An abductive Framework for Datalog$^{\pm}$ Ontologies

MARCO GAVANELLI, EVELINA LAMMA, FABRIZIO RIGUZZI,

ELENA BELLODI, RICCARDO ZESE and GIUSEPPE COTA
University of Ferrara, Italy

submitted 29 May 2015; accepted 14 July 2015

Abstract

Ontologies are a fundamental component of the Semantic Web since they provide a formal and machine manipulable model of a domain. Description Logics (DLs) are often the languages of choice for modeling ontologies. Great effort has been spent in identifying decidable or even tractable fragments of DLs. Conversely, for knowledge representation and reasoning, integration with rules and rule-based reasoning is crucial in the so-called Semantic Web stack vision. Datalog$^{\pm}$ is an extension of Datalog which can be used for representing lightweight ontologies, and is able to express the DL-Lite family of ontology languages, with tractable query answering under certain language restrictions.

In this work, we show that Abductive Logic Programming (ALP) is also a suitable framework for representing Datalog$^{\pm}$ ontologies, supporting query answering through an abductive proof procedure, and smoothly achieving the integration of ontologies and rule-based reasoning. In particular, we consider an Abductive Logic Programming framework named $\mathcal{S}$CIFF and derived from the IFF abductive framework, able to deal with existentially (and universally) quantified variables in rule heads, and Constraint Logic Programming constraints. Forward and backward reasoning is naturally supported in the ALP framework. We show that 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.

KEYWORDS: Abductive Logic Programming, Datalog$^{\pm}$, Description Logics, Semantic Web.

1 Introduction

The main idea of the Semantic Web is making information available in a form that is understandable and automatically manageable by machines (Hitzler et al. 2009). 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 concepts should be classified), possibly together with constraints capturing additional knowledge about the domain. Ontologies provide a formal and machine manipulable model of a domain, and this justifies their use in the Semantic Web.

In order to realize this vision, the W3C has supported the development of a family of knowledge representation formalisms of increasing complexity for defining ontologies, called Web Ontology Language (OWL). In particular, OWL 1 defines the sublanguages OWL-Lite, OWL-DL (based on Description Logics) and OWL-Full. Therefore, ontologies are a fundamental component of the Semantic Web and Description Logics (DLs) are often the languages of choice for modeling them.

Extensive work has focused on developing tractable DLs, identifying the *DL-Lite* family (Calvanese et al. 2007), for which answering conjunctive queries is in AC_0 in data complexity.

In a related research direction, Calì et al. (2009b) proposed Datalog$^\pm$, an extension of Datalog with existential rules for defining ontologies. Datalog$^\pm$ can be used for representing lightweight ontologies, and encompasses the DL-Lite family (Calì et al. 2009a). By suitably restricting the language syntax and adopting appropriate syntactic conditions, also Datalog$^\pm$ achieves tractability (Calì et al. 2008).

In this work, we consider the Datalog$^\pm$ language and show how ontologies expressed in this language can be also modeled in an Abductive Logic Programming (ALP, for short) framework (Kakas et al. 1993), where query answering is supported by the underlying ALP proof procedure. ALP has been proved a powerful tool for knowledge representation and reasoning, taking advantage from ALP operational support as a (static or dynamic) verification tool. ALP languages are usually equipped with a declarative (model-theoretic) semantics, and an operational semantics given in terms of a proof-procedure. Several abductive proof procedures have been defined (both backward, forward, and a mix of the two such) (Kakas and Mancarella 1990; Bry 1990; Kakas 2000; Denecker and Schreye 1998; Alferes et al. 1999; Abdennadher and Christiansen 2000; Christiansen and Dahl 2005; Endriss et al. 2004), with many different applications (diagnosis, monitoring, verification, etc.). Among them, a notable one is the IFF abductive proof-procedure (Fung and Kowalski 1997) which was proposed to deal with forward rules, and with non-ground abducibles. This proof procedure has been later extended in (Alberti et al. 2008), and the resulting proof procedure, named $\mathcal{S}$CIFF, can deal with both existentially and universally quantified variables in rule heads, and Constraint Logic Programming (CLP) constraints (Jaffar and Maher 1994). The resulting system has been used for modeling and implementing several knowledge representation frameworks, such as deontic logic (Alberti et al. 2006), normative systems (Alberti et al. 2012), interaction protocols for multi-agent systems (Alberti et al. 2004), Web services choreographies (Alberti et al. 2006), etc. also providing an effective reasoning system.

Here we concentrate on Datalog$^\pm$ ontologies, and show how an ALP language enriched with quantified variables (existential to our purposes) can be a useful knowledge representation and reasoning framework for them. We do not focus here on complexity results of the overall system, which is, however, not tractable.

Forward and backward reasoning is naturally supported by the ALP proof procedure, and the considered $\mathcal{S}$CIFF language smoothly supports the integration of rules, expressed in a Logic Programming language, with ontologies expressed in Datalog$^\pm$. In fact, In fact, $\mathcal{S}$CIFF allows us to map Datalog$^\pm$ ontologies into the forward integrity constraints on which it is based. Other ALP languages could be used (Kakas et al. 2001; Mancarella et al. 2009); we chose $\mathcal{S}$CIFF because it is freely available on the web and it is supported on the last versions of commercial and open source Prolog systems (SICStus (Carlsson and Mildner 2012) and SWI (Wielemaker et al. 2012)).

In the following, Section 2 introduces Datalog$^{\pm}$. Section 3 introduces Abductive Logic Programming, and the $\mathcal{S}$CIFF language. Section 4 shows how the considered Datalog$^{\pm}$ language can be mapped into $\mathcal{S}$CIFF. Section 5 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 the DL-Lite family of ontology languages (Calì et al. 2009a). By restricting the language syntax, Datalog$^{\pm}$ achieves decidability (Calì et al. 2008).

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 $q(D) \neq \emptyset$.

Datalog$^{\pm}$ syntax includes three types of implication rules. 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 on $\mathcal{R}$, and a database D for $\mathcal{R}$, the set of models of D given T, denoted $\mathit{mods}(D, T)$, is the set of all (possibly infinite) databases B such that $D \subseteq B$ and every $F \in T$ is satisfied in B. The set of answers to a CQ q on D given T, denoted $\mathit{ans}(q, D, T)$, is the set of all tuples $\mathbf{t}$ such that $\mathbf{t} \in q(B)$ for all $B \in \mathit{mods}(D, T)$. The answer to a BCQ q over D given T is Yes, denoted $D \cup T \models q$, iff $B \models q$ for all $B \in \mathit{mods}(D, T)$.

The second component of a Datalog$^{\pm}$ theory is represented by *negative constraints* (NC): first-order formulas of the form $\forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow \mathit{false}$, where $\Phi(\mathbf{X})$ is a conjunction of atoms. The universal quantifiers are usually left implicit.

Equality-generating dependencies (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.

2.1 The Chase

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 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. 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 a non-constant element $e \in \{h(X_i), h(X_j)\}$ in D by the other element e' (if e and e' are both nulls, then e precedes e' 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 (Calì et al. 2010). 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).

The two problems of CQ and BCQ evaluation under TGDs and EGDs are LOGSPACE-equivalent (Calì et al. 2009b). Moreover, query answering under TGDs is equivalent to query answering under TGDs with only single atoms in their heads (Calì et al. 2008). Henceforth, we focus only on the BCQ evaluation problem and we assume that every TGD has a single atom in its head (without loss of generality since by introducing new predicate symbols, we can always transform TGD-rules with multiple atoms in the head in sets of rules with only one). 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 1 (*Adapted from (Gottlob et al. 2011)*)
Consider the following ontology for a real estate information extraction system:
$F_1 = ann(X, label), ann(X, price), visible(X) \rightarrow priceElem(X)$
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 s.t. 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 answers $q(L) = \{summertown\}$. 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 $summertown = z_1$. If F_7 were absent then $q(L) = \{summertown, z_1\}$.

Answering BCQs q over databases and ontologies containing NCs can be performed by first checking whether the BCQ $\Phi(\mathbf{X})$ evaluates to false for each NC of the form $\forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow false$. If one of these checks fails, then the answer to the original BCQ q

is positive, otherwise the negative constraints can be simply ignored when answering the original BCQ q.

3 ALP and the $\mathcal{S}$CIFF language

Abductive Logic Programming (ALP, for short) is a family of programming languages that integrate abductive reasoning into logic programming. An ALP program consists of a set of clauses, that can contain in the body some distinguished predicates, belonging to a set $\mathcal{A}$ and called *abducibles*. The aim is finding a set of abducibles **EXP**, built from symbols in $\mathcal{A}$ that, together with the knowledge base, is an explanation for a given known effect (called *goal* $\mathcal{G}$) and satisfies a set of logic formulae, called *Integrity Constraints* (IC):

$$KB \cup \mathbf{EXP} \models \mathcal{G}$$
$$KB \cup \mathbf{EXP} \models IC$$

$\mathcal{S}$CIFF (Alberti et al. 2008) is a language in the ALP class, originally designed to model and verify interactions in open societies of agents and it is an extension of the IFF proof-procedure (Fung and Kowalski 1997). As the IFF, it relies on the three-valued completion semantics (Kunen 1987), it considers integrity constraints of the form *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 CLP constraints (Jaffar and Maher 1994), occurring events (only in the body), or positive and negative expectations, as will be explained soon.

Definition 1
A *$\mathcal{S}$CIFF Program* is a pair $\langle KB, \mathcal{IC}\rangle$ where KB is a set of clauses (an extended logic program) and $\mathcal{IC}$ is a set of forward rules called *Integrity Constraints* (ICs, for short in the following).

$\mathcal{S}$CIFF considers a (possibly dynamically growing) set of facts (called *history*) **HAP**, that contains ground atoms $\mathbf{H}(Event[, Time])$. This set can grow dynamically, during the computation, thus implementing a dynamic acquisition of events. Some distinguished abducibles are called *expectations*. A *positive expectation*, written $\mathbf{E}(Event[, Time])$ means that a corresponding event $\mathbf{H}(Event[, Time])$ is expected to happen, while $\mathbf{EN}(Event[, Time])$ is a *negative expectation*, and requires events $\mathbf{H}(Event[, Time])$ not to happen. To simplify the notation, we will omit the *Time* argument from events and expectations.

Variables occurring only in positive expectations are existentially quantified (expressing the idea that a single event is enough to support them), while those in negative expectations are universally quantified, so that any event matching with a negative expectation leads to inconsistency with the current hypothesis. CLP (Jaffar and Maher 1994) 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 **EXP** of abduced literals.

The declarative semantics of $\mathcal{S}$CIFF includes the classic conditions of ALP:

$$KB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathcal{G} \tag{1}$$
$$KB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathcal{IC} \tag{2}$$

plus specific conditions to support the confirmation of expectations.

Positive/negative expectations are confirmed (not violated) if

$$KB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathbf{E}(X) \rightarrow \mathbf{H}(X) \tag{3}$$
$$KB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathbf{EN}(X) \wedge \mathbf{H}(X) \rightarrow \mathit{false} \tag{4}$$

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

$$KB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathbf{E}(X) \wedge \mathbf{EN}(X) \rightarrow \mathit{false} \tag{5}$$

Definition 2 (*$\mathcal{S}$CIFF answer*)
Given a $\mathcal{S}$CIFF program $\langle KB, \mathcal{IC}\rangle$ and a history **HAP**, a goal $\mathcal{G}$ is a $\mathcal{S}$CIFF answer if there is a set **EXP** such that equations (1)-(5) 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 proof tree, whose nodes represent states of the computation. A set of transitions rewrite a node into one or more children nodes.

The main transitions, inherited from the IFF are:

Unfolding replaces a (non abducible) atom with its definitions;
Propagation if an abduced atom $\mathbf{a}(X)$ occurs in the condition of an IC (e.g., $\mathbf{a}(Y) \rightarrow p$), the atom is removed from the condition (generating $X = Y \rightarrow p$);
Case Analysis given an implication containing an equality in the condition (e.g., $X = Y \rightarrow p$), generates two children in logical or (in the example, either $X = Y$ and p, or $X \neq Y$);
Equality rewriting rewrites equalities as in the Clark's equality theory;
Logical simplifications other simplifications like (true $\rightarrow A$) $\Leftrightarrow A$, etc.

$\mathcal{S}$CIFF includes also the transitions of CLP (Jaffar and Maher 1994) for constraint solving.

A complete description is in (Alberti et al. 2008), with proofs of soundness, completeness, and termination. $\mathcal{S}$CIFF was implemented in CHR (Frühwirth 1998), an efficient implementation is described in (Alberti et al. 2013).

In this paper we consider the *generative version* of $\mathcal{S}$CIFF, called g-$\mathcal{S}$CIFF (Alberti et al. 2006), in which also the **H** events are considered as abducibles, and can be assumed like the other abducible predicates, beside being provided as input in the history **HAP**; they are then collected in a set $\mathbf{HAP}' \supseteq \mathbf{HAP}$.

Definition 3 (*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 (1)-(5) are satisfied[1]. In this case, we write

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

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 a history. $\mathcal{S}$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 in their heads. We show that this subset of the language suffices to represent Datalog$^{\pm}$ ontologies[2].

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.

Definition 4
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 Datalog$^{\pm}$ formulae:

$$\begin{array}{rcl}
\tau(\mathit{Body} \rightarrow \mathit{Head}) & = & \tau_{\mathbf{H}}(\mathit{Body}) \rightarrow \tau_{\mathbf{E}}(\mathit{Head}) \\
\tau_{\mathbf{H}}(A) & = & \mathbf{H}(A) \\
\tau_{\mathbf{E}}(A) & = & \mathbf{E}(A) \\
\tau_{\mathbf{M}}(F_1 \land F_2) & = & \tau_{\mathbf{M}}(F_1) \land \tau_{\mathbf{M}}(F_2) \\
\tau_{\mathbf{M}}(\mathit{false}) & = & \mathit{false} \\
\tau_{\mathbf{M}}(Y_i = Y_j) & = & Y_i = Y_j \\
\tau_{\mathbf{E}}(\exists \mathbf{X}\ A) & = & \mathbf{E}(A)
\end{array}$$

A Datalog$^{\pm}$ database D for $\mathcal{R}$ corresponds to the (possibly infinite) $\mathcal{S}$CIFF history $\mathbf{HAP}$, 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, but they are left implicit in the $\mathcal{S}$CIFF syntax, while the rest of the variables are universally quantified with scope the entire IC.

[1] In the equations (1)-(5) the set $\mathbf{HAP}$ should be substituted with $\mathbf{HAP}'$.
[2] As should be clear soon, the only CLP constraint used in the mapping is the equality constraint

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

Recall that for a set of TGDs T on $\mathcal{R}$, and a database D for $\mathcal{R}$, the set of models of D given T, denoted $mods(D, T)$, is the set of all (possibly infinite) databases B such that $D \subseteq B$ and every $F \in T$ is satisfied in B. For any such database B, we can prove that there exists an abductive explanation $\mathbf{EXP} = \tau_{\mathbf{E}}(B)$, $\mathbf{HAP'} = \tau_{\mathbf{H}}(B)$ such that:

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

where $\mathbf{HAP'} \supseteq \mathbf{HAP} = \tau_{\mathbf{H}}(D)$, and $\mathcal{IC} = \tau(T)$.

Finally, Datalog$^{\pm}$ negative constraints NC 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.

Therefore, informally speaking, the set of models of D given T, $mods(D, T)$, corresponds to the set of all the abductive explanations $\mathbf{EXP}$ satisfying the set of $\mathcal{S}$CIFF integrity constraints $\mathcal{IC} = \tau(\mathcal{T})$.

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: $G = \tau_{\mathbf{E}}(\Phi(\mathbf{Y}))$.

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

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

Theorem 1 (*Completeness of query answering*)
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 $\langle \emptyset, \mathcal{A}, \tau(T) \rangle$ 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 $G\theta = \tau_{\mathbf{E}}(\Phi(\mathbf{t}, \_))$.

Corollary 1 (*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:

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

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

The $\mathcal{S}$CIFF proof procedure was proved sound and complete w.r.t. $\mathcal{S}$CIFF declarative semantics in (Alberti et al. 2008), thus for each abductive explanation $\mathbf{EXP}$ for a given goal G in a $\mathcal{S}$CIFF program, there exists a $\mathcal{S}$CIFF-based computation producing a set of abducibles (positive expectations to our purposes) $\delta \subseteq \mathbf{EXP}$, and a computed answer substitution for goal G possibly more general than θ.

Example 2 (*Real estate information extraction system in ALP*)
The TGDs F_1-F_8 from the Datalog$^\pm$ ontology of Example 1 are one-to-one mapped into the following $\mathcal{S}$CIFF ICs:[3]
$IC_1 : \mathbf{H}(ann(X, label)), \mathbf{H}(ann(X, price)), \mathbf{H}(visible(X)) \rightarrow \mathbf{E}(priceElem(X))$
$IC_2 : \mathbf{H}(ann(X, label)), \mathbf{H}(ann(X, priceRange)), \mathbf{H}(visible(X)) \rightarrow \mathbf{E}(priceElem(X))$
$IC_3 : \mathbf{H}(priceElem(E)), \mathbf{H}(group(E, X)) \rightarrow \mathbf{E}(forSale(X))$
$IC_4 : \mathbf{H}(forSale(X)) \rightarrow (\exists P)\ \mathbf{E}(price(X, P))$
$IC_5 : \mathbf{H}(hasCode(X, C)), \mathbf{H}(codeLoc(C, L)) \rightarrow \mathbf{E}(loc(X, L))$
$IC_6 : \mathbf{H}(hasCode(X, C)) \rightarrow (\exists L)\ \mathbf{E}(codeLoc(C, L)), \mathbf{E}(loc(X, L))$
$IC_7 : \mathbf{H}(loc(X, L1)), \mathbf{H}(loc(X, L2)) \rightarrow L1 = L2$
$IC_8 : \mathbf{H}(loc(X, L)) \rightarrow \mathbf{E}(advertised(X))$
The database is then simply mapped into the following history $\mathbf{HAP}$:

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

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.

Each of the (ground) atomic queries of Example 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 (with unification $L = summertown$) since:

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

Also in this case, if we remove IC_7 we obtain the previous answer, and a further one, similar to the one obtained by Datalog$^\pm$, where the set $\mathbf{HAP}'$ contains two *loc* events:

$$\mathbf{H}(loc(prop1, summertown)), \quad \exists L\ \mathbf{H}(loc(prop1, L))$$

and the answer includes a further CLP constraint $L \neq summertown$ (being, instead, L and *summertown* unified in the previous answer).

[3] We show for clarity the quantification of existentially quantified variables, although in the $\mathcal{S}$CIFF syntax the quantification is implicit.

5 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 and inspired by the IFF proof procedure, was implemented in Constraint Handling Rules (Frühwirth 1998). The $\mathcal{S}$CIFF system has already been used for modeling and implementing several knowledge representation frameworks, also providing an effective reasoning system.

Here we have shown how the $\mathcal{S}$CIFF language can be a useful knowledge representation and reasoning framework for Datalog$^\pm$ ontologies. In fact, the underlying abductive proof procedure can be directly exploited as an ontological reasoner for query answering and consistency check, also supporting inline incrementality of the extensional part of the knowledge base (namely, the $\mathcal{A}$Box), represented in $\mathcal{S}$CIFF as a (possibly incremental) set of events. To the best of our knowledge, this is the first application of ALP to model and reason upon ontologies.

Many issues have not been addressed in this paper, and they will be subject of future work. First of all, we have not focused here on complexity results. Future work will be devoted to identify syntactic conditions guaranteeing tractable ontologies in $\mathcal{S}$CIFF, in the style of what has been done for Datalog$^\pm$.

A second issue for future work concerns experimentation and comparison with other approaches, even not Logic Programming based, on real-size ontologies.

Finally, $\mathcal{S}$CIFF language is richer than the subset here used to represent Datalog$^\pm$ ontologies. In fact, the $\mathcal{S}$CIFF integrity constraints can have the disjunction in the head, and 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 to them. Moreover, the $\mathcal{S}$CIFF language smoothly supports the integration of rules, expressed in a Logic Programming language, with ontologies expressed in Datalog$^\pm$, since a Logic Programming program can be added to the set of ICs, giving the opportunity to consider deductive rules besides the forward ICs themselves. $\mathcal{S}$CIFF also allows for CLP constraints beside the equality one, which can be used also in the ICs as well. Finally, this rich language could be used to add further expressivity to query languages.

References

Abdennadher, S. and Christiansen, H. 2000. An experimental CLP platform for integrity constraints and abduction. In *FQAS, Flexible Query Answering Systems*, H. Larsen, J. Kacprzyk, S. Zadrozny, T. Andreasen, and H. Christiansen, Eds. LNCS. Springer-Verlag, Warsaw, Poland, 141–152.

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., and Montali, M. 2006. An abductive framework for a-priori verification of web services. In *Proceedings of the Eighth Symposium on Principles and Practice of Declarative Programming*, M. Maher, Ed. ACM Press, New York, USA, 39–50.

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., and Torroni, P. 2006. Security protocols verification in abductive logic programming: a case study. In

ESAW 2005 Post-proceedings, O. Dikenelli, M.-P. Gleizes, and A. Ricci, Eds. Number 3963 in LNAI. Springer-Verlag, Kusadasi, Aydin, Turkey, 106–124.

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., and Torroni, P. 2008. Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM Transactions on Computational Logic 9,* 4, 29:1–29:43.

Alberti, M., Gavanelli, M., and Lamma, E. 2012. Deon + : Abduction and constraints for normative reasoning. In *Logic Programs, Norms and Action - Essays in Honor of Marek J. Sergot on the Occasion of His 60th Birthday*, A. Artikis, R. Craven, N. K. Cicekli, B. Sadighi, and K. Stathis, Eds. Lecture Notes in Computer Science, vol. 7360. Springer, 308–328.

Alberti, M., Gavanelli, M., and Lamma, E. 2013. The CHR-based implementation of the SCIFF abductive system. *Fundamenta Informaticae 124,* 4, 365–381.

Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Sartor, G., and Torroni, P. 2006. Mapping deontic operators to abductive expectations. *Computational and Mathematical Organization Theory 12,* 2–3 (Oct.), 205 – 225.

Alberti, M., Gavanelli, M., Lamma, E., Mello, P., and Torroni, P. 2004. Specification and verification of agent interactions using social integrity constraints. *Electronic Notes in Theoretical Computer Science 85,* 2 (Apr.), 94–116.

Alferes, J. J., Pereira, L. M., and Swift, T. 1999. Well-founded abduction via tabled dual programs. In *Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA*, D. D. Schreye, Ed. MIT Press, Cambridge, MA, 426–440.

Bry, F. 1990. Intensional updates: Abduction via deduction. In *Logic Programming, Proceedings of the Seventh International Conference, Jerusalem, Israel*, D. Warren and P. Szeredi, Eds. MIT Press, Cambridge, MA, 561–578.

Calì, A., Gottlob, G., and Kifer, M. 2008. Taming the infinite chase: Query answering under expressive relational constraints. In *International Conference on Principles of Knowledge Representation and Reasoning.* AAAI Press, 70–80.

Calì, A., Gottlob, G., and Lukasiewicz, T. 2009a. A general datalog-based framework for tractable query answering over ontologies. In *Symposium on Principles of Database Systems.* ACM, 77–86.

Calì, A., Gottlob, G., and Lukasiewicz, T. 2009b. Tractable query answering over ontologies with Datalog$^{\pm}$. In *International Workshop on Description Logics.* CEUR Workshop Proceedings, vol. 477. CEUR-WS.org.

Calì, A., Gottlob, G., Lukasiewicz, T., Marnette, B., and Pieris, A. 2010. Datalog$^{\pm}$: A family of logical knowledge representation and query languages for new applications. In *IEEE Symposium on Logic in Computer Science.* 228–242.

Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., and Rosati, R. 2007. Tractable reasoning and efficient query answering in description logics: The *dl-lite* family. *J. Autom. Reasoning 39,* 3, 385–429.

Carlsson, M. and Mildner, P. 2012. SICStus Prolog – the first 25 years. *Theory and Practice of Logic Programming 12*, 35–66.

Christiansen, H. and Dahl, V. 2005. HYPROLOG: A new logic programming language with assumptions and abduction. In *Proc. ICLP 2005*, M. Gabbrielli and G. Gupta, Eds. LNCS, vol. 3668. Springer, 159–173.

Denecker, M. and Schreye, D. D. 1998. SLDNFA: an abductive procedure for abductive logic programs. *Journal of Logic Programming 34,* 2, 111–167.

Endriss, U., Mancarella, P., Sadri, F., Terreni, G., and Toni, F. 2004. The CIFF proof procedure for abductive logic programming with constraints. In *Proc. JELIA 2004*, J. J. Alferes and J. A. Leite, Eds. LNAI, vol. 3229. Springer-Verlag, 31–43.

Frühwirth, T. 1998. Theory and practice of constraint handling rules. *Journal of Logic Programming 37,* 1-3 (Oct.), 95–138.

Fung, T. H. and Kowalski, R. A. 1997. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming 33,* 2 (Nov.), 151–165.

Gottlob, G., Lukasiewicz, T., and Simari, G. I. 2011. Conjunctive query answering in probabilistic Datalog+/- ontologies. In *International Conference on Web Reasoning and Rule Systems*. LNCS, vol. 6902. Springer, 77–92.

Hitzler, P., Krötzsch, M., and Rudolph, S. 2009. *Foundations of Semantic Web Technologies*. CRCPress.

Jaffar, J. and Maher, M. 1994. Constraint logic programming: a survey. *Journal of Logic Programming 19-20*, 503–582.

Kakas, A. C. 2000. ACLP: integrating abduction and constraint solving. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning, NMR'00, Breckenridge, CO*.

Kakas, A. C., Kowalski, R. A., and Toni, F. 1993. Abductive Logic Programming. *Journal of Logic and Computation 2,* 6, 719–770.

Kakas, A. C. and Mancarella, P. 1990. On the relation between Truth Maintenance and Abduction. In *Proceedings of the 1st Pacific Rim International Conference on Artificial Intelligence, PRICAI-90, Nagoya, Japan*, T. Fukumura, Ed. Ohmsha Ltd.

Kakas, A. C., van Nuffelen, B., and Denecker, M. 2001. $\mathcal{A}$-System: Problem solving through abduction. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, Seattle, Washington, USA (IJCAI-01)*, B. Nebel, Ed. Morgan Kaufmann Publishers, Seattle, Washington, USA, 591–596.

Kunen, K. 1987. Negation in logic programming. In *Journal of Logic Programming*. Vol. 4. 289–308.

Mancarella, P., Terreni, G., Sadri, F., Toni, F., and Endriss, U. 2009. The CIFF proof procedure for abductive logic programming with constraints: Theory, implementation and experiments. *TPLP 9,* 6, 691–750.

Wielemaker, J., Schrijvers, T., Triska, M., and Lager, T. 2012. SWI-Prolog. *Theory and Practice of Logic Programming 12.*