Abductive Logic Programming for normative reasoning and ontologies

Marco Gavanelli[1], Evelina Lamma[1], Fabrizio Riguzzi[2], Elena Bellodi[1], Riccardo Zese[1], and Giuseppe Cota[1]

[1] Dipartimento di Ingegneria – University of Ferrara
[2] Dipartimento di Matematica e Informatica – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy [name.surname]@unife.it

Abstract. Abductive Logic Programming (ALP) has been exploited to formalize societies of agents, commitments and norms, taking advantage from ALP operational support as a (static or dynamic) verification tool. In [7], the most common deontic operators (obligation, prohibition, permission) are mapped into the abductive expectations of an ALP framework for agent societies. Building upon such correspondence, in [5], authors introduced $Deon^{+}$, a language where obligation and prohibition deontic operators are enriched with quantification over time, by means of ALP and Constraint Logic Programming (CLP).
In recent work [30, 31], we have shown that the same ALP framework can be suitable to represent Datalog$^{\pm}$ ontologies. Ontologies are a fundamental component of both the Semantic Web and knowledge-based systems, even in the legal setting, since they provide a formal and machine manipulable model of a domain.
In this work, we show that ALP is a suitable framework for representing both norms and ontologies. Normative reasoning and ontological query answering are obtained by applying the same abductive proof procedure, smoothly achieving their integration. In particular, we consider the ALP 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 CLP constraints.
The main advantage is that this integration is achieved within a single language, grounded on abduction in computational logic.

1 Introduction

Norms represent desirable behaviors of members of a human or artificial society.

A normative system is a set of norms, together with mechanisms to reason about, apply, and modify them. Norms can be encoded by exploiting the notions of obligation, permission and prohibition, often modelled as modal operators, in the tradition of Deontic Logic [49].

As for the structure of the formulas representing norms, a widely adopted approach is to encode norms as logical rules in the form of implications, since: (i) implications correspond intuitively to conditional norms, which state that some deontic consequence (such as the obligation for an agent to perform an

action) follows from a state of affairs; (*ii*) rule-based systems also provide an operational support for reasoning, and draw conclusions (regarding, e.g., expected behavior, or norm violations and related sanctions). In the legal domain, the British Nationality Act was formalized using Logic Programming (LP) [47]; later, argument-based extended LP with defeasible priorities [43] and the use of defeasible logic was proposed [33]. Satoh et al.'s PROLEG is a Prolog implementation of the Presupposed Ultimate Fact Theory of the Japanese Civil Code [46]. The contract cancellation under the Japanese law was also formalized in computational logics [22].

Normative systems have been also advocated as a tool to model and reason upon a single agent, as in [13] where a normative system can be seen as a normative agent, equipped with mental attitudes, about which other agents can reason, choosing either to fulfill their obligations, or to face the possible sanctions.

More often, normative systems regulate interaction in multi-agent systems [12]. Among the organizational models, [25, 24] exploit Deontic Logic to specify the society norms and rules. The whole research project ALFEBIITE [10] was focused on the formalization of an open society of agents using Deontic Logic.

The EU IST Project SOCS proposed a Computational Logic approach to multi-agent systems. The SOCS social model represents and verifies both social interaction protocols among members regulated via abductive expectations [2], and member specifications themselves [14] in Abductive Logic Programming (ALP). Both approaches have been later applied to model and reason about norms with deontic flavours [7, 44].

In the EU project IMPACT [11, 27], agent programs may be used to specify what an agent is obliged to do, what an agent may do, and what an agent cannot do on the basis of deontic operators of Permission, Obligation and Prohibition (whose semantics does not rely on a Deontic Logic semantics).

In the meantime, legal ontologies have proved crucial for representing, processing and retrieving legal information. A collective reflection on the theoretical foundations of legal ontology engineering is [45]. The ESTRELLA project [28] aimed at developing a standard based platform allowing public administrations to deploy comprehensive legal knowledge management solutions. To this purpose, the project developed a Legal Knowledge Interchange Format (LKIF), building upon OWL, and modeled European tax related legislation and national tax legislation of two European countries as case studies. Integration of rules and ontologies has been faced in [9], which proposes a normative language that combines expressivity of LP and Description Logic (DL) for hybrid knowledge bases modeling human laws, with examples from the Portuguese Penal Code. Another approach is the mapping of DL into computational logic; for example the $\mathcal{ALCN}$ Description Logics was also mapped into Open Logic Programming [48], an extension of ALP. Notable work has been done also in applying abductive reasoning to DL [38, 26].

In Computational Logic, ALP was proved a powerful tool for knowledge representation and reasoning [35], taking advantage from ALP operational support as (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, with many applications (diagnosis, monitoring, verification, etc.). Fung and Kowalski proposed the IFF abductive proof-procedure [29] to deal with forward rules, and with non-ground abducibles. It has been later extended [4], 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 [34]. The resulting system was used for modeling and implementing several knowledge representation frameworks, such as deontic logic [7], where the deontic notions of obligation and permission are mapped into special $\mathcal{S}$CIFF abducible predicates, normative systems [5], interaction protocols for multi-agent systems [8], Web services choreographies [1], etc.

In this work, we move a step forward, by showing that ALP, and $\mathcal{S}$CIFF in particular, is a suitable framework for representing and integrating both norms and Datalog$^{\pm}$ ontologies. Normative reasoning and ontological query answering are obtained by applying the $\mathcal{S}$CIFF abductive proof procedure. The main advantage is that this integration is achieved within a single language, grounded on abduction in Computational Logic.

We assume a basic familiarity with Logic Programming and Abductive Logic Programming, good introductions are, respectively, [40] and [35].

The paper is organized as follows. We first introduce ALP and the $\mathcal{S}$CIFF framework in Section 2, also mentioning its underlying proof procedure. Then, in Section 3, we introduce (a subset of) the deontic language $Deon^{+}$ and show its mapping into $\mathcal{S}$CIFF. Section 4 introduces Datalog$^{\pm}$ to formalize ontologies, and shows its correspondence to $\mathcal{S}$CIFF integrity constraints. This paves the way to the integration of norms and ontologies, discussed in Section 5, where we show a simple example, with norms and an ontology, and discuss the kind of inference supported by the $\mathcal{S}$CIFF proof procedure. More relevant related work is mentioned throughout the various sections. In Section 6 we conclude the paper, and outline future work.

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

Abductive Logic Programming (ALP) is a family of programming languages that integrate abductive reasoning into LP. An ALP program consists of a logic program, also called *knowledge base* KB, that is a set of clauses $h \leftarrow b$. As usual in LP, a set of clauses in which the h share a same functor symbol define a predicate. Differently from classical LP, in ALP there are also predicates that have no definition, that belong to a set $\mathcal{A}$ and are called *abducibles*. Abducible literals cannot occur in the h of a clause, but they can occur in the b. The aim in ALP is finding a set of abducibles $\Delta^{\mathcal{A}}$, built from symbols in $\mathcal{A}$, that, together with the KB, is an explanation for a given known effect (called *goal* $\mathcal{G}$) and satisfies a set of logic formulae, called *Integrity Constraints (ICs)*:

$$KB \cup \Delta^{\mathcal{A}} \models \mathcal{G} \qquad KB \cup \Delta^{\mathcal{A}} \models IC.$$

While in the early abductive proof-procedures [36] the set of abduced literals is ground, in later proof-procedures [29, 23, 37] it can also contain existentially-quantified atoms.

$\mathcal{S}$CIFF [4] is a language in the ALP class, extension of the IFF proof-procedure [29]. As in the IFF, integrity constraints are in the form *body* $\rightarrow$ *head* where *body* is a conjunction of literals and *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, occurring events (only in the body), or positive and negative expectations, as explained in the following.

Definition 1. *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 logic implications called* Integrity Constraints.

$\mathcal{S}$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* $\mathbf{E}(Event[, Time])$ 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 when not needed.

Variables occurring in positive expectations are existentially quantified (expressing the idea that a single event is enough to support them), while those occurring only in negative expectations are universally quantified, so that any event matching with a negative expectation leads to inconsistency with the current hypothesis. Nested existential quantifications are forbidden by the language syntax. CLP [34] constraints can be imposed on variables. The computed answer includes three elements: a substitution for the variables in the goal (as usual in Prolog), the constraint store (as in CLP), and the set $\Delta^{\mathcal{A}}$ of abduced literals.

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

$$KB \cup \mathbf{HAP} \cup \Delta^{\mathcal{A}} \models \mathcal{G} \tag{1}$$

$$KB \cup \mathbf{HAP} \cup \Delta^{\mathcal{A}} \models \mathcal{IC} \tag{2}$$

plus specific conditions to support the confirmation of expectations.

As the history can be dynamically growing, it makes sense to adopt either an open or closed world assumption on the history, depending on the envisaged application. $\mathcal{S}$CIFF can support both types of reasoning. In a *skeptical* reasoning attitude, all hypotheses that are not explicitly confirmed are rejected, assuming that no more events can happen (closed world assumption on the history): all the positive expectations that are not matched with an actual event are disconfirmed (symmetrically, the pending negative expectations are confirmed). In a *credulous* reasoning attitude, a set of hypotheses is acceptable even if some hypotheses are not explicitly confirmed, as long as the set is consistent. Declaratively, in skeptical reasoning positive expectations are confirmed if

$$KB \cup \mathbf{HAP} \cup \Delta^{\mathcal{A}} \models \mathbf{E}(X) \rightarrow \mathbf{H}(X), \tag{3}$$

which is not required in a credulous attitude. In this paper, we adopt a credulous reasoning attitude. In both cases, negative expectations should not be disconfirmed by actual events, and the same event cannot be expected both to happen and not to happen:

$$KB \cup \mathbf{HAP} \cup \Delta^{\mathcal{A}} \models \mathbf{EN}(X) \wedge \mathbf{H}(X) \rightarrow \mathit{false}. \quad (4)$$

$$KB \cup \mathbf{HAP} \cup \Delta^{\mathcal{A}} \models \mathbf{E}(X) \wedge \mathbf{EN}(X) \rightarrow \mathit{false}. \quad (5)$$

Note that in (3), (4), (5), additional object-level integrity constraints are introduced, to be accomplished by the declarative semantics.

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 $\Delta^{\mathcal{A}}$ such that (1), (2), (4) and (5) are satisfied. In this case, we write*

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

The operational counterpart to this declarative semantics is represented by $\mathcal{S}$CIFF proof procedure. $\mathcal{S}$CIFF is a rewriting system that searches a proof tree representing all abductive solutions and whose nodes represent states of the computation. A set of transitions rewrite a node into one or more children nodes. $\mathcal{S}$CIFF inherits the transitions of the IFF proof-procedure [29], and extends it in various directions. We recall the basics of $\mathcal{S}$CIFF; a complete description is in [4], with proofs of soundness, completeness, and termination. An efficient implementation of $\mathcal{S}$CIFF is described in [6].

Each node of the proof is a tuple $T \equiv \langle R, CS, PSIC, \Delta^{\mathcal{A}}\rangle$, where R is the resolvent, CS is the CLP constraint store, PSIC is a set of implications (called *Partially Solved Integrity Constraints*) derived from propagation of integrity constraints, and $\Delta^{\mathcal{A}}$ is the current set of abduced literals. 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 [20];
Logical simplifications other simplifications like (true $\rightarrow A$) $\Leftrightarrow A$, etc.

The $\mathcal{S}$CIFF proof procedure includes also the transitions of CLP [34] for constraint solving.

In this paper we consider the *generative version* of $\mathcal{S}$CIFF, called g-$\mathcal{S}$CIFF [3, 41], in which also the **H** events are considered as abducibles. Although from an ALP point of view the **H** events should be collected together with the other abducibles, we prefer to maintain a notation coherent with that used for the $\mathcal{S}$CIFF proof-procedure, and distinguish the set of abduced events from the set

containing the remaining abducibles, $\Delta^{\mathcal{A}}$. A history **HAP** is provided as input, and further **H** atoms can be assumed like the other abducible predicates; 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* **HAP**, *we say that a goal $\mathcal{G}$ is a g-$\mathcal{S}$CIFF answer if there exist a set $\Delta^{\mathcal{A}}$ and a set* $\mathbf{HAP'} \supseteq \mathbf{HAP}$ *such that*

$$KB \cup \mathbf{HAP'} \cup \Delta^{\mathcal{A}} \models \mathcal{G} \tag{6}$$
$$KB \cup \mathbf{HAP'} \cup \Delta^{\mathcal{A}} \models \mathcal{IC} \tag{7}$$
$$KB \cup \mathbf{HAP'} \cup \Delta^{\mathcal{A}} \models \mathbf{EN}(X) \wedge \mathbf{H}(X) \rightarrow \mathit{false}. \tag{8}$$
$$KB \cup \mathbf{HAP'} \cup \Delta^{\mathcal{A}} \models \mathbf{E}(X) \wedge \mathbf{EN}(X) \rightarrow \mathit{false}. \tag{9}$$

In this case, we write

$$\langle KB, \mathcal{IC}\rangle \models^{g}_{\mathbf{HAP} \rightsquigarrow \mathbf{HAP'}} \mathcal{G} \qquad \text{or simply} \qquad \langle KB, \mathcal{IC}\rangle \models^{g}_{\mathbf{HAP}} \mathcal{G}.$$

3 Norms in $\mathcal{S}$CIFF

In [5], $\mathcal{S}$CIFF was exploited to support legal regulations expressed in a deontic language, named $Deon^{+}$. In $Deon^{+}$, (positive) actions are represented by terms and, as usual in logic programming, terms can contain variables, constants, terms. Building upon this action language, obligations are represented as $\mathcal{S}$CIFF atoms $\mathbf{E}(A, T)$, where A is any action description, and T is a CLP variable, existentially quantified. For instance, the sentence "It is mandatory that John answers me", corresponds to:

$$\exists T \qquad \mathbf{E}(answer(john, me), T)$$

as any reply in any time complies to the obligation, and the obligation will no longer hold after John sends his answer. Note that $\mathbf{H}(answer(john, me), T)$ is different from $\mathbf{E}(answer(john, me), T)$: the first expresses that indeed John answers me (or, in g-$\mathcal{S}$CIFF, that we assume he answers), while the second expresses that he *should* answer, independently from the fact that he actually answers or not.

Prohibitions are represented as atoms $\mathbf{EN}(A, T)$, where again A is any action description, and T is a CLP variable, universally quantified (unless it occurs also in a **H** or **E** atom, in such a case it is existentially quantified). For instance, the sentence "It is forbidden that John smokes", corresponds to:

$$\forall T \qquad \mathbf{EN}(smoke(john), T)$$

because in any time John is not allowed to smoke, and the fact he did not smoke one minute ago does not allow him to smoke now and later on. In general, this is different from $not\mathbf{H}(smoke(john), T)$ (or, equivalently, $\mathbf{H}(smoke(john), T) \rightarrow \mathit{false}$) because the first expresses that he should not smoke, while the second

expresses that he does not smoke. In this work, however, we adopt rule (4) that maps violations to failures, which makes the two equivalent in practice; in other works one can consider recovery actions to violations.

A notable advantage of adopting ALP is that the action language is not limited to the propositional case, as in the examples above, but it can contain variables in its turn, quantified (existentially or universally) as the T variable.

The adoption of CLP variables for representing time adds expressiveness to deontic operators and easily recovers deadlines by constraints over time variables. Constraints imposed on universally quantified variables are considered as quantifier restrictions [16]; a sentence like "It is forbidden that John leaves the meeting before 10" is therefore represented in $Deon^+$ as:

$$\forall T : T < 10 \qquad \mathbf{EN}(leave(john, meeting), T),$$

and it is interpreted (coherently with the semantics of quantifier restrictions) as

$$\forall T, \quad T < 10 \rightarrow \mathbf{EN}(leave(john, meeting), T).$$

Integrity constraints can be also exploited to represent conditional obligatoriness and the deontic logic of deadlines, as shown in [7]. For instance, integrity constraints of the kind $\mathbf{H}(B) \rightarrow \mathbf{E}(A)$ are suitable to represent the obligatoriness of A given B, or Deontic logic with deadlines [15].

4 Datalog$^{\pm}$ Ontologies in $\mathcal{S}$CIFF

W3C has supported the development of a family of knowledge representation formalisms of increasing complexity for defining ontologies, called Web Ontology Language (OWL). DLs were chosen as the logic-based counterpart for the OWL family of languages.

In the Computational Logic realm, more recently, [19, 18] proposed Datalog$^{\pm}$, an extension of Datalog with existential rules for representing lightweight ontologies, encompassing the DL-Lite family, and achieving decidability and tractability [17] under appropriate syntactic conditions.

In short, Datalog$^{\pm}$ extends Datalog by allowing existential quantifiers, the equality predicate and the constant *false* in rule heads. Any Datalog$^{\pm}$ theory may, in fact, include three types of implication rules: Tuple-Generating Dependencies (TGDs), Negative Constraints (NCs) and Equality Generating Dependencies (EGDs), as shown in the following. In standard Datalog, we can represent a rule stating that the father X of any person Y is also a person:

$$\forall X \, \forall Y \; fatherOf(X, Y) \; \wedge \; person(Y) \; \rightarrow \; person(X)$$

In Datalog$^{\pm}$, we get higher expressiveness, and by a TGD rule we can state that any person X has a father Y who is also a person:

$$\forall X \, person(X) \; \rightarrow \; \exists Y \; fatherOf(Y, X) \; \wedge \; person(Y)$$

or that for any person X, and any couple of his/her fathers Y and Z, then Y and Z must be the same, by the following EGD:

$$\forall X \, \forall Y \, \forall Z \, fatherOf(Y, X) \, \wedge \, fatherOf(Z, X) \, \rightarrow \, Y = Z$$

Finally, we can also state, by the following NC, that for any X, his/her father and mother cannot be the same Y:

$$\forall X \, \forall Y \, fatherOf(Y, X) \, \wedge \, motherOf(Y, X) \, \rightarrow \, false$$

Declaratively, given a finite set of relation names $\mathcal{R}$, a Datalog$^{\pm}$ theory T (a set of TGDs, NCs and EGDs) on $\mathcal{R}$, and a database D (a set of ground atoms) 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.

In Datalog$^{\pm}$ the set of answers to a Conjunctive Query (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.

Operationally, Datalog$^{\pm}$ query answering for CQs and Boolean Conjunctive Queries (BCQs) is achieved via the *chase*, a bottom-up algorithm for deriving atoms entailed by a given database D and a Datalog$^{\pm}$ theory. Informally, the chase works on the database D and extends it through the so-called TGD and EGD chase rules. When the body of a TGD is true in the database D, the TGD chase rule adds to D new atomic formulas corresponding to TGD's heads with new (null) variables for the existential ones, that are not already in D. The EGD chase rule, when the body of an EGD is true in the database D, tries to unify the two terms implied in the equality in the EGD's head. A *hard violation* is raised if the unification fails. A more formal description can be found in [18, 17].

In [30, 31] we showed that, by suitably extending the $\mathcal{S}$CIFF proof-procedure, we are able to represent in $\mathcal{S}$CIFF a Datalog$^{\pm}$ program, and to use it for ontological reasoning. $\mathcal{S}$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, which uses both a top-down, backward reasoning from the goal, and a forward reasoning for integrity constraints.

In Datalog$^{\pm}$, tuples can be added to the database through the TGD chase rule, and unifications can be done via the EGD chase rule. As explained earlier, we mimic the chase through the propagation of $\mathcal{S}$CIFF integrity constraints. The $\mathcal{S}$CIFF syntax for integrity constraints is extended to allow for **H** literals in the head of integrity constraints. **H** atoms are now considered as abducible atoms, so that, through the propagation of integrity constraints, they are assumed as true if they occur in the head of a (transformed) TGD. Coherently with both the Datalog$^{\pm}$ and $\mathcal{S}$CIFF syntax, variables that occur only in the head of an IC and that occur in a **H** literal are implicitly existentially quantified.

The finite set of relation names of a Datalog$^{\pm}$ relational schema $\mathcal{R}$ is mapped into the set of terms occurring in the **H** predicates of the corresponding $\mathcal{S}$CIFF program. A Datalog$^{\pm}$ database D for $\mathcal{R}$ corresponds to the (possibly infinite)

$\mathcal{S}$CIFF history **HAP**, since there is a one-to-one correspondence between each tuple in D and each (ground) fact in **HAP**. This mapping may seem unintuitive from an ALP viewpoint, since intensional predicate definitions are mapped into integrity constraints; on the other hand, as will be clear shortly, it lets one reuse the same implications used in a Datalog$^{\pm}$ theory.

In fact, a Datalog$^{\pm}$ theory T is mapped into a $\mathcal{S}$CIFF program with an empty KB, and $IC = \tau(T)$, where ICs have (conjunctions of) **H** atoms and CLP constraints (equalities in particular), or $false$ in their heads, and are obtained by the τ mapping from the original TGDs, EGDs, and NCs.

The τ mapping is recursively defined as follows, where A is an atom, and F_1, F_2, ..., are Datalog$^{\pm}$ formulae:

$$\begin{aligned} \tau(Body \rightarrow Head) &= \tau(Body) \rightarrow \tau(Head) \\ \tau(A) &= \mathbf{H}(A) \\ \tau(F_1 \wedge F_2) &= \tau(F_1) \wedge \tau(F_2) \\ \tau(false) &= false \\ \tau(Y_i = Y_j) &= Y_i = Y_j \\ \tau(\exists \mathbf{X}\ A) &= \mathbf{H}(A) \end{aligned}$$

where the last equation comes from the fact that the quantification for variables that occur only in **H** literals in the head of an IC is always existential, and it is implicit in the $\mathcal{S}$CIFF syntax.

A Datalog TGD $F = body \rightarrow head$ 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 **H** 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.

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

Other possible mappings could be considered; interesting research directions could be to map the database D to the KB, and/or atoms to normal abducible predicates, instead of **H** events.

Given a Datalog$^{\pm}$ theory 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, given a Datalog$^{\pm}$ theory 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, in [30, 31], we have proved that there exists an abductive explanation $\mathbf{HAP'} = \tau(B)$ such that:

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

where $\mathbf{HAP'} \supseteq \mathbf{HAP} = \tau(D)$, and $\mathcal{IC} = \tau(T)$, in which Datalog$^{\pm}$ nulls are mapped to existentially quantified variables.

Therefore, informally speaking, the set of models of D given T, $mods(D, T)$, corresponds to the set of all the abductive explanations $\mathbf{HAP'}$ satisfying the set of $\mathcal{S}$CIFF integrity constraints $\mathcal{IC} = \tau(\mathcal{T})$.

In [30, 31], we have stated and proved theorems for (model-theoretic) completeness of query answering. Informally, we recall here the completeness result for CQ-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 $\mathbf{HAP}'$ for goal $G = \tau(\Phi(\mathbf{X}, \_))$ (where the underscore stands for an unnamed variable) such that:

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

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

The $\mathcal{S}$CIFF proof procedure was proved sound and complete w.r.t. $\mathcal{S}$CIFF declarative semantics in [4], thus for each abductive explanation δ 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) $\delta' \subseteq \delta$, and a computed answer substitution for goal G possibly more general than θ.

Example 1 (Real estate information extraction system in ALP). In [32], the authors present a simple ontology for a real estate information extraction system[3]:

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

The TGDs F_1-F_8 from the Datalog$^{\pm}$ ontology above are one-to-one mapped into the following $\mathcal{S}$CIFF ICs:[4]

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

[3] The universal quantifiers are usually left implicit.

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

$IC_7 : \mathbf{H}(loc(X, L1)), \mathbf{H}(loc(X, L2)) \rightarrow L1 = L2$
$IC_8 : \mathbf{H}(loc(X, L)) \rightarrow \mathbf{H}(advertised(X))$

The database in [32] is mapped into the following history **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:

$$\mathbf{HAP'} = \{\mathbf{H}(priceElem(e1)), \mathbf{H}(forSale(prop1)), \exists P\ \mathbf{H}(price(prop1, P)),$$
$$\mathbf{H}(loc(prop1, summertown)), \mathbf{H}(advertised(prop1))\}$$

Each of the (ground) atomic queries (BCQs) outlined in [32] is also entailed in the $\mathcal{S}$CIFF program above. In particular, for the previous set $\mathbf{HAP'}$:

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

Also, the CQ $\exists L\ \mathbf{H}(loc(prop1, L))$ is entailed as well (with unification $L = summertown$, as in [32]) since:

$$\mathbf{HAP'} \models \mathbf{H}(loc(prop1, summertown))$$

4.1 Related work

A very related approach for mapping DL theories into ALP is contained in [48]. The authors consider $\mathcal{ALCN}$ Description Logics theories, i.e., ontologies where the terminological part consists of concept definitions of kind:

$$C \equiv F$$

where C is a concept symbol and F is a (non recursive) concept description. Given that R is a role, C a concept symbol and F, G concepts descriptions, valid concept descriptions are the terms:

$$C \mid \forall R.F \mid \exists R.F \mid F \cap G \mid F \cup G \mid \neg F \mid \leq nR \mid \geq nR$$

Any concept symbol C occurring in a concept definition of kind $C \equiv F$ is named a defined concept, otherwise it is named primitive.

Their mapping transforms concept definitions of such a kind into Open Logic Programming clauses [21], an extension of ALP, and they prove that this operation is equivalence preserving. They map concept definitions to program clauses, first translating the definitions into *general clauses*, where the head is an atom and the body any First Order Logic expression. Then, general clauses are transformed into a set of Horn clauses, by the Lloyd-Topor transformation [39].

Unluckily, in their mapping, they lose half of the definition. A concept definition of kind $C \equiv F$ is, in fact, transformed into a general clause of kind:

$$C(X) \leftarrow T'(F, X)$$

where T' is a mapping transformation inductively defined over the syntax of F (see above).

For instance, given the ontological definition stating that a father is any male person having at least one child:

$$Father \equiv Male \cap Person \cap \exists child.Person$$

their mapping produces only the clause:

$$father(X) \leftarrow male(X), person(X), child(X, Y), person(Y)$$

thus losing the second part of the definition, which corresponds to:

$$father(X) \rightarrow male(X), person(X), \exists Y(child(X, Y), person(Y)).$$

In this sense, even if [48] captures the intuition that DL theories, with the Open World Assumption approach, have much in common with Abductive or Open Logic Programming, the mapping and representation they provide do not support the notion of *concept definition* which is peculiar of terminological systems, and which lets one reason both ways when considering a definition $C \equiv F$.

The major issue for mapping DL theories into Logic Programming languages is exactly the need to represent implications having in their heads existentially quantified variables (as the example above outlines). Datalog$^{\pm}$ is, instead, a logic programming language enriched with implications having also existentially quantified variables in their heads. This feature is fundamental to fully represent even the simplest $\mathcal{ALC}$ Description Logic.

Moreover, Datalog$^{\pm}$ conflates logic programming clauses, forward rules and integrity constraints, thus proving an integration of ontological reasoning with rule-based programing, in a single language.

As well, $\mathcal{S}$CIFF is a language naturally providing the same syntax extension, and smoothly integrating into ALP both ontological representation and rule-based programming.

5 Integrating norms and ontologies in $\mathcal{S}$CIFF

After mapping a Datalog$^{\pm}$ ontology into $\mathcal{S}$CIFF, we are now ready to show how normative reasoning is smoothly integrated with ontological reasoning within the $\mathcal{S}$CIFF abductive framework. We will show this via a simple example, starting from the real estate ontology, and enriching it with normative rules for some interacting agents in a real estate scenario.

As discussed in Section 3, the set of regulations holding in the given society of agents can be expressed again through integrity constraints.

In the previous example, suppose that a real estate agent (rea) is in charge of selling a property, and it uses the data from the real estate information extracted from the ontological knowledge. If another agent has seen an announcement that a property (e.g., a flat) is for sale, it can request to buy it. In such a case, for

some fixed time ΔT, the real estate agent is obliged to reserve the flat for that client. For ΔT time units, the real estate agent cannot sell the flat to other agents and, if the buyer issues a payment, the flat must be declared sold (within some deadline D). The expected behavior of the real estate agent can be defined by the following rules:

$$\mathbf{H}(tell(X, rea, buy(E), T), \mathbf{H}(forSale(E)) \\ \rightarrow \mathbf{EN}(sell(rea, Y, E), T_s), Y \neq X, T_s < T + \Delta T$$

$$\mathbf{H}(tell(X, rea, buy(E), T), \mathbf{H}(forSale(E)), \mathbf{H}(price(E, P)), \\ \mathbf{H}(pay(X, rea, P), T_p), T_p < T + \Delta T \rightarrow \mathbf{E}(sell(rea, X, E), T_s), T_s < T_p + D.$$

If some agent e asks to buy property $prop1$ at price $e1$ at time 1, i.e., $\mathbf{H}(tell(e, rea, buy(prop1), 1)$, the proof procedure is able to infer the following information about the expected behavior of the rea agent:

$$\forall T_s \text{ s.t. } T_s < 1 + \Delta T, \ \forall Y \neq e \qquad \mathbf{EN}(sell(rea, Y, prop1), T_s)$$

i.e., agent rea is not allowed to sell property $prop1$ to any other agent until ΔT time units have passed. Let us suppose that $\Delta T = 5$ and $D = 3$; if agent e actually executes the payment, e.g. $\mathbf{H}(pay(e, rea, e1), 3)$, agent rea is now expected to sell $prop1$, and the following expectation is raised:

$$\exists T_s \text{ s.t. } T_s < 6 \qquad \mathbf{E}(sell(rea, e, prop1), T_s).$$

In this way, not only the $\mathcal{S}$CIFF proof procedure is able to infer the knowledge from the ontological database, but also to provide the expected behavior of the agents (in our example, the rea agent) including obligations and prohibitions.

6 Conclusions and Future Work

We have shown that Abductive Logic Programming (ALP) is a powerful tool for knowledge representation and reasoning about norms and ontologies. We have focused in detail about the $\mathcal{S}$CIFF ALP framework, used in the past to model and verify agent societies, interaction protocols for multi-agent systems [8], Web services choreographies [1], but also powerful enough to represent deontic operators [7] and normative systems [5]. Its underlying $\mathcal{S}$CIFF proof procedure [4], derived from the IFF one, considers, in rule heads, atoms that can contain existentially and universally quantified variables, as well as CLP constraints. Recently [30, 31], $\mathcal{S}$CIFF has been also proved useful for representing ontologies expressed in Datalog$^{\pm}$.

In this work, we have exploited the $\mathcal{S}$CIFF framework for representing and integrating both norms and Datalog$^{\pm}$ ontologies. Normative, rule-based reasoning and ontological query answering are obtained by applying the $\mathcal{S}$CIFF abductive proof procedure. Both norms and a Datalog$^{\pm}$ theory can be encoded as $\mathcal{S}$CIFF integrity constraints. The main advantage is that this integration is

achieved within a single language, grounded on ALP. Nonetheless, different ALP approaches might be possible.

A number of issues are subject of future work. First of all, we have not focused here on complexity results. Identifying syntactic conditions that guarantee tractable ontologies in $\mathcal{S}$CIFF, in the style of what has been done for Datalog$^{\pm}$, is crucial for achieving nice computational performance.

A second issue for future work concerns experimentation with real cases, in the normative and legal domain, and on real-size ontologies.

Finally, different mappings of Datalog$^{\pm}$ to ALP might exist, possibly enjoying different properties: another research direction is oriented toward identifying the mapping that suits best for legal reasoning.

Acknowledgements. We would like to thank the anonymous referees for their helpful comments, and JURISIN2015 participants for questions and discussions during the workshop. The second author would like to thank NII, Tokyo (J), for supporting her travel to Japan and for making possible to attend JURISIN2015.

References

1. 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. (ed.) Proceedings of the Eighth Symposium on Principles and Practice of Declarative Programming. pp. 39–50. ACM Press, New York, USA (Jul 2006)
2. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Compliance verification of agent interaction: a logic-based software tool. Applied Artificial Intelligence 20(2-4), 133–157 (Feb-Apr 2006)
3. 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, M.P., Ricci, A. (eds.) ESAW 2005 Post-proceedings. pp. 106–124. No. 3963 in LNAI, Springer-Verlag (2006)
4. 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 9(4) (2008)
5. Alberti, M., Gavanelli, M., Lamma, E.: Deon+ : Abduction and constraints for normative reasoning. In: Artikis, A., Craven, R., Cicekli, N.K., Sadighi, B., Stathis, K. (eds.) Logic Programs, Norms and Action - Essays in Honor of Marek J. Sergot on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 7360, pp. 308–328. Springer (2012)
6. Alberti, M., Gavanelli, M., Lamma, E.: The CHR-based implementation of the SCIFF abductive system. Fundamenta Informaticae 124(4), 365–381 (2013)
7. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Sartor, G., Torroni, P.: Mapping deontic operators to abductive expectations. Computational and Mathematical Organization Theory 12(2–3), 205 – 225 (Oct 2006)
8. 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 85(2) (2003)
9. Alberti, M., Gomes, A.S., Gonçalves, R., Leite, J., Slota, M.: Normative systems represented as hybrid knowledge bases. In: Leite, J., Torroni, P., Ågotnes, T., Boella, G., van der Torre, L. (eds.) CLIMA XII. LNAI, vol. 6814 (2011)

10. ALFEBIITE: A Logical Framework for Ethical Behaviour between Infohabitants in the Information Trading Economy of the universal information ecosystem. IST-1999-10298 (1999)
11. Arisha, K.A., Ozcan, F., Ross, R., Subrahmanian, V.S., Eiter, T., Kraus, S.: IMPACT: a Platform for Collaborating Agents. IEEE Intelligent Systems 14(2) (1999)
12. Boella, G., van der Torre, L., Verhagen, H.: Introduction to normative multiagent systems. Computational and Mathematical Organization Theory 12, 71–79 (2006)
13. Boella, G., van der Torre, L.: Attributing mental attitudes to normative systems. In: Rosenschein, J.S., Sandholm, T., Wooldridge, M., Yokoo, M. (eds.) Proc AAMAS-2003. pp. 942–943. ACM Press (Jul 14–18 2003)
14. Bracciali, A., Demetriou, N., Endriss, U., Kakas, A.C., Lu, W., Mancarella, P., Sadri, F., Stathis, K., Terreni, G., Toni, F.: The KGP model of agency for global computing: Computational model and prototype implementation. In: Priami, C., Quaglia, P. (eds.) Global Computing. LNCS, vol. 3267. Springer (2004)
15. Broersen, J., Dignum, F., Dignum, V., Meyer, J.J.C.: Designing a deontic logic of deadlines. In: Lomuscio, A., Nute, D. (eds.) DEON. Lecture Notes in Computer Science, vol. 3065, pp. 43–56. Springer (2004)
16. Bürckert, H.: A resolution principle for constrained logics. Artificial Intelligence 66, 235–271 (1994)
17. 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. pp. 70–80. AAAI Press (2008)
18. Calì, A., Gottlob, G., Lukasiewicz, T.: A general Datalog-based framework for tractable query answering over ontologies. In: Symposium on Principles of Database Systems. pp. 77–86. ACM (2009)
19. Calì, A., Gottlob, G., Lukasiewicz, T.: Tractable query answering over ontologies with Datalog$^{\pm}$. In: International Workshop on Description Logics. CEUR Workshop Proceedings, vol. 477. CEUR-WS.org (2009)
20. Clark, K.L.: Negation as Failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 293–322. Plenum Press (1978)
21. Console, L., Theseider Dupré, D., Torasso, P.: On the relationship between abduction and deduction. Journal of Logic and Computation 1(5), 661–690 (1991)
22. De Vos, M., Padget, J., Satoh, K.: Legal modelling and reasoning using institutions. In: Onada et al. [42], pp. 129–140
23. Denecker, M., De Schreye, D.: SLDNFA: an abductive procedure for abductive logic programs. Journal of Logic Programming 34(2), 111–167 (1998)
24. Dignum, V., Meyer, J.J., Weigand, H.: Towards an organizational model for agent societies using contracts. In: Castelfranchi, C., Lewis Johnson, W. (eds.) AAMAS-2002. pp. 694–695. ACM Press, Bologna, Italy (Jul 15–19 2002)
25. Dignum, V., Meyer, J.J., Weigand, H., Dignum, F.: An organizational-oriented model for agent societies. In: Proc. International Workshop on Regulated Agent-Based Social Systems: Theories and Applications. AAMAS'02, Bologna (2002)
26. Du, J., Wang, K., Shen, Y.D.: A tractable approach to ABox abduction over description logic ontologies. In: Brodley, C., Stone, P. (eds.) Proc. AAAI 2014 (2014)
27. Eiter, T., Subrahmanian, V., Pick, G.: Heterogeneous active agents, I: Semantics. Artificial Intelligence 108(1-2), 179–255 (March 1999)
28. ESTRELLA: European project for Standardized Transparent Representations in order to Extend Legal Accessibility. IST-2004-027655 (2004), home Page: `http://www.estrellaproject.org`
29. Fung, T.H., Kowalski, R.A.: The IFF proof procedure for abductive logic programming. Journal of Logic Programming 33(2), 151–165 (Nov 1997)

30. Gavanelli, M., Lamma, E., Riguzzi, F., Bellodi, E., Zese, R., Cota, G.: An abductive framework for Datalog$^{\pm}$ ontologies. In: Eiter, T., Toni, F. (eds.) Technical Communications of the 31st Int'l. Conference on Logic Programming (ICLP 2015). CEUR Workshop Proceedings, Sun SITE Central Europe, Aachen, Germany (2015)
31. Gavanelli, M., Lamma, E., Riguzzi, F., Bellodi, E., Zese, R., Cota, G.: Abductive logic programming for Datalog$^{\pm}$ ontologies. In: Ancona, D., Maratea, M., Mascardi, V. (eds.) Proceedings of the 30th Italian Conference on Computational Logic (CILC2015), Genova, Italy, July 1-3, 2015. CEUR Workshop Proceedings, vol. 1459, pp. 128–143. Sun SITE Central Europe, Aachen, Germany (2015)
32. Gottlob, G., Lukasiewicz, T., Simari, G.I.: Conjunctive query answering in probabilistic Datalog+/- ontologies. In: International Conference on Web Reasoning and Rule Systems. LNCS, vol. 6902, pp. 77–92. Springer (2011)
33. Governatori, G., Rotolo, A.: BIO logical agents: Norms, beliefs, intentions in defeasible logic. Autonomous Agents and Multi-Agent Systems 17(1), 36–69 (2008)
34. Jaffar, J., Maher, M.: Constraint logic programming: a survey. Journal of Logic Programming 19-20, 503–582 (1994)
35. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive Logic Programming. Journal of Logic and Computation 2(6), 719–770 (1993)
36. Kakas, A.C., Mancarella, P.: On the relation between Truth Maintenance and Abduction. In: Fukumura, T. (ed.) Proc. PRICAI-90. Ohmsha Ltd. (1990)
37. Kakas, A.C., van Nuffelen, B., Denecker, M.: $\mathcal{A}$-System: Problem solving through abduction. In: Nebel, B. (ed.) Proc. IJCAI-01 (2001)
38. Klarman, S., Endriss, U., Schlobach, S.: ABox abduction in the description logic $\mathcal{ALC}$. J. Autom. Reasoning 46(1), 43–80 (2011)
39. Lloyd, J., Topor, R.: Making Prolog more expressive. Journal of logic programming 1(3), 225–240 (1984)
40. Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag (1987)
41. Montali, M., Torroni, P., Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P.: Verification from declarative specifications using logic programming. In: de la Banda, M.G., Pontelli, E. (eds.) Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings. Lecture Notes in Computer Science, vol. 5366, pp. 440–454. Springer (2008)
42. Onada, T., Bekki, D., McCready, E. (eds.): New Frontiers in Artificial Intelligence - JSAI-isAI 2010 Workshops, LNCS, vol. 6797. Springer (2011)
43. Prakken, H., Sartor, G.: Argument-based extended logic programming with defeasible priorities. Journal of Applied Non-Classical Logics 7(1) (1997)
44. Sadri, F., Stathis, K., Toni, F.: Normative KGP agents. Computational & Mathematical Organization Theory 12(2-3), 101–126 (2006)
45. Sartor, G., Casanovas, P., Biasiotti, M.A., Fernández-Barrera, M. (eds.): Approaches to Legal Ontologies. Law, Governance and Technology, Springer (2011)
46. Satoh, K., Asai, K., Kogawa, T., Kubota, M., Nakamura, M., Nishigai, Y., Shirakawa, K., Takano, C.: PROLEG: an implementation of the presupposed ultimate fact theory of Japanese civil code by PROLOG technology. In: Onada et al. [42]
47. Sergot, M.J., Sadri, F., Kowalski, R.A., Kriwaczek, F., Hammond, P., Cory, H.T.: The British Nationality Act as a logic program. Commun. ACM 29, 370–386 (1986)
48. Van Belleghem, K., Denecker, M., De Schreye, D.: A strong correspondence between description logics and open logic programming. In: Naish, L. (ed.) Proc. Fourteenth International Conference on Logic Programming. MIT Press (1997)
49. Wright, G.: Deontic logic. Mind 60, 1–15 (1951)