Fundamenta Informaticae XX (2018) 1–29
DOI 10.3233/FI-2015-0000
IOS Press

Reasoning on Datalog$^{\pm}$ Ontologies with Abductive Logic Programming

Marco Gavanelli*
Dipartimento di Ingegneria
University of Ferrara

Evelina Lamma
Dipartimento di Ingegneria
University of Ferrara

Fabrizio Riguzzi
Dipartimento di Matematica e Informatica
University of Ferrara

Elena Bellodi
Dipartimento di Ingegneria
University of Ferrara

Riccardo Zese
Dipartimento di Ingegneria
University of Ferrara

Giuseppe Cota
Dipartimento di Ingegneria
University of Ferrara

Abstract. Ontologies form the basis of the Semantic Web. Description Logics (DLs) are often the languages of choice for modeling ontologies. Integration of DLs with rules and rule-based reasoning is crucial in the so-called Semantic Web stack vision - a complete stack of recommendations and languages each based on and/or exploiting the underlying layers - which adds new features to the standards used in the Web. The growing importance of the integration between DLs and rules is proved by the definition of the profile OWL 2 RL[1] and the definition of languages such as RIF[2] and SWRL[3]. Datalog$^{\pm}$ is an extension of Datalog which can be used for representing lightweight ontologies and expressing some languages of the DL-Lite family, with tractable query answering under certain language restrictions. In particular, it is able to express the DL-Lite version defined in OWL. In this work, we show that Abductive Logic Programming (ALP) can be used to represent Datalog$^{\pm}$ ontologies, supporting query answering through an abductive proof procedure, and smoothly achieving the integration of ontologies and rule-based reasoning. Often, reasoning with DLs means finding explanations for the truth of queries, that are useful when debugging ontologies and to understand answers given by the reasoning process. We show that reasoning under existential

*This work was partially supported by GNCS project DECORE.

[1] `https://www.w3.org/TR/owl2-profiles`
[2] `https://www.w3.org/standards/techs/rif`
[3] `https://www.w3.org/Submission/SWRL/`

rules can be expressed by ALP languages and we present a solving system, which is experimentally proved to be competitive with DL reasoning systems. In particular, we consider an ALP framework named $\mathcal{S}$CIFF derived from the IFF abductive framework. Forward and backward reasoning is naturally supported in this ALP framework. The $\mathcal{S}$CIFF language smoothly supports the integration of rules, expressed in a Logic Programming language, with Datalog$^{\pm}$ ontologies, mapped into $\mathcal{S}$CIFF (forward) integrity constraints. The main advantage is that this integration is achieved within a single language, grounded on abduction in computational logic, and able to model existential rules.

1. Introduction

The main goal of the Semantic Web is to disseminate information in a form automatically processable by machines [1]. Semantic Web technologies allow the reuse and sharing of data together with more sophisticated queries. Data, in order to be managed by machines, must be represented in a standardized logic format.

Ontologies are engineering artefacts consisting of a vocabulary describing some domain, and an explicit specification of the intended meaning of the vocabulary (i.e., how entities should be classified), possibly together with constraints capturing additional knowledge about the domain. Ontologies therefore provide a formal and machine manipulable model for objects, concepts and entities that exist in a particular domain, and constitute the basis of the Semantic Web.

The W3C, one of the main supporters of the Semantic Web, developed a family of formalisms of increasing complexity for representing ontologies, called Web Ontology Language (OWL)[4]. Description Logics (DLs) form the theoretical foundations of OWL. Of particular interest is the *DL-Lite* family [2], that identifies a set of tractable DLs for which answering conjunctive queries is in AC_0 in data complexity.

When reasoning with DLs it is of foremost importance to find explanations for queries, which can help to understand the results of the queries and, in case of errors, show where information must be corrected. In this paper we concentrate on this aspect of query answering, i.e. explanation finding. In the past years, several DL reasoners have been developed, such as Pellet [3], RacerPro [4] and HermiT [5], and most of them implement the tableau algorithm [6] in a procedural language to compute such explanations. This algorithm builds a graph representing the information about the objects in the knowledge base, called tableau, and then expands it by applying a set of rules in order to decide whether an axiom is entailed or not. This decision is done by refutation. Nonetheless, some tableau expansion rules are nondeterministic, thus requiring the implementation of a search strategy in an or-branching search space. In [7] the authors proposed an implementation of the tableau algorithm in Prolog, exploiting Prolog's backtracking facilities.

In a related research direction, the authors of [8] proposed Datalog$^{\pm}$, an extension of Datalog with existential rules. This formalism can be used for representing lightweight ontologies, and encompasses the DL-Lite family [9, 10]. By suitably restricting its syntax, also Datalog$^{\pm}$ achieves tractability [8]. Possible restrictions are given by sticky rules, weakly-acyclic rules and by guardedness [11].

A combination of both these two approaches is given by PAGOdA [12], which combines Datalog with Hermit [5] for improving the performances when answering queries by resorting to the OWL reasoner only where necessary and delegating to Datalog most of the computational load.

[4]Two major versions of this family of formalisms have been proposed (OWL 1 and OWL 2).

In this work we show how to model Datalog$^{\pm}$ ontologies in an Abductive Logic Programming (ALP) language enriched with quantified variables, where query answering is supported by the underlying ALP proof procedure. In other words, we show that reasoning under existential rules can be expressed by ALP languages. We do not focus here on complexity results of the overall system, which are, however, not tractable in general, since ALP is Turing-complete. ALP has been proved a powerful tool for knowledge representation and reasoning [13], taking advantage of ALP proof procedures. ALP languages are usually equipped with a declarative (model-theoretic) semantics, and an operational semantics is given in terms of a proof-procedure. Several abductive proof procedures have been defined (backward, forward, and a mix of the two), with many different applications (diagnosis, monitoring, verification, etc.). Among them, the IFF proof-procedure [14] was proposed to deal with forward rules, and with non-ground abducibles. In [15], $\mathcal{S}$CIFF was proposed, an extension of the IFF proof procedure that can deal with both existentially and universally quantified variables in rule heads, and Constraint Logic Programming (CLP) constraints [16]. The resulting system was used for modeling and implementing several knowledge representation frameworks, such as deontic logic [17], normative systems, interaction protocols for multi-agent systems [18], Web services choreographies [19], as well as to represent and implement the integration of declarative choreography languages into commitments [20].

Forward and backward reasoning is naturally supported by the ALP proof procedure, and the $\mathcal{S}$CIFF language smoothly supports the integration of rules, expressed in a Logic Programming language, with ontologies expressed in Datalog$^{\pm}$. In fact, $\mathcal{S}$CIFF allows us to map Datalog$^{\pm}$ ontologies into forward integrity constraints. We compare the $\mathcal{S}$CIFF abductive semantics and the Datalog$^{\pm}$ semantics, and find that the ALP semantics is more loose, meaning that a $\mathcal{S}$CIFF procedure is not always sound with respect to the Datalog$^{\pm}$ semantics and, symmetrically, the Datalog$^{\pm}$ proof procedure (the chase) is not always complete with respect to the $\mathcal{S}$CIFF semantics. We also define a property of the input program, that can be statically checked, that ensures that the two procedures produce the same results.

Experimental results presented in this paper show that the $\mathcal{S}$CIFF proof procedure has competitive performance to that of existing DL reasoning systems, in particular when all the explanations are required or when there are tight memory limits.

In the following, Section 2 briefly introduces Datalog$^{\pm}$. Section 3 presents Abductive Logic Programming and the $\mathcal{S}$CIFF language, with a mention to its abductive proof procedure. Section 4 shows how the considered Datalog$^{\pm}$ language can be mapped into $\mathcal{S}$CIFF, and the kind of queries that the abductive proof procedure can handle. Section 5 illustrates related work. In Section 6 we show experimentally that the $\mathcal{S}$CIFF proof procedure is able to answer queries in a time comparable to that of ontological reasoners. Section 7 concludes the paper and outlines future work.

2. Datalog$^{\pm}$

Datalog$^{\pm}$ extends Datalog by allowing existential quantifiers, the equality predicate and the truth constant *false* in rule heads. Datalog$^{\pm}$ can be used for representing lightweight ontologies and is able to express some members of the DL-Lite family of ontology languages [9]. In particular, it is able to express the DL-Lite version defined in OWL. By suitably restricting the language syntax, Datalog$^{\pm}$ achieves tractability [21].

In order to describe Datalog$^{\pm}$, let us assume (i) an infinite set of data constants Δ, (ii) an infinite set of labeled nulls Δ_N (used as "fresh" Skolem terms), and (iii) an infinite set of variables Δ_V. Different

constants represent different values (unique name assumption), while different nulls may represent the same value. We assume a lexicographic order on $\Delta \cup \Delta_N$, with every symbol in Δ_N following all symbols in Δ. We denote by $\mathbf{X}$ vectors of variables $X_1, \ldots, X_k$ with $k \geq 0$. A relational schema $\mathcal{R}$ is a finite set of relation names (or predicates). A term t is a constant, null or variable. An atomic formula (or atom) has the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate, and $t_1, \ldots, t_n$ are terms. A database D for $\mathcal{R}$ is a possibly infinite set of atoms with predicates from $\mathcal{R}$ and arguments from $\Delta \cup \Delta_N$. A Conjunctive Query (CQ) over $\mathcal{R}$ has the form $q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms having as arguments variables $\mathbf{X}$ and $\mathbf{Y}$ and constants (but no nulls). A Boolean CQ (BCQ) over $\mathcal{R}$ is a CQ having head predicate q of arity 0 (i.e., no variables in $\mathbf{X}$).

We often write a BCQ as the set of all its atoms, having constants and variables as arguments, and omitting the quantifiers. Answers to CQs and BCQs are defined via homomorphisms, which are mappings $\mu : \Delta \cup \Delta_N \cup \Delta_V \rightarrow \Delta \cup \Delta_N \cup \Delta_V$ such that (i) $c \in \Delta$ implies $\mu(c) = c$, (ii) $c \in \Delta_N$ implies $\mu(c) \in \Delta \cup \Delta_N$, and (iii) μ is naturally extended to term vectors, atoms, sets of atoms, and conjunctions of atoms. The set of all answers to a CQ $q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$ over a database D, denoted $q(D)$, is the set of all tuples $\mathbf{t}$ over Δ for which there exists a homomorphism $\mu : \mathbf{X} \cup \mathbf{Y} \rightarrow \Delta \cup \Delta_N$ such that $\mu(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$ and $\mu(\mathbf{X}) = \mathbf{t}$. The answer to a BCQ $q = \exists \mathbf{Y} \Phi(\mathbf{Y})$ over a database D, denoted $q(D)$, is Yes, denoted $D \models q$, iff there exists a homomorphism $\mu : \mathbf{Y} \rightarrow \Delta \cup \Delta_N$ such that $\mu(\Phi(\mathbf{Y})) \subseteq D$, i.e., if $q(D) \neq \emptyset$.

Given a relational schema $\mathcal{R}$, a *tuple-generating dependency* (or TGD) F is a first-order formula of the form $\forall \mathbf{X} \forall \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ and $\Psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$, called the *body* and the *head* of F, respectively. Such F is satisfied in a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h such that $h(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$, there exists an extension h' of h such that $h'(\Psi(\mathbf{X}, \mathbf{Z})) \subseteq D$. We usually omit the universal quantifiers in TGDs.

Query answering under TGDs is defined as follows. For a set of TGDs T_T on $\mathcal{R}$, and a database D for $\mathcal{R}$, the set of models of D given T_T, denoted $mods(D, T_T)$, is the set of all (possibly infinite) databases B such that $D \subseteq B$ and every $F \in T_T$ is satisfied in B. The set of answers to a CQ q on D given T_T, denoted $ans(q, D, T_T)$, is the set of all tuples $\mathbf{t}$ such that $\mathbf{t} \in q(B)$ for all $B \in mods(D, T_T)$. The answer to a BCQ q over D given T_T is Yes, denoted $D \cup T_T \models q$, iff $B \models q$ for all $B \in mods(D, T_T)$.

A Datalog$^{\pm}$ theory may contain also *negative constraints* (or NC), which are first-order formulas of the form $\forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow \bot$, where $\Phi(\mathbf{X})$ is a conjunction of atoms. The universal quantifiers are usually left implicit.

Equality-generating dependencies (or EGDs) are the third component of a Datalog$^{\pm}$ theory. An EGD F is a first-order formula of the form $\forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow X_i = X_j$, where $\Phi(\mathbf{X})$, called the *body* of F, is a conjunction of atoms, and X_i and X_j are variables from $\mathbf{X}$. We call $X_i = X_j$ the *head* of F. Such F is satisfied in a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h such that $h(\Phi(\mathbf{X})) \subseteq D$, it holds that $h(X_i) = h(X_j)$. We usually omit the universal quantifiers in EGDs.

The *chase* is a bottom-up procedure for deriving atoms entailed by a database and a Datalog$^{\pm}$ theory. The chase works on a database through the so-called TGD and EGD chase rules.

The TGD chase rule is defined as follows. Given a relational database D for a schema $\mathcal{R}$, and a TGD F on $\mathcal{R}$ of the form $\forall \mathbf{X} \forall \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z})$, F is *applicable to* D if there is a homomorphism h that maps the atoms of $\Phi(\mathbf{X}, \mathbf{Y})$ to atoms of D. Let F be applicable and h_1 be a homomorphism that extends h as follows: for each $X_i \in \mathbf{X}$, $h_1(X_i) = h(X_i)$; for each $Z_j \in \mathbf{Z}$, $h_1(Z_j) = z_j$, where z_j is a "fresh" null, i.e., $z_j \in \Delta_N, z_j \notin D$, and z_j lexicographically follows all other labeled nulls already introduced. The TGD chase rule is applied when a new tuple $\mathbf{t}$ is found that satisfies $D \models \exists \mathbf{Y} \Phi(\mathbf{t}, \mathbf{Y})$;

the result of the application of the TGD chase rule for F is the addition to D of all the atomic formulas in $h_1(\Psi(\mathbf{X}, \mathbf{Z}))$ that are not already in D.

The EGD chase rule is defined as follows [22]. An EGD F on $\mathcal{R}$ of the form $\Phi(\mathbf{X}) \rightarrow X_i = X_j$ is *applicable to* a database D for $\mathcal{R}$ iff there exists a homomorphism $h : \Phi(\mathbf{X}) \rightarrow D$ such that $h(X_i)$ and $h(X_j)$ are different and not both constants. If $h(X_i)$ and $h(X_j)$ are different constants in Δ, then there is a *hard violation* of F. Otherwise, the result of the application of F to D is the database $h(D)$ obtained from D by replacing every occurrence of $h(X_i)$ with $h(X_j)$ if $h(X_i)$ precedes $h(X_j)$ in the lexicographic order, and every occurrence of $h(X_j)$ with $h(X_i)$ if $h(X_j)$ precedes $h(X_i)$ in the lexicographic order.

The chase algorithm consists of an exhaustive application of the TGD and EGD chase rules that may lead to an infinite result. The chase rules are applied iteratively: in each iteration (1) a single TGD is applied once and then (2) the EGDs are applied until a fix point is reached. EGDs are assumed to be separable [11]. Intuitively, separability holds whenever: (i) if there is a hard violation of an EGD in the chase, then there is also one on the database w.r.t. the set of EGDs alone (i.e., without considering the TGDs); and (ii) if there is no hard violation, then the answers to a BCQ w.r.t. the entire set of dependencies equals those w.r.t. the TGDs alone (i.e., without the EGDs).

Query answering under TGDs is equivalent to query answering under TGDs with only single atoms in their heads [21]. Henceforth, we assume that every TGD has a single atom in its head. A BCQ q on a database D, a set T_T of TGDs and a set T_E of EGDs can be answered by performing the chase and checking whether the query is entailed by the extended database that is obtained. In this case we write $D \cup T_T \cup T_E \models q$.

Example 2.1. Let us consider the following ontology for a real estate information extraction system, a slight modification of the one presented in Gottlob et al. [22]:

$F_1 = ann(X, label), ann(X, price), visible(X) \rightarrow priceElem(X)$

If X is annotated as a label, as a price and is visible, then it is a price element.

$F_2 = ann(X, label), ann(X, priceRange), visible(X) \rightarrow priceElem(X)$

If X is annotated as a label, as a price range, and is visible, then it is a price element.

$F_3 = priceElem(E), group(E, X) \rightarrow forSale(X)$

If E is a price element and is grouped with X, then X is for sale.

$F_4 = forSale(X) \rightarrow \exists P\, price(X, P)$

If X is for sale, then there exists a price for X.

$F_5 = hasCode(X, C), codeLoc(C, L) \rightarrow loc(X, L)$

If X has postal code C, and C's location is L, then X's location is L.

$F_6 = hasCode(X, C) \rightarrow \exists L\, codeLoc(C, L), loc(X, L)$

If X has postal code C, then there exists L such that C has location L and so does X.

$F_7 = loc(X, L1), loc(X, L2) \rightarrow L1 = L2$

If X has the locations $L1$ and $L2$, then $L1$ and $L2$ are the same.

$F_8 = loc(X, L) \rightarrow advertised(X)$

If X has a location L then X is advertised.

Suppose we are given the database:

$$codeLoc(ox1, central), codeLoc(ox1, south), codeLoc(ox2, summertown)$$
$$hasCode(prop1, ox2), ann(e1, price), ann(e1, label), visible(e1),$$
$$group(e1, prop1)$$

The atomic BCQs $priceElem(e1)$, $forSale(prop1)$ and $advertised(prop1)$ evaluate to true, while the CQ $loc(prop1, L)$ has answer $q(L) = \{summertown\}$. In fact, even if $loc(prop1, z_1)$ with $z_1 \in \Delta_N$ is entailed by formula F_6, formula F_7 imposes that $summertown = z_1$. □

Checking whether a set of NCs is satisfied by a database given a set of TGDs corresponds with query answering [11]. In particular, given a database D, a set T_T of TGDs and a set $T_\perp$ of NCs, for each constraint $\forall \mathbf{X}\Phi(\mathbf{X}) \rightarrow \perp$ we evaluate the BCQ $q = \exists \mathbf{X}\Phi(\mathbf{X})$ over $D \cup T_T$. If at least one of such queries answers positively, then $D \cup T_T \cup T_\perp \models \perp$ (i.e., the theory is inconsistent), and thus for every BCQ q it holds that $D \cup T_T \cup T_\perp \models q$; otherwise, given a BCQ q, we have that $D \cup T_T \cup T_\perp \models q$ iff $D \cup T_T \models q$, i.e, we can answer q by ignoring the constraints.

3. Abductive Logic Programming

Abductive Logic Programming (ALP, for short) is a family of programming languages that integrate abductive reasoning into logic programming. In the following, we assume some familiarity with logic programming terminology; a good introduction is given by Lloyd [23]. An ALP program is a logic program, consisting of a set of clauses

$$head \leftarrow body$$

where the $head$ is an atom. A set of clauses with the same predicate symbol in the $head$ define a predicate. The $body$ can contain literals built either from defined predicates or from some distinguished predicates, belonging to a set $\mathcal{A}$ and called *abducibles*. The aim is finding a set of abducible literals $\mathbf{EXP}$, built from symbols in $\mathcal{A}$ that, together with the knowledge base KB, is an explanation for a given known effect (also called *goal* $\mathcal{G}$):

$$KB \cup \mathbf{EXP} \models \mathcal{G}. \tag{1}$$

Also, $\mathbf{EXP}$ should satisfy a set of logic formulae, called *Integrity Constraints* IC:

$$KB \cup \mathbf{EXP} \models IC. \tag{2}$$

Example 3.1. A knowledge base might contain a set of rules stating that a person is a nature lover

$$natureLover(X) \leftarrow \mathbf{hasAnimal}(X, Y), \mathbf{pet}(Y).$$
$$natureLover(X) \leftarrow \mathbf{biologist}(X).$$

From this knowledge base one can infer, e.g., that each person who owns a pet is a nature lover. However, in some cases we might have the information that $kevin$ is a nature lover, and wish to infer more information about him. In such a case we might label predicates $\mathbf{hasAnimal}$, $\mathbf{pet}$ and $\mathbf{biologist}$ as abducible (in the following, abducible predicates are written in **bold**) and apply an abductive proof procedure to the knowledge base. Two explanations are possible: either there exists an animal that is owned by $kevin$ and that is a pet:

$$(\exists Y) \qquad \mathbf{hasAnimal}(kevin, Y), \mathbf{pet}(Y)$$

or $kevin$ is a biologist:

$$\mathbf{biologist}(kevin)$$

We see that the computed answers includes abduced atoms, which can contain variables. □

Integrity constraints can help reducing the number of computed explanations, ruling out those that are not possible.

Example 3.2. Let us consider the knowledge base of Example 3.1. The following integrity constraint states that to become a biologist one needs to be at least 25 years old:

$$\mathbf{biologist}(X), \mathbf{age}(X, A) \rightarrow A \geq 25$$

We might know that $kevin$ is a child, and have a definition of the predicate $child$:

$$child(X) \leftarrow \mathbf{age}(X, A), A < 10.$$

In this example we see the usefulness of constraints as in Constraint Logic Programming [16]: the symbols $<, \geq, ...$ are handled as constraints, i.e., they are not predicates defined in a knowledge base, but they associate a numeric domain to the involved variables and restrict it according to constraint propagation. Now, the goal $natureLover(kevin), child(kevin)$ returns only one possible explanation:

$$(\exists Y)(\exists A) \qquad \mathbf{hasAnimal}(kevin, Y), \mathbf{pet}(Y), \mathbf{age}(kevin, A) \qquad A < 10$$

since the option that $kevin$ is a biologist is ruled out. Note that we do not need to know the exact age of $kevin$ to rule out the biologist hypothesis. □

3.1. The $\mathcal{S}$CIFF declarative semantics and proof-procedure

$\mathcal{S}$CIFF [15] is a language in the ALP class, originally designed to model and verify interactions in open societies of agents [18], and it is an extension of the IFF proof-procedure [14]. As in the IFF language, it considers integrity constraints (ICs, for short in the following) in the form of forward rules

$$body \rightarrow head$$

where the $body$ is a conjunction of literals and the head is a disjunction of conjunctions of literals. While in the IFF the literals can be built only on defined or abducible predicates, in $\mathcal{S}$CIFF they can also be Constraint Logic Programming (CLP) constraints, occurring events (only in the body), or positive and negative expectations.

Definition 3.3. A *$\mathcal{S}$CIFF Program* is a pair $\langle KB, \mathcal{IC} \rangle$ where KB is a set of clauses and $\mathcal{IC}$ is a set of *Integrity Constraints*. □

Variables occurring in $\mathcal{S}$CIFF integrity constraints can be existentially or universally quantified; the quantification is left implicit in the $\mathcal{S}$CIFF syntax, and the correct quantification is given by the set of quantification rules reported in [15]. Concerning the cases relevant for this work, the quantification rules can be simplified as follows:

- all variables occurring in the body are universally quantified;
- all variables occurring only in the head are existentially quantified.

In the following, we will often omit the quantification, and make it explicit only when we wish to highlight it to the reader.

$\mathcal{S}$CIFF considers a (possibly dynamically growing) set of facts (named event set) denoted $\mathbf{HAP}$, that contains ground atoms $\mathbf{H}(Event)$. This set, also called *history*, can grow dynamically, during the computation, thus implementing a dynamic acquisition of events. Some distinguished abducibles are called *expectations*. A *positive expectation*, written $\mathbf{E}(Event)$, means that a corresponding event $\mathbf{H}(Event)$ is expected to happen, while $\mathbf{EN}(Event)$ is a *negative expectation*, and requires that a matching event $\mathbf{H}(Event)$ does not appear in the event set.

While events in $\mathbf{HAP}$ are ground atoms, expectations can contain variables. In positive expectations all variables are existentially quantified (expressing the idea that a single event is enough to support them), while negative expectations are universally quantified, so that any event matching with a negative expectation leads to inconsistency with the current hypothesis. CLP [16] constraints can be imposed on variables. The computed answer includes in general three elements: a substitution for the variables in the goal (as usual in Prolog), the constraint store (as in CLP), and the set $\mathbf{EXP}$ of abduced literals, also called abductive explanation.

The declarative semantics of $\mathcal{S}$CIFF includes the classic conditions of abductive logic programming

$$KB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathcal{G} \tag{3}$$

$$KB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathcal{IC} \tag{4}$$

(where the $\models$ symbol is interpreted, as in the IFF, as 3-valued completion semantics [24]), plus specific conditions to support the confirmation of expectations. In this paper, variables in $\mathcal{G}$ are considered existentially quantified, and variables in $\mathcal{IC}$ are quantified as previously explained.

Positive expectations are confirmed if

$$(\forall X) \qquad KB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathbf{E}(X) \rightarrow \mathbf{H}(X), \tag{5}$$

while negative expectations are confirmed (or better they are not violated) if

$$(\forall X) \qquad KB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathbf{EN}(X) \wedge \mathbf{H}(X) \rightarrow \mathit{false}. \tag{6}$$

The declarative semantics of $\mathcal{S}$CIFF also requires that the same event cannot be expected both to happen and not to happen

$$(\forall X) \qquad KB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathbf{E}(X) \wedge \mathbf{EN}(X) \rightarrow \mathit{false} \tag{7}$$

Definition 3.4. ($\mathcal{S}$CIFF answer)
Given a $\mathcal{S}$CIFF program $\langle KB, \mathcal{IC} \rangle$ and a history $\mathbf{HAP}$, a goal $\mathcal{G}$ is a $\mathcal{S}$CIFF answer if there is a set $\mathbf{EXP}$ such that equations (3), (4), (5), (6) and (7) are satisfied. In this case, we write

$$\langle KB, \mathcal{IC} \rangle \models_{\mathbf{HAP}} \mathcal{G}$$

□

The $\mathcal{S}$CIFF proof-procedure is a rewriting system that defines a tree, whose nodes represent states of the computation. A set of transitions rewrite a node into one or more child nodes. $\mathcal{S}$CIFF inherits the transitions of the IFF proof-procedure [14], and extends it in various directions. We recall the basics

of $\mathcal{S}$CIFF; a complete description can be found in [15], with proofs of soundness, completeness, and termination. An efficient implementation of $\mathcal{S}$CIFF is described in [25].

Each node of the proof is a tuple $N \equiv \langle R, CS, PSIC, \mathbf{EXP} \rangle$, where R is the resolvent, CS is the CLP constraint store, PSIC is a set of implications (called *Partially Solved Integrity Constraints*) derived from the propagation of integrity constraints, and $\mathbf{EXP}$ is the current set of abduced literals. $\mathcal{S}$CIFF includes a set of transitions inherited from the IFF, the transitions of CLP [16] for constraint solving, plus some transitions devoted to specific features of the $\mathcal{S}$CIFF language, such as handling of dynamically happening events and fulfillment/violation of expectations.

The following is the subset of $\mathcal{S}$CIFF transitions that are relevant for this work, in a proof-theory style notation.

- propagation

$$\frac{\mathbf{H}(N) \qquad \mathbf{H}(N'), B \rightarrow H}{N = N', B \rightarrow H}$$

- case analysis

$$\frac{(N = N', B) \rightarrow H}{N = N' \quad B \rightarrow H \qquad \vee \qquad N \neq N'}$$

- equality rewriting

$$\frac{[\exists E][\forall A] A = E}{\theta = \{A/E\}} \qquad \frac{[\exists E][\forall A] A \neq E}{false}$$

$$\frac{[\exists E_1][\exists E_2] E_1 = E_2}{\theta = \{E_1/E_2\}}$$

$$\frac{X = t \qquad t \text{ does not contain } X}{\theta = \{X/t\}}$$

$$\frac{X = t \qquad t \text{ contains } X}{false} \qquad \frac{X \neq t \qquad t \text{ contains } X}{true}$$

$$\frac{p(t_1, \ldots, t_n) = p(s_1, \ldots, s_n)}{t_1 = s1, \ldots, t_n = s_n} \qquad \frac{p(t_1, \ldots, t_n) \neq p(s_1, \ldots, s_n)}{t_1 \neq s1 \vee \cdots \vee t_n \neq s_n}$$

$$\frac{p(t_1, \ldots, t_n) = q(s_1, \ldots, s_m) \text{ where } p \neq q \vee n \neq m}{false}$$

$$\frac{p(t_1, \ldots, t_n) \neq q(s_1, \ldots, s_m) \text{ where } p \neq q \vee n \neq m}{true}$$

where θ is the resulting substitution. In case two variables with different quantifiers are unified, the new variable is existentially quantified.

- logical simplifications

$$\frac{true \rightarrow A}{A} \qquad \frac{false \rightarrow A}{true} \qquad \frac{true \wedge A}{A}$$

$$\frac{false \wedge A}{false} \qquad \frac{true \vee A}{true} \qquad \frac{false \vee A}{A}$$

In this paper we consider the *generative version* of $\mathcal{S}$CIFF, called g-$\mathcal{S}$CIFF [26], in which also the $\mathbf{H}$ events are considered as abducibles, and can be assumed like the other abducible predicates, beside being provided as input in the history $\mathbf{HAP}$; they are then collected in a set $\mathbf{HAP'} \supseteq \mathbf{HAP}$.

Definition 3.5. (g-$\mathcal{S}$CIFF answer)
Given a $\mathcal{S}$CIFF program $\langle KB, \mathcal{IC} \rangle$ and a history $\mathbf{HAP}$, we say that a goal $\mathcal{G}$ is a g-$\mathcal{S}$CIFF answer if there exist a set $\mathbf{EXP}$ and a set $\mathbf{HAP'} \supseteq \mathbf{HAP}$ such that equations (3)-(7) are satisfied[5]. In this case, we write

$$\langle KB, \mathcal{IC} \rangle \models^{g}_{\mathbf{HAP} \rightsquigarrow \mathbf{HAP'}} \mathcal{G}$$

or simply

$$\langle KB, \mathcal{IC} \rangle \models^{g}_{\mathbf{HAP}} \mathcal{G}$$

□

The g-$\mathcal{S}$CIFF proof procedure also includes transition *regimentation*, that produces an event from each positive expectation that is not already fulfilled:

$$\frac{\mathbf{E}(N)}{\mathbf{H}(N)}.$$

4. Mapping Datalog$^{\pm}$ into ALP programs

In this section, we show that a Datalog$^{\pm}$ program can be represented as a set of $\mathcal{S}$CIFF integrity constraints and an event set. $\mathcal{S}$CIFF abductive declarative semantics provides the model-theoretic counterpart to Datalog$^{\pm}$ semantics. Operationally, query answering is achieved bottom-up via the *chase* in Datalog$^{\pm}$, while in the ALP framework it is supported by the $\mathcal{S}$CIFF proof procedure. $\mathcal{S}$CIFF is able to integrate a knowledge base KB, expressed in terms of Logic Programming clauses, possibly with abducibles in their body, and to deal with integrity constraints.

To our purposes, we consider only $\mathcal{S}$CIFF programs with an empty KB, ICs with only conjunctions of positive expectations and CLP constraints (or $false$) in their heads. We show that this subset of the language suffices to represent Datalog$^{\pm}$ ontologies.

We map the finite set of relation names of a Datalog$^{\pm}$ relational schema $\mathcal{R}$ into the set of predicates of the corresponding $\mathcal{S}$CIFF program.

[5] In the equations (3)-(7) the set $\mathbf{HAP}$ should be substituted with $\mathbf{HAP'}$.

Definition 4.1. The τ mapping is recursively defined as follows, where A is an atom, $\mathbf{M}$ can be either $\mathbf{H}$ or $\mathbf{E}$, and F_1, F_2, ... are formulae ($\tau_{\mathbf{M}}$ stands for both $\tau_{\mathbf{H}}$ and $\tau_{\mathbf{E}}$, depending on what one wants to map) :

$$\begin{aligned}
\tau(body \rightarrow head) &= \tau_{\mathbf{H}}(body) \rightarrow \tau_{\mathbf{E}}(head) \\
\tau_{\mathbf{H}}(A) &= \mathbf{H}(A) \\
\tau_{\mathbf{E}}(A) &= \mathbf{E}(A) \\
\tau_{\mathbf{M}}(F_1 \wedge F_2) &= \tau_{\mathbf{M}}(F_1) \wedge \tau_{\mathbf{M}}(F_2) \\
\tau_{\mathbf{M}}(false) &= false \\
\tau_{\mathbf{M}}(Y_i = Y_j) &= Y_i = Y_j \\
\tau_{\mathbf{E}}(\exists \mathbf{X}\ A) &= \tau_{\mathbf{E}}(A)
\end{aligned}$$

□

The last rule means that in $\mathcal{S}$CIFF syntax it is not necessary to add explicitly the quantification of a variable. Note, however, that the existential quantification is correctly retained (see quantification rules in Section 3.1).

A Datalog$^{\pm}$ database D for $\mathcal{R}$ corresponds to the (possibly infinite) $\mathcal{S}$CIFF event set $\mathbf{HAP}$, since there is a one-to-one correspondence between each tuple in D and each (ground) fact in $\mathbf{HAP}$. This mapping is denoted as $\mathbf{HAP} = \tau_{\mathbf{H}}(D)$.

A Datalog TGD F of the kind $body \rightarrow head$ is mapped into the $\mathcal{S}$CIFF integrity constraint $IC = \tau(F)$, where the $body$ is mapped into conjunctions of $\mathcal{S}$CIFF atoms, and $head$ into conjunctions of $\mathcal{S}$CIFF abducible atoms. Existential quantifications of variables occurring in the $head$ of the TGD are maintained in the head of the $\mathcal{S}$CIFF IC while the rest of the variables are universally quantified with scope the entire IC.

Given a set of TGDs T_T, let us denote the mapping of T_T into the corresponding set $\mathcal{IC}$ of $\mathcal{S}$CIFF integrity constraints, as $\mathcal{IC} = \tau(T_T)$.

Recall that for a set of TGDs T_T on $\mathcal{R}$, and a database D for $\mathcal{R}$, the set of models of D given T_T, denoted $mods(D, T_T)$, is the set of all (possibly infinite) databases B such that $D \subseteq B$ and every $F \in T_T$ is satisfied in B. For any such database B, we can prove that there exists an abductive explanation[6] $\mathbf{EXP} = \tau_{\mathbf{E}}(B)$, $\mathbf{HAP'} = \tau_{\mathbf{H}}(B)$ such that:

$$\mathbf{HAP'} \cup \mathbf{EXP} \models \mathcal{IC}$$

where $\mathbf{HAP'} \supseteq \mathbf{HAP} = \tau_{\mathbf{H}}(D)$, and $\mathcal{IC} = \tau(T_T)$.

Finally, Datalog$^{\pm}$ negative constraints NCs are mapped into $\mathcal{S}$CIFF ICs with head $false$, and equality-generating dependencies EGDs into $\mathcal{S}$CIFF ICs, each one with an equality CLP constraint in its head.

A Datalog$^{\pm}$ CQ $q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$ over $\mathcal{R}$ is mapped into a $\mathcal{S}$CIFF goal $G = \tau_{\mathbf{E}}(\Phi(\mathbf{X}, \mathbf{Y}))$, where $\tau_{\mathbf{E}}(\Phi(\mathbf{X}, \mathbf{Y}))$ is a conjunction of $\mathcal{S}$CIFF atoms. Notice that in the $\mathcal{S}$CIFF framework we have therefore a goal with existential variables only, and among them, we are interested in computed answer substitutions for the original (tuple of) variables $\mathbf{X}$ (and therefore $\mathbf{Y}$ variables can be made anonymous).

A Datalog$^{\pm}$ BCQ $q = \Phi(\mathbf{Y})$ is mapped similarly: $G = \tau_{\mathbf{E}}(\Phi(\mathbf{Y}))$.

[6] With an abuse of notation, when S is a set we denote with $\tau(S) = \{\tau(s) | s \in S\}$.

Recall that in Datalog$^{\pm}$ the set of answers to a CQ q on D given T_T, denoted $ans(q, D, T_T)$, is the set of all tuples $\mathbf{t}$ such that $\mathbf{t} \in q(B)$ for all $B \in mods(D, T_T)$. With abuse of notation, we will write $q(\mathbf{t})$ to mean answer $\mathbf{t}$ for q on D given T_T.

We can hence state the following theorems for (model-theoretic) completeness of query answering.

Theorem 4.2. (Completeness of query answering)
Let T a set of implications (consisting of Tuple-Generating Dependencies (TGDs), Equality Generating Dependencies (EGDs) and/or Negative Constraints (NCs)). For each answer $q(\mathbf{t})$ of a CQ $q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$ on D given T, in the corresponding $\mathcal{S}$CIFF program $\langle \emptyset, \mathcal{IC} \rangle$ there exists an answer substitution θ and an abductive explanation $\mathbf{EXP} \cup \mathbf{HAP'}$ for goal $G = \tau_{\mathbf{E}}(\Phi(\mathbf{X}, \_))$ such that:

$$\langle \emptyset, \mathcal{IC} \rangle \models^{g}_{\mathbf{HAP}} G\theta$$

where $\mathbf{HAP} = \tau_{\mathbf{H}}(D)$, $\mathcal{IC} = \tau(T)$, and $G\theta = \tau_{\mathbf{E}}(\Phi(\mathbf{t}, \_))$. □

Proof:
Consider an answer $q(\mathbf{t})$ of a CQ $q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$ on D given T, and a database $B \in mods(D, T)$. By definition, every $F \in T$ is satisfied in B.

Consider now the sets $\mathbf{HAP'} = \tau_{\mathbf{H}}(B)$ and $\mathbf{EXP} = \tau_{\mathbf{E}}(B)$.

Let F be a TGD, a NC or an EGD in T; in general, let $F \equiv (Body(\mathbf{X}, \mathbf{Y}) \rightarrow Head(\mathbf{Y}, \mathbf{Z}))$. F is satisfied in B; i.e., for each homomorphism h such that $h(Body(\mathbf{X}, \mathbf{Y})) \subseteq B$ there exists an extension h' such that $h'(Head(\mathbf{Y}, \mathbf{Z})) \subseteq B$. Consider

$$\begin{aligned} & h'(\tau(Body(\mathbf{X}, \mathbf{Y}) \rightarrow Head(\mathbf{Y}, \mathbf{Z}))) \\ =& \tau_{\mathbf{H}}(h'(Body(\mathbf{X}, \mathbf{Y}))) \rightarrow \tau_{\mathbf{E}}(h'(Head(\mathbf{Y}, \mathbf{Z}))) \end{aligned}$$

Since $h'(Body(\mathbf{X}, \mathbf{Y})) \subseteq B$,

$$\tau_{\mathbf{H}}(h'(Body(\mathbf{X}, \mathbf{Y}))) \subseteq \tau_{\mathbf{H}}(B) = \mathbf{HAP'}$$

Analogously, since $h'(Head(\mathbf{Y}, \mathbf{Z})) \subseteq B$,

$$\tau_{\mathbf{E}}(h'(Head(\mathbf{Y}, \mathbf{Z}))) \subseteq \tau_{\mathbf{E}}(B) = \mathbf{EXP}$$

showing that $\tau(F)$ is satisfied. Since for every $F \in T$ there is one integrity constraint $\tau(F) \in \mathcal{IC}$, we have that

$$\mathbf{HAP'} \cup \mathbf{EXP} \models \mathcal{IC}. \tag{8}$$

Since $q(\mathbf{t})$ is an answer of the CQ $q(\mathbf{X}) = \Phi(\mathbf{X}, \mathbf{Y})$, there exists a homomorphism $\mu : \mathbf{X} \cup \mathbf{Y} \rightarrow \Delta \cup \Delta_N$ such that $\mu(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq B$ and $\mu(\mathbf{X}) = \mathbf{t}$.

Consider the substitution $\theta = \{\mathbf{X}/\mathbf{t}, \mathbf{Y}/\mu(\mathbf{Y})\}$.

$$\mu(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq B$$

implies that

$$\tau_{\mathbf{E}}(\mu(\Phi(\mathbf{X}, \mathbf{Y}))) \subseteq \tau_{\mathbf{E}}(B) = \mathbf{EXP}$$

from which we have

$$\mathbf{EXP} \models \tau_{\mathbf{E}}(\mu(\Phi(\mathbf{X}, \mathbf{Y})))$$

and from the definition of θ

$$\mathbf{EXP} \models \tau_{\mathbf{E}}(\Phi(\mathbf{X}, \mathbf{Y})\theta)$$

and obviously

$$\mathbf{HAP} \cup \mathbf{EXP} \models \tau_{\mathbf{E}}(\Phi(\mathbf{X}, \mathbf{Y})\theta).$$

By construction of $\mathbf{HAP'}$ and $\mathbf{EXP}$ we immediately have that

$$\mathbf{HAP'} \cup \mathbf{EXP} \models \mathbf{E}(X) \rightarrow \mathbf{H}(X);$$

moreover since $\mathbf{EXP}$ does not contain $\mathbf{EN}$ literals, we also have that

$$\mathbf{HAP'} \cup \mathbf{EXP} \models \mathbf{EN}(X) \wedge \mathbf{H}(X) \rightarrow \mathit{false}$$

and

$$(\mathbf{HAP'} \cup)\ \mathbf{EXP} \models \mathbf{E}(X) \wedge \mathbf{EN}(X) \rightarrow \mathit{false}$$

□

Corollary 4.3. (Completeness of boolean query answering)
If the answer to a BCQ $q = \exists \mathbf{Y} \Phi(\mathbf{Y})$ over D given T is Yes, denoted $D \cup T \models q$, then in the corresponding $\mathcal{S}$CIFF program there exists an abductive explanation $\mathbf{EXP} \cup \mathbf{HAP'}$ such that:

$$\langle \emptyset, \mathcal{IC} \rangle \models^{g}_{\mathbf{HAP}} G$$

where $\mathbf{HAP} = \tau_{\mathbf{H}}(D)$, $\mathcal{IC} = \tau(T)$, and $G = \tau_{\mathbf{E}}(\Phi(\_))$. □

The soundness of the translation must face the fact that the semantics of abductive logic programming and of Datalog$^{\pm}$ are different. Declaratively, Datalog$^{\pm}$ considers all the models $B \in mods(D, T_T)$, and a Boolean Conjunctive Query (BCQ) is true iff it is true in all B, while in ALP a goal is true if there is (at least) an abductive answer.

Example 4.4. Consider the set T_T consisting of the rules

$$p(X, Y) \rightarrow q(A, Y).$$
$$p(X, Y), q(Y, Z) \rightarrow r(X, Y, Z).$$

and a database $D = \{p(1, 2)\}$.

Amongst the models, we have $B_1 = \{p(1, 2), q(2, 2), r(1, 2, 2)\}$ and $B_2 = \{p(1, 2), q(7, 2)\}$; the BCQs $p(1, 2)$ and $\exists X\ q(X, 2)$ are true, while $q(2, 2)$ and $\exists A, B, C\ r(A, B, C)$ are false.

In ALP, instead (assuming that predicates q and r are abducible), there are various ground abductive answers that can be summarized into two cases: either $Y = 2$ or $Y \neq 2$, and in the first case $r(1, 2, 2)$ must be true. □

It makes sense to study the set of programs for which both ALP and Datalog provide the same semantics; we give the following definition:

Definition 4.5. (propagation-safe programs)
Let D be a database and T a set of implications (consisting of TGDs, EGDs and/or NCs).

For each predicate symbol p, we have a_p *positions*, if a_p is the arity of the predicate. We write positions with the syntax $p[i]$, to indicate the i-th argument of predicate symbol p.

Consider the following marking procedure.

1. (Preliminary step): for each variable X occurring more than once in the body of a rule, mark all the positions in which X occurs in that rule;

2. if in a rule a variable occurs in a position that is marked, mark all the positions in which the variable occurs.

Repeat step 2 until a fixed point is reached.

If

- for each atom $p(\mathbf{t})$ in the database, for each marked position $p[i]$, $\mathbf{t}[i] \in \Delta$ (i.e., the term in position $p[i]$ is not null)

- for each atom $p(\mathbf{X})$ in the head of a TGD, for each marked position $p[i]$, $\mathbf{X}[i]$ is not an existentially quantified variable

then we say that T and D are *propagation-safe*.

Note that the program in Example 4.4 is not propagation-safe because the position $q[1]$ is marked, but variable A, in the head of the first rule, is existentially quantified, violating the second condition for propagation-safeness. In a propagation-safe program, for all databases obtained by applying the chase to D, in the marked positions there cannot be null values, i.e., in the marked positions only constants can occur. In particular, in *join* positions only constant (and no nulls, nor existential variables) can occur.

Theorem 4.6. (Soundness of propagation-safe programs)
Given a database D, let $\mathbf{HAP} = \tau_{\mathbf{H}}(D)$. Let $\langle \emptyset, \tau(T) \rangle$ be the corresponding $\mathcal{S}$CIFF program of a set of propagation-safe rules T.

Let $q = \exists X \Phi(X)$ be a BCQ.

If

$$\langle \emptyset, \mathcal{IC} \rangle \vdash^{g}_{\mathbf{HAP} \rightsquigarrow \mathbf{HAP}'} true$$

and

$$\mathbf{HAP}' \models \tau(q)$$

then the BCQ q over D given T is true.

Proof:
The proof is based on the fact that in propagation-safe programs, the atoms generated by the chase contain only constants in the the *join positions*. For such programs, the $\mathcal{S}$CIFF proof-procedure executes the same steps as the chase, that is sound.

Let

$$b_1(\mathbf{X_1}), b_2(\mathbf{X_2}), \ldots b_n(\mathbf{X_n}) \rightarrow \exists \mathbf{Y} h(\mathbf{X}', \mathbf{Y}) \tag{9}$$

be a TGD, where without loss of generality we consider a single atom in the head, and in which $\mathbf{X}' \subseteq \cup_i \mathbf{X_i}$.

In this case, the corresponding IC through the τ-mapping is

$$\mathbf{H}(b_1(\mathbf{X_1})), \mathbf{H}(b_2(\mathbf{X_2})), \ldots \mathbf{H}(b_n(\mathbf{X_n})) \rightarrow [\exists \mathbf{Y}]\mathbf{E}(h(\mathbf{X}', \mathbf{Y})) \tag{10}$$

The TGD generates $h(\mathbf{X}', \mathbf{Y})$ iff the body is true, i.e., if there exists a set of atoms

$$b_1(\mathbf{a_1}), \ldots, b_n(\mathbf{a_n}) \tag{11}$$

matching the body. The $\mathcal{S}$CIFF proof-procedure will trigger IC (10) iff the body is true, i.e., if there exists a set of atoms

$$\mathbf{H}(b_1(\mathbf{a_1})), \ldots, \mathbf{H}(b_n(\mathbf{a_n})) = \tau(b_1(\mathbf{a_1}), \ldots, b_n(\mathbf{a_n})) \tag{12}$$

matching the body.

In fact, let us assume there exist atoms (11) in D and (12) in $\mathbf{HAP}$. Let us assume w.l.o.g., that the $\mathcal{S}$CIFF proof procedure selects the atoms in the order 1 to n. From (12) and $\mathbf{H}(b_1(\mathbf{a_1}))$, propagation can be applied obtaining

$$\mathbf{X_1} = \mathbf{a_1}, \mathbf{H}(b_2(\mathbf{X_2})), \ldots \mathbf{H}(b_n(\mathbf{X_n})) \rightarrow [\exists \mathbf{Y}]\mathbf{E}(h(\mathbf{X}', \mathbf{Y}))$$

from which case analysis is applied obtaining two disjuncts:

- $\mathbf{X_1} = \mathbf{a_1}$ and $\mathbf{H}(b_2(\mathbf{X_2})), \ldots \mathbf{H}(b_n(\mathbf{X_n})) \rightarrow [\exists \mathbf{Y}]\mathbf{E}(h(\mathbf{X}', \mathbf{Y}))$
- $\mathbf{X_1} \neq \mathbf{a_1}$.

As $\mathbf{X_1}$ is universally quantified, the latter is rewritten to $false$, while in the first $\mathbf{X_1}$ is unified with $\mathbf{a_1}$ and the implication is rewritten into

$$\mathbf{H}(b_2(\mathbf{X_2})), \ldots \mathbf{H}(b_n(\mathbf{X_n})) \rightarrow [\exists \mathbf{Y}]\mathbf{E}(h(\mathbf{X}', \mathbf{Y})) \quad [\mathbf{X_1}/\mathbf{a_1}]$$

If the set of atoms (11) does not contain any null, since we assumed that the set of atoms (11) unifies with the body of (9), clearly the same sequence (*propagation*, *case analysis* and *equality rewriting*) is applicable for all atoms in the body of (10).

From the previous observation, the set of atoms cannot contain any null in the join positions, meaning that either the shared constant is the same (and in this case the $\mathcal{S}$CIFF proof procedure can take only the first branch, while the second immediately fails, and the chase would fire as well) or the constants are different (and in this case the $\mathcal{S}$CIFF proof procedure can take only the second branch, while the first immediately fails - the implication does not fire in the chase as well).

After n applications of *propagation*, *case analysis* and *equality rewriting*, we obtain

$$true \rightarrow [\exists \mathbf{Y}]\mathbf{E}(h(\mathbf{X}', \mathbf{Y})) \quad [\theta]$$

for some substitution θ; *logical equivalence* can be applied obtaining

$$[\exists \mathbf{Y}]\mathbf{E}(h(\mathbf{X}', \mathbf{Y})) \quad [\theta].$$

Transition *regimentation* can now be applied, generating

$$\mathbf{H}(h(\mathbf{X}', \mathbf{Y})) \quad [\theta]$$

that is equivalent (through the τ-mapping) to the atom $\exists \mathbf{Y}\ h(\mathbf{X}', \mathbf{Y})$ obtained by the chase.

□

Example 4.7. (Real estate information extraction system in ALP)
Let us conclude this section by re-considering the Datalog$^{\pm}$ ontology for the real estate information extraction system of Example 2.1. TGDs F_1-F_8 are one-to-one mapped into the following $\mathcal{S}$CIFF ICs:

$$\mathbf{H}(ann(X, label)), \mathbf{H}(ann(X, price)), \mathbf{H}(visible(X)) \rightarrow \mathbf{E}(priceElem(X)) \quad (IC_1)$$
$$\mathbf{H}(ann(X, label)), \mathbf{H}(ann(X, priceRange)), \mathbf{H}(visible(X)) \rightarrow \mathbf{E}(priceElem(X)) \quad (IC_2)$$
$$\mathbf{H}(priceElem(E)), \mathbf{H}(group(E, X)) \rightarrow \mathbf{E}(forSale(X)) \quad (IC_3)$$
$$\mathbf{H}(forSale(X)) \rightarrow (\exists P)\ \mathbf{E}(price(X, P)) \quad (IC_4)$$
$$\mathbf{H}(hasCode(X, C)), \mathbf{H}(codeLoc(C, L)) \rightarrow \mathbf{E}(loc(X, L)) \quad (IC_5)$$
$$\mathbf{H}(hasCode(X, C)) \rightarrow (\exists L)\ \mathbf{E}(codeLoc(C, L)), \mathbf{E}(loc(X, L)) \quad (IC_6)$$
$$\mathbf{H}(loc(X, L1)), \mathbf{H}(loc(X, L2)) \rightarrow L1 = L2 \quad (IC_7)$$
$$\mathbf{H}(loc(X, L)) \rightarrow \mathbf{E}(advertised(X)) \quad (IC_8)$$

The database is then simply mapped into the following event set $\mathbf{HAP}$:

$$\{\mathbf{H}(codeLoc(ox1, central)), \mathbf{H}(codeLoc(ox1, south)),$$
$$\mathbf{H}(codeLoc(ox2, summertown)), \mathbf{H}(hasCode(prop1, ox2)), \mathbf{H}(ann(e1, price)),$$
$$\mathbf{H}(ann(e1, label)), \mathbf{H}(visible(e1)), \mathbf{H}(group(e1, prop1))\}$$

Note that the program is propagation-safe.

The $\mathcal{S}$CIFF proof procedure applies ICs in a forward manner, and it infers the following set of abducibles from the program above:

$$\begin{aligned}\mathbf{EXP} \quad = \quad & \{\mathbf{E}(priceElem(e1)), \mathbf{E}(forSale(prop1)), \exists P\ \mathbf{E}(price(prop1, P)),\\ & \mathbf{E}(loc(prop1, summertown)), \mathbf{E}(advertised(prop1))\}\end{aligned}$$

plus the corresponding $\mathbf{H}$ atoms, that are not reported for the sake of brevity.

By focusing on a single IC, say IC_6, it can be inferred from the event set $\mathbf{HAP}$ the set of abducibles $\{(\exists L)\ \mathbf{E}(codeLoc(ox2, L)), \mathbf{E}(loc(prop1, L))\}$, by exploiting propagation, case analysis, equality rewriting and logical equivalence.

Each of the (ground) atomic queries of Example 2.1 is entailed in the $\mathcal{S}$CIFF program above, since there exist sets $\mathbf{EXP}$ and $\mathbf{HAP}'$ such that:

$$\mathbf{HAP}' \cup \mathbf{EXP} \models \mathbf{E}(priceElem(e1)), \mathbf{E}(forSale(prop1)), \mathbf{E}(advertised(prop1))$$

The query $\exists L\ \mathbf{E}(loc(prop1, L))$ is entailed as well, considering the unification $L = summertown$ since:

$$\mathbf{HAP}' \cup \mathbf{EXP} \models \mathbf{E}(loc(prop1, summertown)).$$

Note that, if there was no IC_7, $\mathcal{S}$CIFF would provide

$$\mathbf{EXP}' = \{\exists L\ \mathbf{E}(codeLoc(ox2, L))\} \cup \mathbf{EXP}$$

i.e., there would be a further possible code for $ox2$. In the same situation, Datalog$^{\pm}$ would provide to the CQ $loc(prop1, L)$ the answers $summertown$ and a new NULL, call it $z1$. □

It is worth noting that the $\mathcal{S}$CIFF framework is much more expressive than the restricted version used in this paper; in fact, in the mapping we used an empty KB, but in general the Knowledge Base can be a logic program, that can include expectations, abducible literals, as well as CLP constraints. Beside the forward propagation of Integrity Constraints, $\mathcal{S}$CIFF supports also backward reasoning.

5. Related Work

Various approaches have been followed to reason upon ontologies. Usually, DL reasoners implement a tableau algorithm using a procedural language. Since some tableau expansion rules are non-deterministic, the developers have to implement a search strategy from scratch.

Pellet [3] is, until version 2, a free open-source Java-based reasoner. It is able to reason upon $\mathcal{SROIQ}$(**D**) with simple datatypes (i.e., for OWL 1.1). Pellet can compute the set of all the explanations for given queries by exploiting the tableau algorithm. An explanation here is roughly a subset of the knowledge base (KB) that is sufficient for entailing the query. The tableau algorithm starts with a so called tableau, which is a graph representing the assertional information contained in the KB. The tableau is expanded by applying a set of expansion rules in order to check if there is a model for the given query. However, the tableau algorithm can build a single explanation, therefore Pellet applies Reiter's *hitting set algorithm* [27] to find all the explanations. This is a black box method: Pellet repeatedly removes an axiom from the KB and then computes again a new explanation exploiting the tableau algorithm on the new KB, recording all the different explanations so found.

Differently from Pellet, reasoners written in Prolog can exploit Prolog's backtracking facilities for performing the search. This has been observed in various works. In [28, 29] the authors proposed a tableau reasoner in Prolog for First Order Logic (FOL) based on free-variable semantic tableaux. However, the reasoner is not tailored to DLs.

Hustadt, Motik and Sattler [30] 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 $\mathcal{SHIQ}$ KB into a disjunctive Datalog program, while DLog [31, 32] is an ABox reasoning algorithm for the $\mathcal{SHIQ}$ language that allows to store the content of the ABox externally in a database and to answer instance checking and instance retrieval queries by transforming the KB into a Prolog program.

Meissner presented the implementation of a reasoner for the DL $\mathcal{ALCN}$ written in Prolog [33], which was then extended and reimplemented in the Oz language [34]. Starting from [33], Herchenröder [35] implemented heuristic search techniques in order to reduce the inference time for the DL $\mathcal{ALC}$. Faizi [36] added to [35] the possibility of returning information about the steps executed during the inference process for queries, but still handled only $\mathcal{ALC}$.

A different approach is the one by Ricca et al. [37], that presented OntoDLV, a system for reasoning on a logic-based ontology representation language called OntoDLP. This is an extension of (disjunctive)

ASP and can interoperate with OWL. OntoDLV rewrites the OWL KB into the OntoDLP language, also by retrieving information directly from external OWL Ontologies, and answers queries by using ASP. Inside the DLV project there is also a system, called DLV$^{\exists}$, able to answer conjunctive queries over a fragment of Datalog$^{\exists}$ called "Shy", which allows existential quantifiers in rule heads. In [38], after defining Shy, the authors describe a bottom-up evaluation strategy for it which performs well even in dynamic scenarios, where data changes frequently. This strategy, implemented in DLV$^{\exists}$, exploits a modified version of the chase, called parsimonius-chase.

A different system for conjunctive query answering is PAGOdA [12]. It is tailored to OWL 2 and combines the well-known Hermit reasoner [5] with the Datalog reasoner RDFox [39]. The system provides scalable "pay-as-you-go" performance for the query computation by resorting on the Datalog reasoner for the major part of the inference process, and delegating to the usually slower OWL reasoner only when strictly necessary. Trivially, PAGOdA computes lower and upper bound answers for the given conjunctive queries using the Datalog reasoner. In case these bounds do not coincide, the Datalog reasoner is also used to create a subset of the KB sufficient to test the correctness of the tuples in the gap of the two bounds. The resulting KB is usually small and is given as input to the DL reasoner to compute the final answers.

However, both DLV$^{\exists}$ and PAGOdA are implemented for conjunctive query answering, thus they are not directly comparable with $\mathcal{S}$CIFF, which is focused on returning explanations for queries.

TRILL [7, 40] adopts a Prolog-based implementation for the tableau expansion rules for $\mathcal{ALC}$ description logics. Differently from previous reasoners, TRILL is also able to return explanations for the given queries. Moreover, TRILL differs in particular from DLog for the possibility of answering general queries instead of instance check and instance retrieval only.

The combination of DLs and LPs was also studied by Motik and Rosati, in [41]. They applied Minimal Knowledge with Negation as Failure, one of the most effective approaches presented in such field, to define hybrid knowledge bases. These KBs are defined as the combination of logic programs and DL ontologies. In this way, both open and closed world assumption can be exploited in the same KB, in a framework that can preserve the semantics of both formalisms when one is absent and exhibits good properties such as decidability. They also defined an extension of the SLG proof procedure, called SLG(O), making use of an oracle to manage the DL part. However, no implementation has been developed. Based on this approach, in [42] and more recently in [43] we presented probabilistic hybrid knowledge bases, were we extend the work of Motik and Rosati in order to cope with probability theory.

A similar idea was followed in [44], where the authors present a framework where they add on top of a DL KB a set of rules, called dl-rules. Such rules are similar to classical logic programming rules with negation as failure, but they may contain in their body queries to the DL part, possibly under default negation. A generalization of the Herbrand model semantics has been then defined for such rules. The objective is to compute answer sets for the KB with dl-rules working under the closed world assumption. Finally, an algorithm combining a DL reasoner with DLV is presented, in order to find answer sets by using both the DL part and dl-rules.

As reported in Section 2, reasoning upon Datalog$^{\pm}$ ontologies is achieved, instead, via the *chase* bottom-up procedure, which is exploited for deriving atoms entailed by a database and a Datalog$^{\pm}$ theory.

In this work, we apply an ALP proof-procedure to reason upon ontologic data. It is worth noticing that in a previous work [45] the $\mathcal{S}$CIFF proof-procedure was interfaced with Pellet to perform ontological reasoning; in the current work, instead, $\mathcal{S}$CIFF is directly used to perform reasoning by mapping atoms in the ontology to $\mathcal{S}$CIFF concepts (like events and expectations).

6. Experimental evaluation

In order to assess the practical usability of $\mathcal{S}$CIFF as an ontological reasoner, we devised a set of experiments, including both crafted and real instances. In the formers we wanted to stress the algorithms out by forcing them to perform an increasing number of backtracking on contructs commonly used when defining KBs, while in the latters we tested the algorithm on real world domains to test its applicability in concrete problems.

We compare $\mathcal{S}$CIFF with Pellet [3], a popular DL reasoner implemented in an object oriented language, and with TRILL and TRILL^P [46], two recent reasoners implemented in Prolog. We concentrated on these systems because, like $\mathcal{S}$CIFF, they are able to return a set containing all of the explanations for the query. Other systems, such as PAGOdA and DLV$^\exists$ cannot be directly compared with $\mathcal{S}$CIFF.

$\mathcal{S}$CIFF can run on top of SWI [47] or SICStus [48] Prolog. Usually, the SICStus version is faster; however, since TRILL and TRILL^P cannot be executed on SICStus, we decided to run all the experiments with SWI.

Translation from $\mathcal{S}$CIFF to DL and viceversa was performed by applying the standard definition as shown in [49]. All the experiments were performed on a Linux machine with a 3.10 GHz Intel Xeon E5-2687W.

6.1. Crafted instances

A first set of experiments was devoted to stress the non-determinism associated with the choice of which rule to apply. We artificially created a set of knowledge bases of increasing size, containing the following axioms:

$$\begin{gathered} C_{1,1}(X) \rightarrow C_{1,2}(X) \rightarrow ... \rightarrow C_{1,n}(X) \\ C_{1,1}(X) \rightarrow C_{2,2}(X) \rightarrow ... \rightarrow C_{2,n}(X) \\ C_{1,1}(X) \rightarrow C_{3,2}(X) \rightarrow ... \rightarrow C_{3,n}(X) \\ ... \\ C_{1,1}(X) \rightarrow C_{m,2}(X) \rightarrow ... \rightarrow C_{m,n}(X) \end{gathered} \tag{13}$$

An instance a is added to $C_{1,1}$, i.e., $C_{1,1}(a)$, and the goal is to check whether $C_{m,n}(a)$. The goal is clearly true, but it stresses the ontological reasoner as there are a number of paths that must be explored in order to find the one satisfying the query.

Figure 1 shows the time necessary to compute all the explanations when $m = 40$ and n varies from 5 to 40. We also did experiments to find one solution; the graphs are qualitatively similar, and the ranking of the systems was the same. Each instance was run 50 times and the average time was taken.

The running time of $\mathcal{S}$CIFF is higher than that of Pellet; however it is of the same order of magnitude and is competitive with TRILL and TRILL^P.

Figure 2 shows analogous results when varying m and keeping constant n=40.

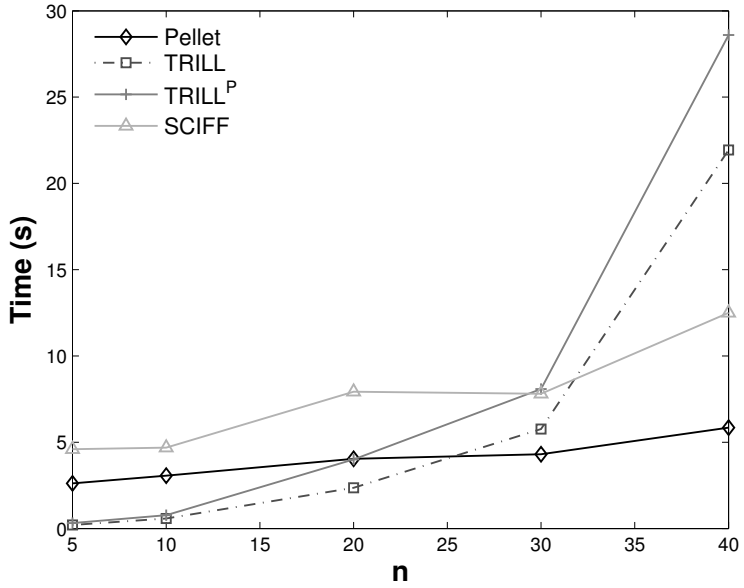

Figure 1. Running time for computing all explanations in the knowledge base in equation 13 with $m = 40$ and n varying from 5 to 40.

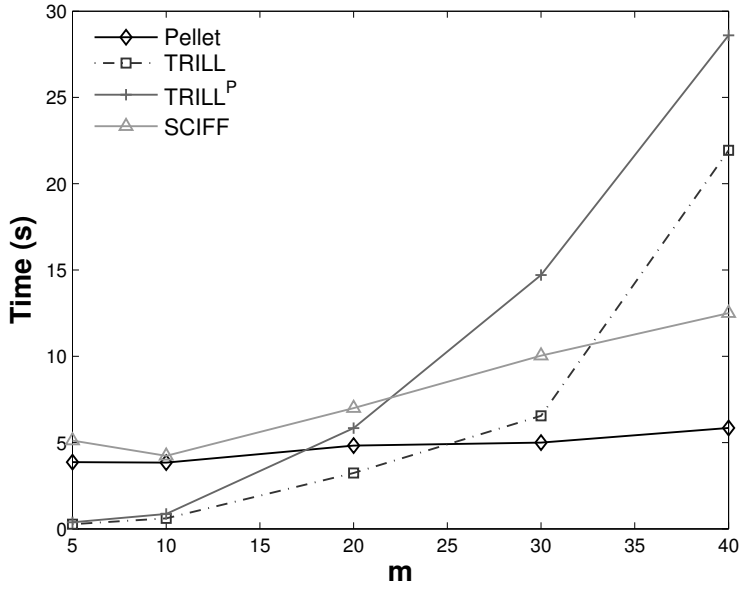

Figure 2. Running time for computing all explanations in the knowledge base in equation 13 with $n = 40$ and m varying from 5 to 40.

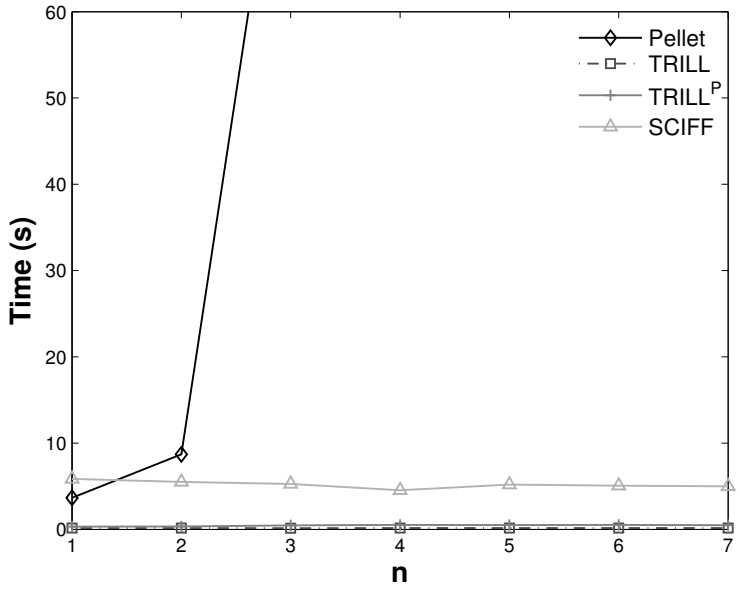

Figure 3. Running time for computing all explanations in the knowledge base in equation 14 with $m = 7$ and n varying from 1 to 7.

We also developed a variation of the previous experiment, in which the axioms are defined as follows:

$$\begin{aligned} &C_{1,1}(X) \rightarrow C_{1,2}(X) \rightarrow ... \rightarrow C_{1,n}(X) \rightarrow C_{n+1}(X) \\ &C_{1,1}(X) \rightarrow C_{2,2}(X) \rightarrow ... \rightarrow C_{2,n}(X) \rightarrow C_{n+1}(X) \\ &C_{1,1}(X) \rightarrow C_{3,2}(X) \rightarrow ... \rightarrow C_{3,n}(X) \rightarrow C_{n+1}(X) \\ &\qquad\qquad ... \\ &C_{1,1}(X) \rightarrow C_{m,2}(X) \rightarrow ... \rightarrow C_{m,n}(X) \rightarrow C_{n+1}(X) \end{aligned} \tag{14}$$

i.e., we have a further class C_{n+1} which subsumes all classes. The query, in this case, was to check if a belongs to C_{n+1}, i.e., $C_{n+1}(a)$. The results are shown in Figure 3 for a fixed $m = 7$ varying n, and in Figure 4 for a fixed n and varying m.

In both cases, when the size of the KB grows large, the runtime of Pellet exceeded the allowed timeout (that was fixed for all experiments to 600s) because of the algorithm used for the exploration of the search space. This algorithm, based on Reiter's *hitting set algorithm* [27], repeatedly removes axioms from the KB in order to try to find new explanations. $\mathcal{S}$CIFF takes more time than TRILL and TRILL^P, but it remains within reasonable running times.

6.2. Real instances

We also experimented on real instances; we considered some benchmarks taken from [46], in which we do not consider the probabilistic axioms.

The considered benchmarks are real examples from DL knowledge bases:

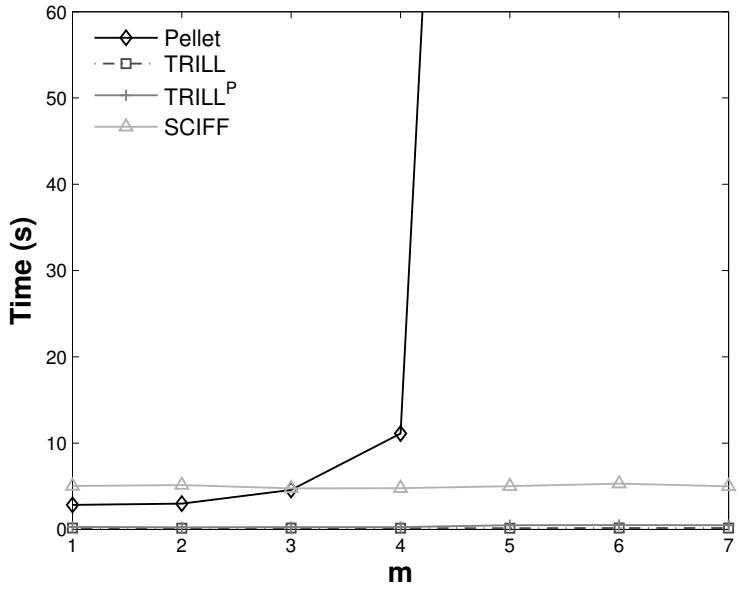

Figure 4. Running time for computing all explanations in the knowledge base in equation 14 with $n = 7$ and m varying from 1 to 7.

- an extract of the DBpedia ontology obtained from Wikipedia;
- BioPAX level 3, which models metabolic pathways;
- VICODI, which contains information on European history.

DBpedia [50] contains structured information defined in the sideboxes in the pages of Wikipedia. It has expressiveness $\mathcal{EL}$ and contains 267 axioms and 118 classes.

BioPAX [51] represents molecular and genetic interactions together with pathways including molecular states. It is part of the BioPAX project, which standardizes data definition of analysis of biological pathways and defines 3 different levels modeling different interactions. The version of BioPAX used was level 3, with expressiveness $\mathcal{SHIN}(\mathbf{D})$, has 925 axioms, 69 classes, 55 object properties and 41 data properties.

Finally, VICODI [52] is an extract of the VICODI knowledge base that contains information on historical events and important personalities of European history. VICODI's expressiveness is $\mathcal{ALH}(\mathbf{D})$, it contains 209 axioms, 168 classes, 6 object properties and 2 data properties.

However, the expressivity of Datalog$^{\pm}$ cannot be directly compared with those of DLs, which can be in some cases undecidable. Therefore, we concentrated on fragments of such KBs which are translatable to Datalog$^{\pm}$. It is worth noting that all the obtained KBs are propagation-safe. Among the resulting knowledge bases, DBpedia and VICODI do not have existentially quantified variables in rules head, conversely, BioPAX contains axioms corresponding to existential rules.

For the DBpedia and BioPAX datasets, we created 100 subclass-of queries, while for VICODI we created 80 subclass-of and 20 instance-of queries. To ensure that each query had at least one explanation,

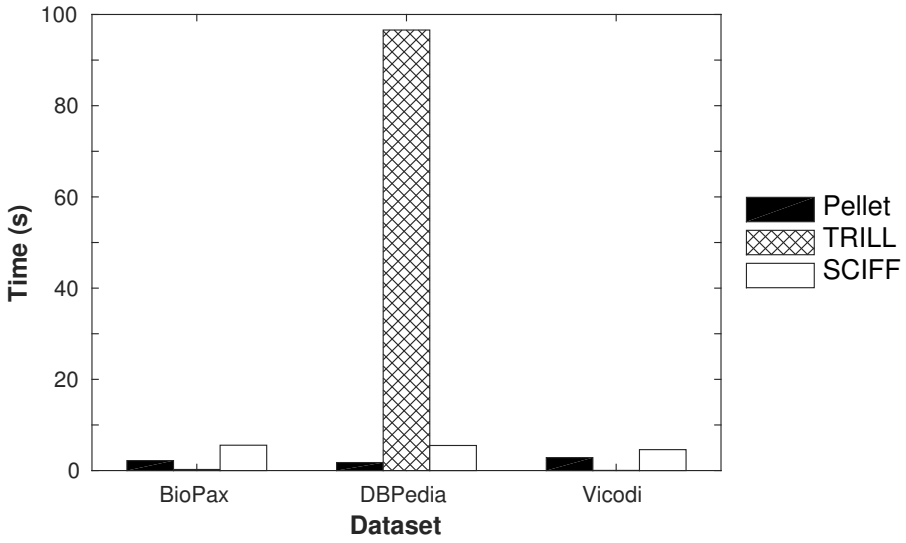

Figure 5. Running time for computing the first explanation in the real benchmarks.

we generated them by computing the hierarchy of classes. After that, for subclass-of queries we randomly selected two classes that are connected in the hierarchy, while for the instance-of queries we randomly selected an individual a and a class to which a belongs by following the hierarchy, starting from the classes to which a explicitly belongs in the KB.

Figure 5 shows the mean running time necessary to compute the first explanation on the three benchmarks computed on all the created queries. Pellet is the best, $\mathcal{S}$CIFF is slower but always within reasonable bounds, while the performance of TRILL depends significantly on the considered benchmark: in VICODI and BioPAX it is the fastest, while on DBpedia it is by far the slowest. In this test we did not use TRILL^P since it computes a Boolean formula representing the set of all explanations, it cannot return just a single explanation.

Similar considerations can be done when considering the problem of finding all explanations in the three benchmarks (see Figure 6). In this second test we ran also TRILL^P. Note that in DBpedia $\mathcal{S}$CIFF is the best performing one.

Finally, the required memory is plotted for all experiments in Figures 7 (for finding one solution) and 8 (for finding all solutions). Here we can note how the memory requirements for Pellet (that is Java-based) are much higher, as expected, than for the other systems, that are all Prolog-based.

7. Conclusions and Future Work

In this paper, we addressed representation and reasoning for Datalog$^{\pm}$ ontologies in an Abductive Logic Programming framework, with existential (and universal) variables, and Constraint Logic Programming constraints in rule heads. The underlying proof procedure, named $\mathcal{S}$CIFF, is inspired by the IFF proof procedure, and had been implemented in Constraint Handling Rules [53]. The $\mathcal{S}$CIFF system has already

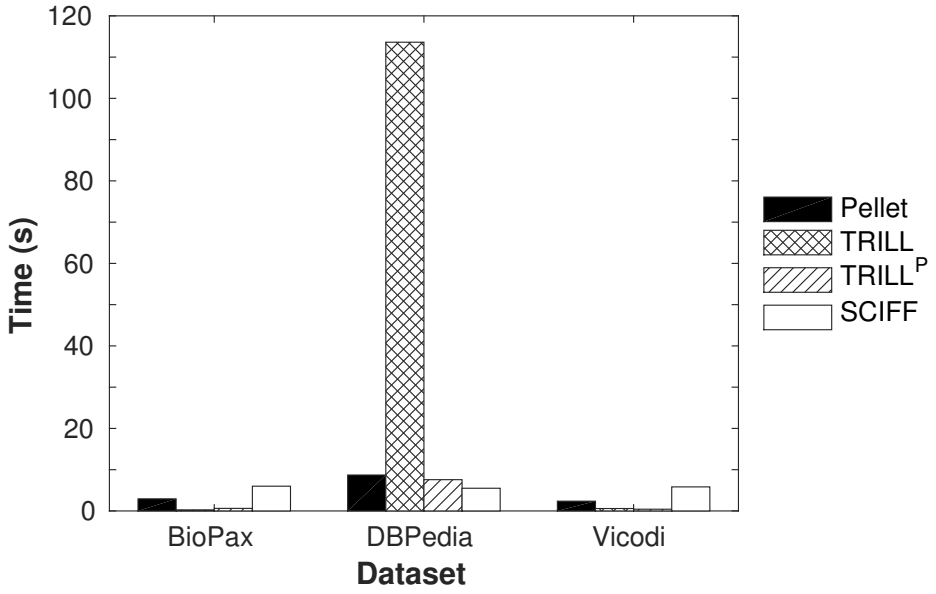

Figure 6. Running time for computing all explanations in the real benchmarks.

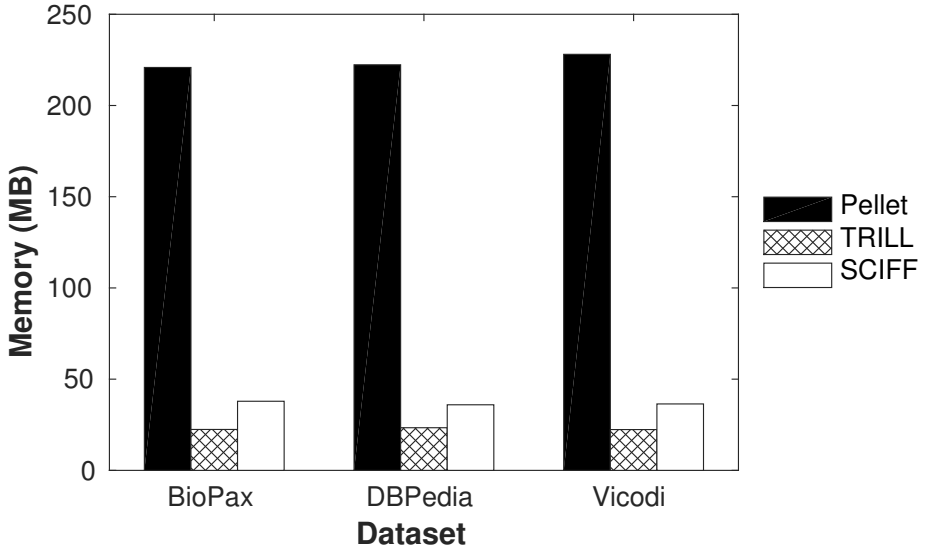

Figure 7. Memory requirements for computing the first explanation in the real benchmarks (Maximum resident set size of the process during its lifetime, in MBytes).

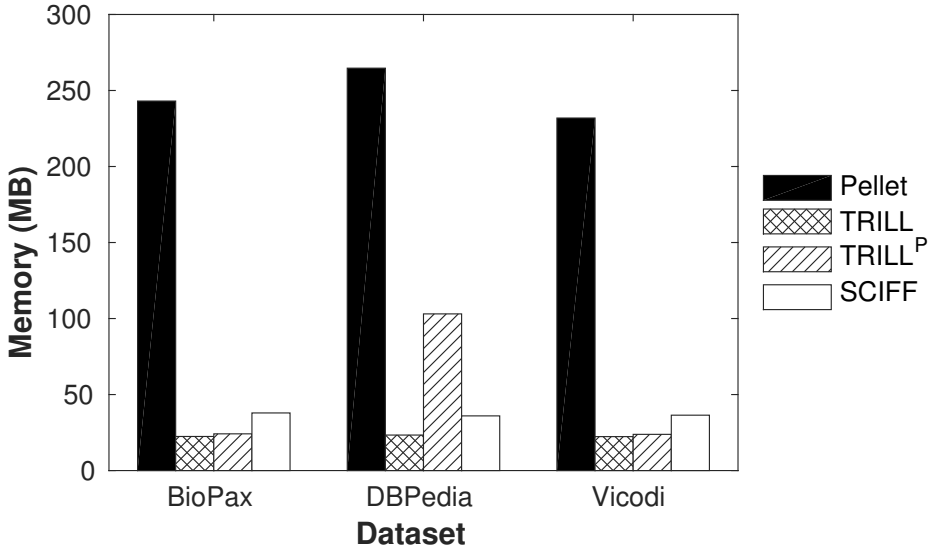

Figure 8. Memory requirements for computing all explanations in the real benchmarks (Maximum resident set size of the process during its lifetime, in MBytes).

been used for modeling and implementing several knowledge representation frameworks, also providing an effective reasoning system.

Here we have considered Datalog$^{\pm}$ ontologies, and shown how the $\mathcal{S}$CIFF language can be a useful knowledge representation and reasoning framework for them. In fact, the underlying abductive proof procedure can be directly exploited as an ontological reasoner for query answering and consistency checking. To the best of our knowledge, this is the first application of ALP to model and reason upon ontologies.

Moreover, the considered $\mathcal{S}$CIFF language smoothly supports the integration of rules, expressed in a Logic Programming language, with ontologies expressed in Datalog$^{\pm}$, since a logic program can be added as (non-empty) KB to the set of ICs, therefore considering deductive rules besides the forward rules themselves. Furthermore, $\mathcal{S}$CIFF allows the expression of existential quantifiers in rules head. The integration with rules has gained increasing attention in the last years, as evidenced by all the proposals to combine rules and ontologies, such as OWL2 RL, RIF and SWRL. $\mathcal{S}$CIFF fits in with this objective.

Syntactic conditions over Datalog$^{\pm}$ programs for guaranteeing decidability and nice computational complexity results, such as tractability, have been extensively studied; a good overview can be found in [54]. As shown in this paper, the fragment of the Datalog$^{\pm}$ language in which such syntactic conditions are met can be mapped to a $\mathcal{S}$CIFF program, thus the same properties hold also for the same fragment of $\mathcal{S}$CIFF programs. This does not mean that the $\mathcal{S}$CIFF proof-procedure will have nice tractability properties on such a fragment. However, we showed experimentally that the $\mathcal{S}$CIFF approach is a viable one for reasoning on semantic web data, and in particular $\mathcal{S}$CIFF has comparable performances to those of existing DL reasoners based on the tableau algorithm. This has been shown by testing it over real datasets downloaded from DL repositories.

Note that the $\mathcal{S}$CIFF language is richer than the subset here used to represent Datalog$^{\pm}$ ontologies. It

can support, in fact, negative expectations in rule heads, with universally quantified variables too, which basically represent the fact that something ought not to happen, and the proof procedure can identify violations of them.

Therefore, the richness of the language, and the potential of its abductive proof procedure pave the way to add further features to Datalog$^{\pm}$ ontologies.

Future work includes possibly extending the proof procedure to probabilistic reasoning in order to compute the probability of the truth value of the queries, along the line shown in [55].

Moreover, we can use machine learning to learn $\mathcal{S}$CIFF programs and possibly the parameters from real data. A possible way to learn is to exploit Limited-memory BFGS (L-BFGS) [56] for tuning the parameters and constraint refinements for finding good structures. L-BFGS is an optimization algorithm in the family of quasi-Newton methods that approximates the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm using a limited amount of computer memory.

References

[1] Hitzler P, Krötzsch M, Rudolph S. Foundations of Semantic Web Technologies. CRCPress; 2009.

[2] Calvanese D, Giacomo GD, Lembo D, Lenzerini M, Rosati R. Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. J Autom Reasoning. 2007;39(3):385–429.

[3] Sirin E, Parsia B, Cuenca-Grau B, Kalyanpur A, Katz Y. Pellet: A practical OWL-DL reasoner. Journal of Web Semantics. 2007;5(2):51–53.

[4] Haarslev V, Hidde K, Möller R, Wessel M. The RacerPro knowledge representation and reasoning system. Semantic Web. 2012;3(3):267–277.

[5] Glimm B, Horrocks I, Motik B, Stoilos G, Wang Z. HermiT: An OWL 2 Reasoner. J Autom Reasoning. 2014;53(3):245–269. Available from: `http://dx.doi.org/10.1007/s10817-014-9305-1`. doi:10.1007/s10817-014-9305-1.

[6] Schmidt-Schauß M, Smolka G. Attributive Concept Descriptions with Complements. Artificial Intelligence. 1991;48(1):1–26.

[7] Zese R, Bellodi E, Lamma E, Riguzzi F. A Description Logics Tableau Reasoner in Prolog. In: Cantone D, Nicolosi Asmundo M, editors. CILC. vol. 1068 of CEUR Workshop Proceedings. CEUR-WS.org; 2013. p. 33–47.

[8] Calì A, Gottlob G, Kifer M. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. Journal of Artificial Intelligence Research. 2013;48:115–174.

[9] Calì A, Gottlob G, Lukasiewicz T. A general datalog-based framework for tractable query answering over ontologies. In: Symposium on Principles of Database Systems. ACM; 2009. p. 77–86.

[10] Calì A, Gottlob G, Lukasiewicz T. A general Datalog-based framework for tractable query answering over ontologies. Journal of Web Semantics. 2012;14:57–83. Available from: `http://dx.doi.org/10.1016/j.websem.2012.03.001`. doi:10.1016/j.websem.2012.03.001.

[11] Calì A, Gottlob G, Lukasiewicz T, Marnette B, Pieris A. Datalog$^{\pm}$: A Family of Logical Knowledge Representation and Query Languages for New Applications. In: IEEE Symposium on Logic in Computer Science. IEEE Computer Society; 2010. p. 228–242.

[12] Zhou Y, Grau BC, Nenov Y, Kaminski M, Horrocks I. PAGOdA: Pay-As-You-Go Ontology Query Answering Using a Datalog Reasoner. Journal of Artificial Intelligence Research. 2015;54:309–367.

[13] Kakas AC, Kowalski RA, Toni F. Abductive Logic Programming. Journal of Logic and Computation. 1993;2(6):719–770.

[14] Fung TH, Kowalski RA. The IFF proof procedure for abductive logic programming. Journal of Logic Programming. 1997 Nov;33(2):151–165.

[15] Alberti M, Chesani F, Gavanelli M, Lamma E, Mello P, Torroni P. Verifiable Agent Interaction in Abductive Logic Programming: the SCIFF framework. ACM Transactions on Computational Logic. 2008;9(4).

[16] Jaffar J, Maher MJ. Constraint Logic Programming: a Survey. Journal of Logic Programming. 1994;19-20:503–582.

[17] Alberti M, Gavanelli M, Lamma E, Mello P, Sartor G, Torroni P. Mapping Deontic Operators to Abductive Expectations. Computational and Mathematical Organization Theory. 2006 Oct;12(2–3):205 – 225. doi:10.1007/s10588-006-9544-8.

[18] Alberti M, Gavanelli M, Lamma E, Mello P, Torroni P. Specification and Verification of Agent Interactions using Social Integrity Constraints. Electronic Notes in Theoretical Computer Science. 2003;85(2).

[19] Alberti M, Chesani F, Gavanelli M, Lamma E, Mello P, Montali M. An Abductive Framework for A-Priori Verification of Web Services. In: Maher M, editor. Proceedings of the Eighth Symposium on Principles and Practice of Declarative Programming. New York, USA: ACM Press; 2006. p. 39–50.

[20] Chesani F, Mello P, Montali M, Storari S, Torroni P. On the integration of declarative choreographies and Commitment-based agent societies into the SCIFF logic programming framework. Multiagent and Grid Systems. 2010;6(2):165–190. Available from: `http://dx.doi.org/10.3233/MGS-2010-0147`. doi:10.3233/MGS-2010-0147.

[21] Calì A, Gottlob G, Kifer M. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. In: International Conference on Principles of Knowledge Representation and Reasoning. AAAI Press; 2008. p. 70–80.

[22] Gottlob G, Lukasiewicz T, Simari GI. Conjunctive Query Answering in Probabilistic Datalog+/- Ontologies. In: International Conference on Web Reasoning and Rule Systems. vol. 6902 of LNCS. Springer; 2011. p. 77–92.

[23] Lloyd JW. Foundations of Logic Programming. 2nd ed. Springer-Verlag; 1987.

[24] Kunen K. Negation in logic programming. In: Journal of Logic Programming. vol. 4; 1987. p. 289–308.

[25] Alberti M, Gavanelli M, Lamma E. The CHR-based Implementation of the SCIFF Abductive System. Fundamenta Informaticae. 2013;124(4):365–381. doi:10.3233/FI-2013-839.

[26] Alberti M, Chesani F, Gavanelli M, Lamma E, Mello P, Torroni P. Security protocols verification in Abductive Logic Programming: a case study. In: Dikenelli O, Gleizes MP, Ricci A, editors. ESAW 2005 Post-proceedings. No. 3963 in LNAI. Kusadasi, Aydin, Turkey: Springer-Verlag; 2006. p. 106–124.

[27] Reiter R. A Theory of Diagnosis from First Principles. Artif Intell. 1987;32(1):57–95.

[28] Beckert B, Posegga J. leanTAP: Lean Tableau-based Deduction. J Autom Reasoning. 1995;15(3):339–358.

[29] Posegga J, Schmitt P. Implementing Semantic Tableaux. In: D'Agostino M, Gabbay D, Hähnle R, Posegga J, editors. Handbook of Tableau Methods. Springer Netherlands; 1999. p. 581–629. Available from: `http://dx.doi.org/10.1007/978-94-017-1754-0_10`. doi:10.1007/978-94-017-1754-0_10.

[30] Hustadt U, Motik B, Sattler U. Deciding expressive description logics in the framework of resolution. Inf Comput. 2008;206(5):579–601.

[31] Lukácsy G, Szeredi P. Efficient description logic reasoning in Prolog: The DLog system. TPLP. 2009;9(3):343–414.

[32] Straccia U, Lopes N, Lukacsy G, Polleres A. A General Framework for Representing and Reasoning with Annotated Semantic Web Data. In: Fox M, Poole D, editors. Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010. AAAI Press; 2010. p. 1437–1442. Available from: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1590.

[33] Meissner A. An automated deduction system for description logic with ALCN language. Studia z Automatyki i Informatyki. 2004;28-29:91–110.

[34] Meissner A. A simple distributed reasoning system for the connection calculus. Vietnam Journal of Computer Science. 2014;1(4):231–239. Available from: http://dx.doi.org/10.1007/s40595-014-0023-8. doi:10.1007/s40595-014-0023-8.

[35] Herchenröder T. Lightweight Semantic Web Oriented Reasoning in Prolog: Tableaux Inference for Description Logics; 2006.

[36] Faizi I. A Description Logic Prover in Prolog; 2011. Bachelor's thesis, Informatics Mathematical Modelling, Technical University of Denmark.

[37] Ricca F, Gallucci L, Schindlauer R, Dell'Armi T, Grasso G, Leone N. OntoDLV: An ASP-based System for Enterprise Ontologies. J Log Comput. 2009;19(4):643–670.

[38] Leone N, Manna M, Terracina G, Veltri P. Efficiently Computable Datalog$\exists$ Programs. In: Brewka G, Eiter T, McIlraith SA, editors. Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012. AAAI Press; 2012. .

[39] Nenov Y, Piro R, Motik B, Horrocks I, Wu Z, Banerjee J. RDFox: A Highly-Scalable RDF Store. In: Arenas M, Corcho Ó, Simperl E, Strohmaier M, d'Aquin M, Srinivas K, et al., editors. The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II. vol. 9367 of Lecture Notes in Computer Science. Springer; 2015. p. 3–20.

[40] Zese R, Bellodi E, Lamma E, Riguzzi F, Aguiari F. Semantics and Inference for Probabilistic Description Logics. In: Bobillo F, Carvalho RN, da Costa PCG, d'Amato C, Fanizzi N, Laskey KB, et al., editors. Uncertainty Reasoning for the Semantic Web III - ISWC International Workshops, URSW 2011-2013, Revised Selected Papers. vol. 8816 of Lecture Notes in Computer Science. Springer; 2014. p. 79–99. doi:10.1007/978-3-319-13413-0_5.

[41] Motik B, Rosati R. Reconciling Description Logics and Rules. Journal of the ACM. 2010 Jun;57(5):30:1–30:62. doi:10.1145/1754399.1754403.

[42] Alberti M, Lamma E, Riguzzi F, Zese R. Probabilistic Hybrid Knowledge Bases under the Distribution Semantics. In: Adorni G, Cagnoni S, Gori M, Maratea M, editors. Proceedings of the 15th Conference of the Italian Association for Artificial Intelligence (AI*IA2016), Genova, Italy, 28 November - 1 December 2016. vol. 10037 of Lecture Notes in Computer Science. Heidelberg, Germany: Springer International Publishing; 2016. p. 364–376. The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-49130-1_27.

[43] Alberti M, Lamma E, Riguzzi F, Zese R. A Distribution Semantics for non-DL-Safe Probabilistic Hybrid Knowledge Bases. In: Theil Have C, Zese R, editors. 4th International Workshop on Probabilistic logic programming, PLP 2017. vol. 1916 of CEUR Workshop Proceedings. Aachen, Germany: Sun SITE Central Europe; 2017. p. 40–50.

[44] Eiter T, Ianni G, Lukasiewicz T, Schindlauer R, Tompits H. Combining answer set programming with description logics for the Semantic Web. Artif Intell. 2008;172(12-13):1495–1539. Available from: https://doi.org/10.1016/j.artint.2008.04.002.

[45] Alberti M, Cattafi M, Chesani F, Gavanelli M, Lamma E, Mello P, et al. A Computational Logic Application Framework for Service Discovery and Contracting. International Journal of Web Services Research. 2011;8(3):1–25.

[46] Zese R, Bellodi E, Lamma E, Riguzzi F. Logic Programming Techniques for Reasoning with Probabilistic Ontologies. In: Papini O, Benferhat S, Garcia L, Mugnier M, Fermé EL, Meyer T, et al., editors. Proceedings of the Joint Ontology Workshops 2015 Episode 1: The Argentine Winter of Ontology co-located with the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), Buenos Aires, Argentina, July 25-27, 2015.. vol. 1517 of CEUR Workshop Proceedings. CEUR-WS.org; 2015. Available from: http://ceur-ws.org/Vol-1517/JOWO-15_ontolp_paper_3.pdf.

[47] Wielemaker J, Schrijvers T, Triska M, Lager T. SWI-Prolog. Theory and Practice of Logic Programming. 2011;http://arxiv.org/abs/1011.5332.

[48] Carlsson M, Mildner P. SICStus Prolog - The first 25 years. Theory and Practice of Logic Programming. 2012;12(1-2):35–66. Available from: http://dx.doi.org/10.1017/S1471068411000482. doi:10.1017/S1471068411000482.

[49] Baader F, Calvanese D, McGuinness DL, Nardi D, Patel-Schneider PF, editors. The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press; 2003.

[50] Auer S, Bizer C, Kobilarov G, Lehmann J, Cyganiak R, Ives ZG. DBpedia: A Nucleus for a Web of Open Data. In: Aberer K, Choi K, Noy NF, Allemang D, Lee K, Nixon LJB, et al., editors. The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007.. vol. 4825 of Lecture Notes in Computer Science. Springer; 2007. p. 722–735. Available from: http://dx.doi.org/10.1007/978-3-540-76298-0_52. doi:10.1007/978-3-540-76298-0_52.

[51] Demir E, et al. The BioPAX community standard for pathway data sharing. Nature Biotech. 2010 Sep;28(9):935–942. Available from: http://dx.doi.org/10.1038/nbt.1666. doi:10.1038/nbt.1666.

[52] Nagypál G, Deswarte R, Oosthoek J. Applying the Semantic Web: The VICODI Experience in Creating Visual Contextualization for History. Literary and Linguistic Computing. 2005;20(3):327–349. Available from: http://llc.oxfordjournals.org/content/20/3/327.abstract. doi:10.1093/llc/fqi037.

[53] Frühwirth T. Theory and Practice of Constraint Handling Rules. Journal of Logic Programming. 1998 Oct;37(1-3):95–138.

[54] Gottlob G, Lukasiewicz T, Pieris A. Datalog+/-: Questions and answers. In: Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning; 2014. .

[55] Alberti M, Bellodi E, Cota G, Lamma E, Riguzzi F, Zese R. Probabilistic Constraint Logic Theories. In: Hommersom A, Abdallah SA, editors. Proceedings of the 3rd International Workshop on Probabilistic Logic Programming co-located with 26th International Conference on Inductive Logic Programming (ILP 2016), London, UK, September 3, 2016.. vol. 1661 of CEUR Workshop Proceedings. CEUR-WS.org; 2016. p. 15–28.

[56] Nocedal J. Updating quasi-Newton matrices with limited storage. Mathematics of Computation. 1980;35(151):773–782.