set of ABoxes linked to a TBox \( T \). Let \( I \) and \( I' \) be two interpretations and \( \nu_A(I), \nu_A(I') \) be their associated profiles. Then:

\[
I <_{\text{min}} I' \iff \text{Min}(\nu_A(I)) > \text{Min}(\nu_A(I'))
\]

where

\[
\text{Min}(\nu_A(I)) = \text{Min}\{\pi_{A_i}(I) : i \in \{1, \ldots, n\}\}.
\]

The result of the merging \( \Delta_T^{\text{min}}(A) \) is a DL-Liteπ knowledge base whose models are interpretations which are models of a constraint \( T \) and which are maximal with respect to \( <_{\text{min}} \). More formally:

**Definition 3** (Min-based merging operator). Let \( A = \{A_1, \ldots, A_n\} \) be a set of ABoxes and \( T \) be an integrity constraint. Let \( \{\pi_{A_1}, \ldots, \pi_{A_n}\} \) possibility distributions associated with \( (\langle T, A_1 \rangle, \ldots, \langle T, A_n \rangle) \). The result of merging is a DL-Liteπ knowledge base, denoted by \( \Delta_T^{\text{min}}(A) \) where its model are defined by:

\[
\text{Mod}(\Delta_T^{\text{min}}(A)) = \{I \in \text{Mod}(T) : I <_{\text{min}} I' \in \text{Mod}(T), I' <_{\text{min}} I\}
\]

In general, merging two DL-Liteπ normalized possibility distributions may lead to a sub-normalized possibility distribution. The normalization process comes down to set the degrees of interpretations in \( \text{Mod}(\Delta_T^{\text{min}}(A)) \) to 1.

From a syntactic point of view, the min-based merging operator, denoted by \( \Delta_T^{\text{min}}(A) \) is the union of all ABox. Namely:

\[
\Delta_T^{\text{min}}(A) = \bigcup_{I <_{\text{min}} I' <_{\text{min}} I} I
\]
The aggregation of ABoxes is not guaranteed to be consistent. Namely, the resulting knowledge base \((T, \Delta^{min}(A))\) may be inconsistent. To restore the consistency of the resulting knowledge base a normalization step is required. The following definition gives the formal logical representation of the normalized knowledge base.

**Definition 4.** Let \(T\) be a TBox and \(\Delta^{min}(A)\) be the aggregation of \(A_1, \ldots, A_n\), n ABox using classical min-based operator. Let \(x = \Delta^{min}(A)\). Then, the normalized knowledge base, denoted \(\Delta^{min}(K)\) is such that:

\[
\Delta^{min}(K) = \{(f_{ij}, W(f_{ij})): (f, W(f_{ij}) \in \Delta^{min}(A) \text{ and } W(f_{ij}) > x)\}
\]

**Example 2 (continued).** Let us continue with the TBox 
\[\mathcal{T} = (A \sqsubseteq B, B \sqsubseteq \neg C)\] 
presented in Example 1 while assuming that the certainty degree of each axioms is set to 1. Let us consider the following set of ABoxes to be linked to \(T\): \(A_1 = \{(A(a), 6), (C(b), 5)\}, A_2 = \{(C(a), 4), (B(b), 8), (A(b), 7)\}\). We have:

<table>
<thead>
<tr>
<th>(T)</th>
<th>(A)</th>
<th>(\pi_A)</th>
<th>(\Delta^{min}(A))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I_1)</td>
<td>(A = {a}, B = {a}, C = {b})</td>
<td>(0.1)</td>
<td>(0.2)</td>
</tr>
<tr>
<td>(I_2)</td>
<td>(A = {}, B = {}, C = {a})</td>
<td>(0.2)</td>
<td>(0.4)</td>
</tr>
<tr>
<td>(I_3)</td>
<td>(A = {a, b}, B = {a, b}, C = {})</td>
<td>(0.5)</td>
<td>(0.6)</td>
</tr>
<tr>
<td>(I_4)</td>
<td>(A = {b}, B = {b}, C = {a})</td>
<td>(0.4)</td>
<td>(0.1)</td>
</tr>
</tbody>
</table>

Table 2: Example of merging of possibility distributions using min-based operator

One can check that the resulting possibility distribution \(\Delta^{min}(K)\) is sub-normalized. To normalize \(\Delta^{min}(K)\), it is enough to set \(I_3 = 0.5\). At syntactic level, we have \(\Delta^{min}(K) = (T, \{(A(a), 6), (C(b), 5), (C(a), 4), (B(b), 8), (A(b), 7)\})\). We have \(Inc(\Delta^{min}(A)) = 5\) and \(\Delta^{min}(K) = (T, \{(A(a), 6), (B(b), 8), (A(b), 7)\})\).

In the next section, we investigate min-based merging under incomensurability assumption.

**Using compatible scales**

The min-based merging operator presented in the previous section is defined over the assumption that all the sources providing the ABoxes use the same scale to encode uncertainties between facts. In Example 2, when dealing with assertions, we assumed that the weight attached to \(f \in A_i\) can be compared to the weight associated with \(g \in A_j\) with \(j \neq i\). In this section, we drop this assumption and we suppose that sources are incomensurable.

We investigate a min-based fusion operator to merge incomensurable DL-Lite assertional bases. To make sources using different scale commensurable, we use the notion of “compatible scale” on existing scales used by each source.

A ranking scale is said to be compatible with all sources if it preserves original order relations between assertions of each ABox. The new ranking, denoted by \(\mathcal{R}\), defines a new ranking relations for each ABox to be merged. More formally,

**Definition 5** (Compatible ranking scale). Let \(A = \{A_1, \ldots, A_n\}\) where \(A_i = \{(f_{ij}, W_{A_i}(f_{ij}))\}\). Then a ranking \(\mathcal{R}\) is defined by:

\[
\mathcal{R} : \quad A_1 \sqsubseteq \ldots \sqsubseteq A_n \implies [0, 1] \\
(f_{ij}, W_{A_i}(f_{ij})) \implies \mathcal{R}(f_{ij})
\]

A ranking \(\mathcal{R}\) is said to be compatible with \(W_{A_1}, \ldots, W_{A_n}\) if and only if:

\[
\forall A_i \in A, \forall f, W_{A_i}(f), (f', W_{A_i}(f')) \in A_i, \\
W_{A_i}(f) \leq W_{A_i}(f') \iff \mathcal{R}(f) \leq \mathcal{R}(f').
\]

Definition 5 is basically the adaptation of the one given in (Benferhat, Lagrue, and Rossit 2007) for the context of DL-Lite.

**Example 3 (continued).** Let us consider again the following set of ABoxes to be linked to \(T\) given in Example 2: \(A_1 = \{(A(a), 6), (C(b), 5)\}, A_2 = \{(C(a), 4), (B(b), 8), (A(b), 7)\}\). The following table gives examples of ranking scales.

<table>
<thead>
<tr>
<th>(f_{ij})</th>
<th>(W_{A_i}(f_{ij}))</th>
<th>(\mathcal{R}<em>1(f</em>{ij}))</th>
<th>(\mathcal{R}<em>2(f</em>{ij}))</th>
<th>(\mathcal{R}<em>3(f</em>{ij}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A(a))</td>
<td>(0.6)</td>
<td>(0.5)</td>
<td>(0.4)</td>
<td>(0.6)</td>
</tr>
<tr>
<td>(C(b))</td>
<td>(0.5)</td>
<td>(0.2)</td>
<td>(0.7)</td>
<td>(0.5)</td>
</tr>
<tr>
<td>(B(b))</td>
<td>(0.8)</td>
<td>(0.7)</td>
<td>(0.6)</td>
<td>(0.8)</td>
</tr>
<tr>
<td>(A(b))</td>
<td>(0.7)</td>
<td>(0.4)</td>
<td>(0.2)</td>
<td>(0.7)</td>
</tr>
</tbody>
</table>

Table 3: Examples of ranking scales

The scaling \(\mathcal{R}_1\) is a compatible one, because it preserves the order inside each ABox. However, the scaling \(\mathcal{R}_2\) is not a compatible one since it inversed priorities inside \(A_1\) and \(A_2\).

According to Example 3, it is obvious that a compatible ranking scale is not unique. Let us denote by \(\mathcal{R}(A)\) the set of compatible ranking associated with \(A = \{A_1, \ldots, A_n\}\). The set \(\mathcal{R}(A)\) is non-empty and an immediate way to obtain a ranking relation over \(A\) is to consider \(\mathcal{R}(f_{ij}) = W_{A_i}(f_{ij})\) (For instance, the scale \(\mathcal{R}_3(f_{ij})\) given in Example 3). Note that this ranking is compatible in the sense that it permits to preserve the relative ordering between assertions of each \(A_i\).

Given a compatible scales \(\mathcal{R}\), we denote by \(A^R\) the assertional base obtained from \(A\), by replacing each assertion \((f_{ij}, W_{A_i}(f_{ij}))\) by \((f_{ij}, \mathcal{R}(f_{ij}))\). Similarly, we denote by \(A^R\) the set obtained from \(A\) by replacing each \(A_i\) in \(A\) by \(A_i^R\).

Now, given the set of all compatible scales \(\mathcal{R}(A)\), different possibilities may exist in order to merge the ABoxes. For instance, one can only select one scale to perform merging or one can consider all the compatible ranking in \(\mathcal{R}(A)\), etc. To avoid an arbitrary choice, we consider all compatible rankings to perform merging.

**Semantics merging**

We first introduce the notion of preference between interpretation according to the notion of compatible scales. An interpretation \(\mathcal{I}\) is then said to be preferred to \(\mathcal{I}'\), if for each compatible scale \(\mathcal{R}\), \(\mathcal{I}\) is preferred to \(\mathcal{I}'\) using Definition 2 (namely, \(\mathcal{I} \triangleleft_{\mathcal{R}} \mathcal{I}'\)). More precisely,
Definition 6 (Ordering between interpretations). Let $A = \{A_1, \ldots, A_n\}$ be a set of DL-Liteπ ABoxes and $R(A)$ be the set of all compatible interpretations associated with $A$. Let $I, I'$ be two interpretations. Then:

$$ I \prec I' \iff \forall R \in R(A), I \prec_{\text{Min}} I' $$

where $\prec_{\text{Min}}$ is the result of applying Definition 2 on $A^R$.

According to Definition 6, we have models of $\Delta_I^\gamma (A)$ are those which are models of $T$ and minimal for $\prec$, namely:

$$ \text{Mod}(\Delta_I^\gamma (A)) = \{ I \in \text{Mod}(T) : \exists I' \in \text{Mod}(T), I' \prec I \} $$

Note that $\prec$ is only a partial order. The following proposition shows that an interpretation $I$ is a model of $\Delta_I^\gamma (A)$ if and only if there exists a compatible scaling $\mathcal{R}$ where this interpretation belongs to the result fusion, namely it is a model of $\Delta^\mathcal{R}_{\text{Min}} (A)$. More formally:

**Proposition 1.** Let $A$ be a set of ABoxes linked to the same TBox $T$. Then $I \in \text{Mod}(\Delta^\gamma_I (A))$, if and only if there exists a compatible scaling $\mathcal{R}$ such that $I \in \text{Mod}(\Delta^\mathcal{R}_{\text{Min}} (A^\mathcal{R}))$.

The following example illustrates the fusion based on all compatible scalings.

**Example 4 (continued).** Let us consider again the following set of ABoxes given in Example 2: $A_1 = \{(A(a), .6), (C(b), .5)\}$, $A_2 = \{(C(a), .4), (B(b), .8), (A(b), 7)\}$. Let us consider again $R_1$ where $A^R_1 = \{(A(a), .8), (C(b), .4)\}$ and $A^R_2 = \{(C(a), .2), (B(b), .9), (A(b), 6)\}$ and a scaling $\mathcal{R}_2$ where $A^\mathcal{R}_4 = \{(A(a), .4), (C(b), .2)\}$ and $A^\mathcal{R}_2 = \{(C(a), .3), (B(b), .6), (A(b), .5)\}$. Both of them are compatible. Table 4 presents the profile of each interpretation for each scaling.

<table>
<thead>
<tr>
<th>$I$</th>
<th>$\nu_{A^R_1}(I)$</th>
<th>Min</th>
<th>$\nu_{A^R_2}(I)$</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$</td>
<td>$&lt;1, .1 &gt;$</td>
<td>$.4$</td>
<td>$&lt;1, .4 &gt;$</td>
<td>$.4$</td>
</tr>
<tr>
<td>$I_2$</td>
<td>$&lt;2, .1 &gt;$</td>
<td>$.4$</td>
<td>$&lt;6, .4 &gt;$</td>
<td>$.4$</td>
</tr>
<tr>
<td>$I_3$</td>
<td>$&lt;6, .8 &gt;$</td>
<td>$.6$</td>
<td>$&lt;8, .7 &gt;$</td>
<td>$.7$</td>
</tr>
<tr>
<td>$I_4$</td>
<td>$&lt;2, 1 &gt;$</td>
<td>$.6$</td>
<td>$&lt;6, 1 &gt;$</td>
<td>$.6$</td>
</tr>
</tbody>
</table>

Table 4: Two equivalent compatible scalings

Note that in both compatible scalings $R_1$ and $R_2$, $I_3$ is the preferred one.

Once preferred models are computed, query answering from a set of uncertain ABox under incommensurability assumption, is defined as follows:

**Definition 7.** Let $A = A_1, \ldots, A_n$ be a set of ABoxes linked to the same TBox $T$. A query $q$ is said to be consequence of $A$ under incommensurability assumption if $\forall I, I \in \text{Mod}(\Delta^\mathcal{R}_{\text{Min}} (A^\mathcal{R})), I | q$.

Example 5 (continued). From Example 4, we have $\text{Mod}(\Delta^\gamma_I (A^\mathcal{R})) = \{I_3\}$ where $A^\mathcal{R}_3 = \{a, b\}$, $B^\mathcal{R}_3 = \{a, b\}$ and $C^\mathcal{R}_3 = \{}$. Let $q_1(x) \leftarrow A(x) \land B(x)$ be a conjunctive query. One can easily check that $< b >$ is an answer of $q_1(x)$ using $\Delta^\mathcal{R}_{\text{Min}} (A^\mathcal{R})$. Similarly, let $B(a)$ be an instance query, one can check that $B(a)$ follows from $\Delta^\mathcal{R}_{\text{Min}} (A^\mathcal{R})$.

Using the set of all compatible scales may lead to a very cautious merging operation. One way to get rid of incommensurability assumption is to use some normalization function in the spirit of the ones used in clustering methods for gathering attributes having incommensurable domains. Let $A_1$ be an ABox and $\alpha_{A_1}$ be the set of different certainty degrees used in $A_1$. Let $\text{min}_{A_1}$ and $\text{max}_{A_1}$ be respectively the minimum and maximum certainty degrees associated with assertional facts in $\alpha_{A_1}$. Then an example of a normalization function is

$$ N(\alpha_i) = \frac{\alpha_i - \text{min}_{A_1} - \epsilon}{\text{max}_{A_1} - \text{min}_{A_1} - \epsilon} \quad (1) $$

where $\epsilon$ is a certainty degree belonging to $\alpha_{A_1}$ and $\epsilon$ is a very small number (lower than $\min_{A_1}$).

The main advantage of only having one normalization function is that one can have an immediate syntactic counterpart. More precisely, it is enough to replace for each fact $(f_{ij}, W_{A_1}(f_{ij}))$ by $(f_{ij}, N(W_{A_1}(f_{ij})))$ where $N$ is the normalization function given by Equation 1.

**Example 6 (continued).** From Example 2, we have $A_1 = \{(A(a), .6), (C(b), .5)\}$, $A_2 = \{(C(a), .4), (B(b), .8), (A(b), 7)\}$. We have $\text{min}_{A_1} = .5$, $\text{min}_{A_2} = A, \text{max}_{A_1} = .6$ and $\text{max}_{A_2} = .8$. Let $\epsilon = .01$, then applying Equation 1 on $A_1$ and $A_2$ gives: $A_1 = \{(A(a), 1), (C(b), .09)\}$, and $A_2 = \{(C(a), 0.02), (B(b), 1), (A(b), .75)\}$.

Once the syntactic computation of normalized assertional bases is done, it is enough the reuse merging of commensurable possibilistic knowledge bases for query answering.

**Example 7 (continued).** From Example 6, we have $\Delta^\mathcal{R}_{\text{Min}} (A) = \langle T, \{(A(a), 1), (C(b), .09), (C(a), .02), (B(b), 1), (A(b), .75)\} \rangle$. We have $\text{Inc}(\Delta^\mathcal{R}_{\text{Min}} (A)) = .09$ and $\Delta^\mathcal{R}_{\text{Min}} (K) = \langle T, \{(A(a), 1), (B(b), 1), (A(b), .75)\} \rangle$.

Consider now $q_1(x) \leftarrow A(x) \land B(x)$ and $q_2 \leftarrow A(a)$, queries given in Example 5. One can check that $< b >$ is an answer of $q_1(x)$ and the $\text{and} (a)$ holds from the resulting knowledge bases.

Conclusions

This paper proposed a mini-based possibilistic merging operation of uncertain assertional facts under incommensurability assumption. The idea is to reuse standard mini-based merging, over a set of compatible scales. Future work includes developing a syntactic counterpart of incommensurable merging operation. A natural question is whether one can extend a polynomial time complexity algorithm, defined for query answering from a standard uncertain ABox, to the case where uncertainty scales are incommensurable.

Acknowledgment

This work has been supported by the French National Research Agency. ASPIQ project ANR-12-BS02-0003

References
